Android Things y visión artificial

- Internet de las cosas
- OpenCV
- Código nativo
- Ingeniería inversa

Jesús Tomás
Antonio Albiol
Miguel García
Salvador Santonja

Alfaomega
Marcombo

Android Things y visión artificial

Android Things y visión artificial

Jesús Tomás
Antonio Albiol
Miguel García
Salvador Santonja

Alfaomega

Marcombo

Diseño de la cubierta: ENEDENÚ DISEÑO GRÀFICO

Datos catalográficos
Tomás, Jesús; Albiol, Antonio; García, Miguel; Santonja, Salvador
Android Things y visión artificial
Primera Edición
Alfaomega Grupo Editor, S.A. de C.V., México
ISBN: 978-607-538-412-2
Formato: 17 x 23 cm Páginas: 440

Android Things y visión artificial
Jesús Tomás, Antonio Albiol, Miguel García y Salvador Santonja
ISBN: 978-84-267-2666-7, de la edición publicada por MARCOMBO, S.A., Barcelona, España

Primera edición: Alfaomega Grupo Editor, México, enero 2019

Pitágoras 1139, Col. Del Valle, 03100, México D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Pág. Web: **http://www.alfaomega.com.mx**
E-mail: **atencionalcliente@alfaomega.com.mx**

ISBN: 978-607-538-412-2

Nota importante:
La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Edición autorizada para venta en México y todo el continente americano.

Impreso en México. Printed in Mexico.

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. – Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, Ciudad de México – C.P. 06720. Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396 – E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. – Calle 62 No. 20-46, Barrio San Luis, Bogotá, Colombia, Tels.: (57-1) 746 0102 / 210 0415 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Av. Providencia 1443. Oficina 24, Santiago, Chile Tel.: (56-2) 2235-4248 – Fax: (56-2) 2235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Av. Córdoba 1215, piso 10, CP: 1055, Buenos Aires, Argentina, – Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

Para Adela con el amor de su hermano.

JESÚS TOMÁS

A Inma, Laura y Mónica.

ANTONIO ALBIOL

A toda la gente que me ha apoyado en esta última etapa,
en especial a Diana por su cariño y ayuda constante.

MIGUEL GARCÍA PINEDA

A Raquel y Adrián, mi razón de todo

SALVADOR SANTONJA

Índice

¿Cómo leer este libro?

Este libro aborda cuatro temáticas: Visión Artificial, Internet de las cosas, Código nativo e Ingeniería inversa, todas ellas utilizando el sistema operativo Android. No es precisa una lectura secuencial, el lector puede ir directamente al capítulo que le interese.

En este libro se da por supuesto que el lector tiene experiencia en programación sobre Android. No se tratan temas relativos al desarrollo básico como los componentes o estructura de las aplicaciones. Si el lector está interesado en un texto que aborde la programación en Android desde el principio, le recomendamos "El gran libro de Android" publicado en esta misma editorial. También puede ser de interés la lectura de otros libros de esta colección como "El gran libro de Android Avanzado", "Plataformas Android: Wear, TV, Auto y Google Play Games" o "Firebase: Trabajando en la nube".

El libro que tienes entre las manos no ha sido concebido solo para ser leído. Es más bien una guía estructurada que te irá proponiendo una serie de ejercicios, actividades, vídeos explicativos, test de autoevaluación, etc. Parte de este material y algunos recursos adicionales están disponibles en la web www.androidcurso.com. En ella se publicarán las novedades, erratas e información complementaria relativas a este libro. Además, encontrarás:

Material adicional sobre Android: Encontrarás, además, nuevos tutoriales, vídeos, referencias, etc., no incluidos en el libro.

Código abierto de proyectos Android: Muchos alumnos que han realizado cursos con nosotros han tenido la generosidad de compartir sus proyectos. Te recomendamos que consultes la lista de proyectos disponibles de código abierto: puedes aprender mucho estudiando su código.

Cursos online: Si te interesa ampliar tu formación, puedes matricularte en cursos sobre Android impartidos por la Universidad Politécnica de Valencia. Incluso puedes obtener un título de Especialización o de Máster de forma 100% online.

A lo largo del libro se utilizan los siguientes iconos para indicar distintos tipos de recursos:

Objetivos: Antes de empezar cada capítulo lee con detenimiento la introducción y los objetivos.

Ejercicio: La mejor forma de aprender es haciendo. No tendrás más que ir siguiendo los pasos uno tras otro para descubrir cómo se resuelve el ejercicio propuesto. Para que no se te haga pesado teclear todo el código, te proponemos que lo copies y pegues desde la página web del curso.

Práctica: Este será el momento de que tomes la iniciativa y trates de resolver el problema que se propone. Recuerda que para aprender hay que practicar.

Solución: Te será de ayuda si tienes problemas al resolver una práctica o simplemente quieres comparar tu solución con otra diferente.

Vídeo[Tutorial]: Vídeos públicos donde se exponen aspectos complementarios que pueden ayudar una mejor comprensión de los temas tratados.

Enlaces de interés: Internet te será de gran ayuda para completar la información necesaria para programar en Android. Te proponemos las páginas más interesantes de cada apartado.

Preguntas de repaso: ¿Has comprendido correctamente los aspectos clave? Sal de dudas haciendo los test de autoevaluación.

Trivial programación Android: Instálate esta app y mide tus conocimientos jugando en red contra otros oponentes.

PARTE 1.

Visión artificial en Android usando OpenCV

Por ANTONIO ALBIOL

CAPÍTULO 1.

Análisis de imagen en Android usando OpenCV

Por ANTONIO ALBIOL

Hoy en día la mayoría de los dispositivos Android disponen de al menos una cámara. El uso más inmediato de la cámara consiste en la realización de fotografías o vídeos con el fin de ser almacenados o enviados a otra persona.

Existen, no obstante, algunas aplicaciones que hacen uso de la cámara como un dispositivo de entrada o sensor adicional. Algunos ejemplos de este tipo de aplicaciones serían: lecturas de **códigos QR** o códigos de barras; aplicaciones de **realidad aumentada**, normalmente orientadas al campo de los juegos o la educación donde se superponen elementos creados de forma sintética sobre las imágenes de la cámara; **reconocimiento de gestos**: por ejemplo, permitiendo avanzar las páginas de un documento sin tocar la pantalla mediante el paso de la mano; **verificación facial**, usada como método de desbloqueo del teléfono.

Figura 1. Ejemplo de imagen de realidad aumentada. La imagen del camaleón se superpone a la imagen de la mano con el marcador, que es la captada por la cámara.

Incluso las aplicaciones de tipo «cámara» de la mayoría de dispositivos muchas veces realizan un procesamiento de las imágenes con el fin de generar ciertos efectos u optimizar la toma de imágenes. Entre las opciones que son usuales

hoy en día tendríamos: **detección facial** con el fin de determinar la exposición y enfoque óptimos en las zonas de la imagen correspondientes a las caras; detección de **ojos cerrados**; creación de **vistas panorámicas** a partir de una secuencia de imágenes y creación de **imágenes HDR** o de alto margen dinámico. Estas imágenes se obtienen combinando de forma inteligente varias fotografías convencionales con distintos tiempos de exposición y tomadas en rápida sucesión, con el fin de poder captar adecuadamente todos los matices de una escena donde la diferencia de luminosidad entre la parte más brillante y la más oscura de la imagen sea muy grande.

En este módulo, veremos los principios y cómo realizar aplicaciones que llevan a cabo el análisis de imágenes capturadas por la cámara en tiempo real de un dispositivo Android.

Objetivos:

- Instalar la librería OpenCV para Android y ser capaz de desarrollar aplicaciones que la usen.
- Conocer cómo se codifican las imágenes digitales
- Comprender la estructura de un proyecto que analiza imágenes capturadas de modo continuo por la cámara.
- Configurar aspectos básicos de la captura, como la entrada, la resolución, la orientación, el color
- Aprender a utilizar la clase Mat de OpenCV.

1.1. Introducción

Con el fin de centrar las ideas, seguiremos un hilo conductor que tendrá como objetivo realizar una aplicación capaz de reconocer en tiempo real señales de tráfico de limitación de velocidad. La idea es similar a la que implementan algunos coches modernos que alertan al conductor de las limitaciones de velocidad a partir de las imágenes. La ventaja es que, en vez de requerir una instalación en el vehículo, nos serviremos de la cámara del teléfono para capturar y procesar las imágenes.

Desde el punto de vista de análisis de imagen, la aplicación mínima tendría que:

- Analizar continuamente las imágenes de la cámara en busca de señales de limitación de velocidad.
- En el caso de que se detecten señales, mostrar el valor numérico de la limitación, o nada en caso contrario.

Dicha aplicación mínima se podría mejorar bastante, por ejemplo, midiendo la velocidad del vehículo usando el GPS y mostrando advertencias solo en caso de que se esté superando el límite de velocidad, o mediante el uso de síntesis de voz que «lea» la cifra a través del altavoz para no distraer la atención de la conducción, o mediante el sensor de orientación para saber si el teléfono está en orientación vertical o apaisado.

Para ser capaz de realizar una aplicación como la que acabamos de mencionar son necesarios dos aspectos:

- Entender los conceptos de análisis de imagen mínimos que permitan alcanzar el objetivo.
- Ser capaz de llevar a la práctica dichos conceptos. Para ello será necesario saber cómo hacer lo siguiente:
    - Capturar imágenes de las cámaras de los dispositivos.
    - Visualizar imágenes antes y después de ser procesadas.
    - Analizar las imágenes en tiempo real mediante los algoritmos adecuados.

El análisis de imagen, incluso en el caso de usar imágenes de poca resolución, es una tarea costosa computacionalmente. Existen dos opciones para poder acometer la tarea con éxito en dispositivos con una potencia de cálculo limitada, como son los dispositivos Android:

- Utilización de código nativo habitualmente programado en C++.
- Uso de librerías precompiladas que permiten realizar operaciones sobre imágenes. Este enfoque es el que seguiremos nosotros.

La librería de análisis de imágenes que emplearemos será OpenCV (http://opencv.org). Dicha librería es de dominio público, está relativamente bien documentada y además está disponible para Android (además de otros lenguajes como C, C++ o python).

1.2. Instalación de OpenCV para Android

1.2.1. Prerrequisitos

Antes de explicar cómo instalar OpenCV para Android, supondremos que en el ordenador en que vayamos a trabajar, tenga este cualquier sistema operativo (Windows, Mac, Linux), se habrá instalado:

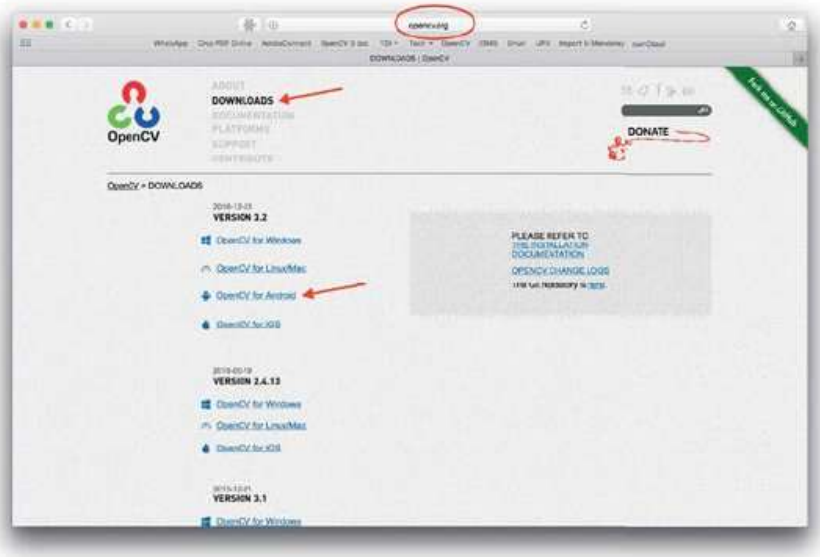
- El entorno de desarrollo Android Studio.

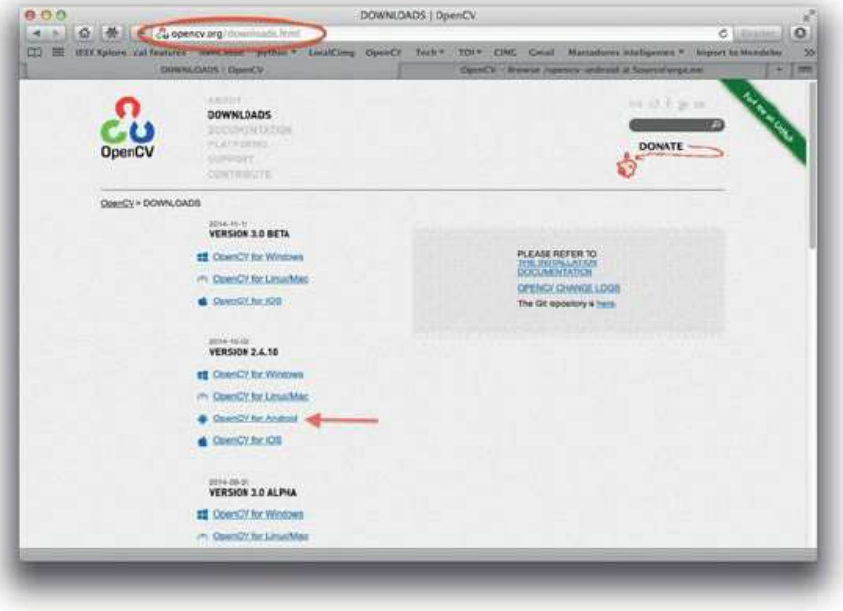
1.2.2. Instalación de la librería OpenCV y los ejemplos

Existen distintas opciones para instalar OpenCV para Android. Una de ellas consiste en bajarse las fuentes y compilarlos. La segunda, más sencilla y que es la que veremos aquí, consiste en bajarse la librería precompilada.

Ejercicio: Instalación de la librería OpenCV y los ejemplos

1. Iremos a la dirección http://opencv.org/downloads.html y descargaremos la última versión estable, que a la fecha de escribir el libro era la 3.2.0.





2. Una vez descargado, descomprimiremos el archivo y obtendremos una carpeta que en adelante denominaremos <opencv>. Dicha carpeta se puede situar donde se quiera.

1.2.3. Prueba en el dispositivo

La librería trae algunos ejemplos que podemos probar en nuestro dispositivo para comprobar que tenemos el entorno funcionando correctamente. Se desaconseja el uso del simulador.

Nos aseguraremos que nuestro dispositivo está en modo «desarrollador». Este paso debería estar hecho si previamente ya has desarrollado algún programa en el dispositivo. Instalaremos en nuestro dispositivo una aplicación llamada OpenCV Manager. Puedes encontrar esta aplicación en Google Play Store.

En enero de 2016 el programa OpenCV Manager que se descargaba de Google Play Store tenía algunos bugs. Se recomienda ir a la Carpeta de OpenCV que nos hemos descargado, y dentro de la carpeta 'apk' instalar OpenCV Manager para nuestro hardware.

Seguidamente podemos probar a ejecutar alguno de los ejemplos que trae la librería.

Para concluir este punto, simplemente quería recordar que una buena manera de aprender es observar el código de estos ejemplos.

Ejercicio: Prueba de los ejemplos en el dispositivo

Para ello:

1. Creamos un nuevo proyecto en AndroidStudio.
2. Nombre: DemosOpenCV.
3. MinimumSDK: API 9.
4. Add No activity.
5. File/New/ImportModule:<opencv>/samples/tutorial-1-camerapreview. Nos indicará que precisa importar también la librería. Pulsamos Next-Finish.
6. Abrir el archivo build.gradle del módulo openCVLibrary320 así:

```
apply plugin: 'com.android.library'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.0"
```

```
    defaultConfig {
        minSdkVersion 9
        targetSdkVersion 25
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.txt'
        }
    }
}
```

7. Nos aseguraremos de que nuestro dispositivo está en modo «desarrollador». Si has ejecutado alguna aplicación que hayas desarrollado, probablemente este paso ya lo habrás dado.
8. Instalaremos "OpenCV Manager" como se ha explicado anteriormente.
9. Conectamos nuestro dispositivo con el cable USB y ejecutamos el programa.
10. Si el programa se detiene indicando que no tiene permisos para usar la cámara, dárselo desde los Ajustes del Dispositivo: Ajustes/Aplicaciones/ OCV T1 Preview.

Enlaces de interés:

Sobre OpenCV en general:

http://opencv.org

Sobre desarrollo de programas en J ava usando OpenCV:

http://docs.opencv.org/java

Sobre información de OpenCV para Android, incluidos tutoriales:

http://opencv.org/platforms/android.html

Preguntas de repaso: Instalación OpenCV

1.3. Imágenes digitales

Una imagen no es más que una representación de la intensidad luminosa en función de la posición f(x,y). En las imágenes digitales, las coordenadas espaciales (x,y) son números enteros, y el valor que suele tomar la intensidad también es un entero que normalmente está en el intervalo 0-255.

Una imagen digital monocroma, de anchura W píxeles y altura H píxeles, se podrá representar como una matriz de WxH números de 1 byte (8 bits por píxel).

1.3.1. Imágenes en color

Las imágenes proporcionadas por las cámaras de los dispositivos Android son normalmente en color. Si la imagen es en color, en cada punto (x,y) tendremos tres cantidades correspondientes a cada una de las componentes de color, R, G y B. Esta información se puede ver de dos maneras:

- Como una matriz de «tripletas RGB». A esto se le denomina formato empaquetado.
- Como tres imágenes monocromas. A esto se le denomina formato planar.

Dependiendo de la manera en que se organice la información, los datos en memoria estarán en un orden u otro. Recordemos que la memoria tiene un espacio de direcciones lineal, y que aunque nosotros pensemos en matrices, realmente los píxeles estarán almacenados de manera consecutiva. Las siguientes figuras ilustran el orden en que estarán los píxeles en memoria en los dos formatos mencionados:

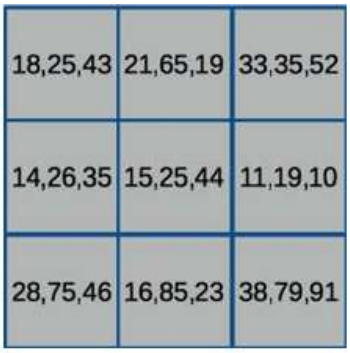


Figura 2. Imagen en color de 3 x 3 píxeles como matriz de tripletas (empaquetado). El orden de los datos en memoria será: 18,25,43; 21,65,19; ...; 14,26,35; 15,25,44; ...; 16,85,23; 38,79,91.

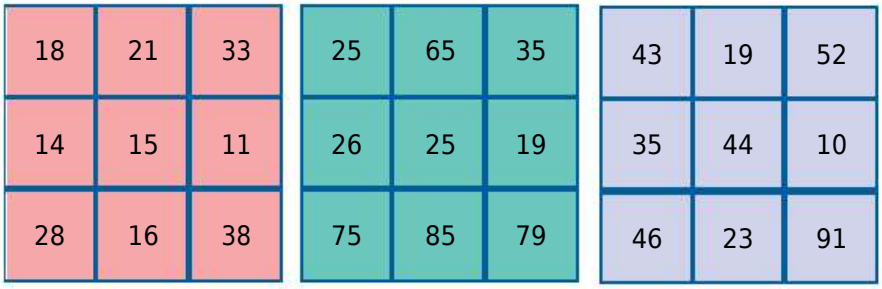


Figura 3. Imagen en color de 3 x 3 píxeles en formato planar. Cada cuadrado representa una componente de color R, G, o B. El orden de los datos en memoria será 18,21,33; 14,15,11; ...; 28,16,38; 25,65,35; ...; 75,85,79; 43,19,52; ...,91

Frecuentemente, en vez de describir el color con RGB, se emplea lo que se denomina otro «espacio de color». Un espacio de color no es más que otra repre-

sentación del color de un píxel con tres números, diferente de RGB y que ofrece alguna ventaja. Normalmente, es posible cambiar la representación de una imagen en color entre distintos espacios de color sin pérdidas. Un espacio de color que se suele emplear es el denominado YUV en el que

- Y=0,3 R + 0,59 G + 0,11 B
- U= 0,49 (B-Y)
- V= 0,88 (R-Y)

En este nuevo espacio de color Y recibe el nombre de luminancia, y está relacionado con la sensación de luminosidad independientemente del color. La componente Y es como la versión monocroma de una imagen en color. Los componentes U y V pueden tomar valores positivos y negativos, y tienen que ver con la tonalidad del color (rojo, azul, verde ...). A los valores U y V se les denomina crominancia. A los valores de U a veces también se les llama Cb y a los de V, Cr. Finalmente indicaremos que los distintos tonos de gris tienen un valor nulo de U y V.

1.3.2. Imágenes en Android

Las imágenes, tal y como las entrega la cámara de un dispositivo Android, están en un formato híbrido entre empaquetado y planar. Dicho formato recibe el nombre de NV21 o también YUV420sp (semi-planar). Este formato se representa esquemáticamente en la figura siguiente. En este formato primero vienen todos los píxeles de la luminancia. Seguidamente vienen los píxeles de la crominancia, pero de forma alternada, uno de Cb, otro de Cr y así sucesivamente. Para terminar de complicar las cosas, resulta que la crominancia tiene una resolución menor que la luminancia, es decir que tenemos un pixel de Cb y otro de Cr por cada 2x2 píxeles de luminancia.

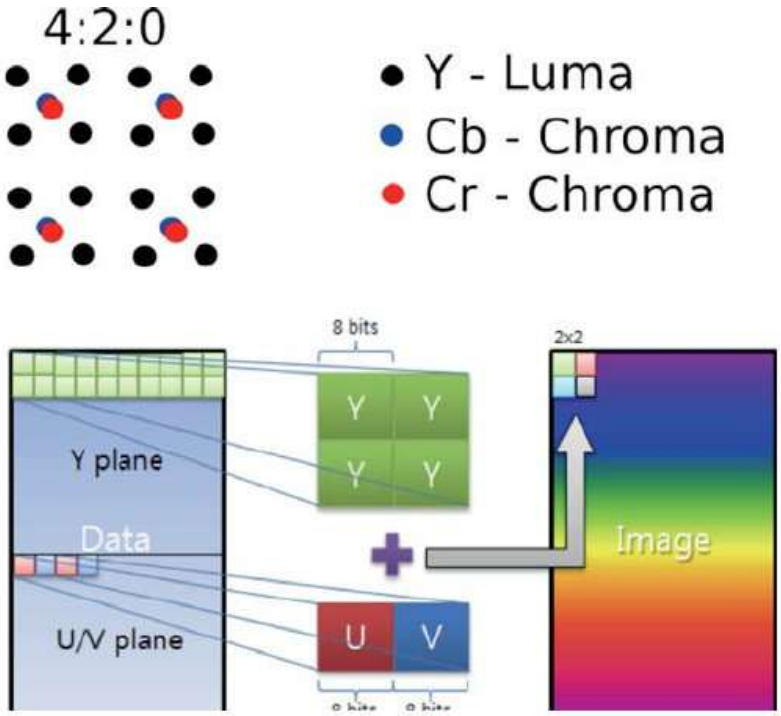


Figura 4. Representación esquemática del formato NV21, usado en las cámaras de los dispositivos Android.

Por tanto, para obtener una imagen RGB clásica a partir de una en formato NV21 se deben hacer ciertas operaciones que deberá realizar la CPU de nuestro dispositivo.

Obsérvese que si nuestro algoritmo pudiera trabajar con imágenes de grises, la obtención de la imagen de gris es inmediata, pues basta con referirse a la zona de memoria adecuada, ya que la luminancia está contigua. En otras palabras, la zona de la luminancia de una imagen NV21 es idéntica a una imagen monocromática simple.

1.4. Programas básicos con OpenCV en Android

En este apartado veremos una serie de programas básicos para conocer cómo manejar las funcionalidades básicas del análisis de imagen:

- Capturar imágenes de la cámara.
- Visualizarlas.
- Seleccionar la cámara a emplear.
- Cambiar la resolución de las imágenes que captura la cámara.
- Guardar resultados en archivos.
- Leer y procesar imágenes de archivos.

1.4.1. Primer programa para procesamiento de imágenes usando OpenCV

En este apartado vamos a describir cómo realizar una aplicación mínima que utilice OpenCV. Su única función será capturar imágenes y mostrarlas.

Este programa podría realizarse fácilmente usando librerías estándar de Android. No obstante, veremos cómo hacerlo usando OpenCV. La razón es que, de este modo, más adelante seremos capaces de analizar las imágenes usando la librería.

Ejercicio: Creación de un proyecto base con OpenCV (Base)

1. Crea un nuevo proyecto con los siguientes datos:

```
Application Name: Proyecto Base
Package Name: org.example.proyectobase
Minimum Required SDK: API 9: Android 2.3 (Gingerbread)
Create Activity: Empty Activity
```

Nota: Deja el resto de los parámetros con su valor por defecto.

2. En el menú, selecciona File/New/ImportModule y elige la carpeta <opencv>/sdk/java. Con esto, la librería OpenCV estará disponible para la aplicación.
3. Pulsando con el botón derecho del ratón sobre App, selecciona Open Module Settings y, en la pestaña Dependencies, haz clic en + para añadir un Module Dependency. Selecciona la librería OpenCV. Debería aparecer como se muestra en la figura:

4. Edita el fichero gradle de la aplicación para que aparezca así:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "com.example.aalbiol.curso5bis"
        minSdkVersion 9
```

```
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
}
```

5. Edita el fichero Gradle de la librería para que aparezca así:

```
apply plugin: 'com.android.library'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        minSdkVersion 9
        targetSdkVersion 23
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.txt'
        }
    }
```

6. La aplicación que vamos a desarrollar va a utilizar la cámara para obtener las imágenes a procesar. Abre el fichero AndroidManifest.java y añade dentro de la etiqueta `<manifest>`:

```
    <uses-permission android:name="android.permission.CAMERA"/>
```

   De esta forma la aplicación tendrá permiso para usar la cámara.

7. A partir de la versión 6 de Android ciertos permisos van a considerarse como peligrosos. Dentro de los permisos peligrosos se incluye el uso de la cámara. Cuando una aplicación realiza una acción asociada a un permiso peligroso, la aplicación ha de solicitar al usuario una validación para hacer la acción. Si no realizas esta solicitud se producirá un error cuando la aplicación se ejecute en

una versión 6 o superior. Existen dos posibilidades para permitir el uso de la cámara:

- Realizar la **activación del permiso de forma manual**. Para ello, tras intentar infructuosamente la ejecución del programa que requiere el permiso de la cámara, iremos al menú ajustes de nuestro dispositivo y seleccionaremos Ajustes/Aplicaciones. Allí seleccionaremos la aplicación en cuestión, y en el apartado permisos activaremos el permiso de la cámara manualmente.
- Realizar la **activación del permiso mediante código** en nuestro programa. De ese modo, la primera vez que se intente ejecutar el programa, se solicitará permiso. Para ello deben seguirse los pasos que se indican a continuación.

8. Para realizar esta solicitud añade el siguiente código en el método `onResume()` de la actividad:

```
if (ContextCompat.checkSelfPermission(this,
        Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED)
{
    if (ActivityCompat.shouldShowRequestPermissionRationale(this,
            Manifest.permission. CAMERA)) {
        ActivityCompat.requestPermissions(MainActivity.this,
                new String[]{ Manifest.permission.CAMERA },
                SOLICITUD_PERMISO_CAMERA);
    }
}
```

El código introducido solo tiene efecto si se ejecuta en una versión 6 de Android o superior.

Nota: Para más información consultad en El Gran Libro de Android el apartado Permisos en Android 6 Marshmallow.

9. Declara la siguiente variable al principio de la clase:

```
private static final int SOLICITUD_PERMISO_CAMERA = 0;
```

10. Abre el fichero res/layout/activity main.xml y reemplaza su contenido por el siguiente:

```
<Framelayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:opencv="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <org.opencv.android.JavaCameraView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/vista_camara"
        opencv:show_fps="true"/>

</Framelayout>
```

Observa como el elemento raíz de este layout es un `Framelayout`. Este tipo de layout nos resultará útil si queremos añadir más información superpuesta a la captura de la cámara. En el ejemplo se visualiza solo una vista que, como se aprecia por su dominio, pertenece a la librería OpenCV. En el siguiente apartado estudiaremos con más detalle la clase `JavaCameraView`. Observa como además de los atributos estándar se ha añadido uno más perteneciente a la definición xmlns opencv.

`show_fps`: Cuando su valor es true muestra en pantalla el número de capturas de imágenes que se realizan por segundo. Es un parámetro muy importante, sobre todo si queremos un procesamiento en tiempo real. Va a depender de la resolución de la imagen capturada, del hardware utilizado y del proceso que realicemos sobre la imagen.

11. Abre el fichero MainActivity.java y reemplaza su contenido por el siguiente:

```
public class MainActivity extends Activity implements CvCameraViewlisten-
er2, loaderCallbacklnterface {
    private static final String TAG = "Ejemplo OCV (MainActivity)";

    private CameraBridgeViewBase cameraView;

    @Override
    public void onCreate(Bundle savedlnstanceState) {
        super.onCreate(savedlnstanceState);
        getWindow().addFlags(WindowManager.layoutParams.FLAG_KEEP_SCREEN_ON);
        setContentView(R.layout.activity_main);
        cameraView = (CameraBridgeViewBase) findViewByld(R.id.vista_camara);
        cameraView.setCvCameraViewlistener(this);
    }
    @Override
    public void onPause() {
        super.onPause();
        if (cameraView != null)
          cameraView.disableView();
    }

    @Override
    public void onResume() {
        super.onResume();
        OpenCVloader.initAsync(OpenCVloader.OPENCV_VERSION_3_2_6, this,
            this);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (cameraView != null)
          cameraView.disableView();
    }

//lnterface CvCameraViewlistener2

    public void onCameraViewStarted(int width, int height) {
```

```
    }

    public void onCameraViewStopped() {
    }

    public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
        return inputFrame.rgba();
    }

//lnterface loaderCallbacklnterface

    public void onManagerConnected(int status) {
       switch (status) {
       case loaderCallbacklnterface.SUCCESS:
           log.i(TAG, "OpenCV se cargo correctamente");
           cameraView.enableView();
           break;
       default:
           log.e(TAG, "OpenCV no se cargo");
           Toast.makeText(MainActivity.this, "OpenCV no se cargo",
                  Toast.LENGTH_LONG).show();
           finish();
           break;
    }
}

    public void onPackagelnstall(int operation,
           lnstallCallbacklnterface callback) {
    }

}
```

Nota: Pulsa **Alt-Intro** en **Android Studio** para que se añadan automáticamente los paquetes que faltan en la sección `import`.

OpenCV proporciona una clase abstracta llamada `CameraBridgeViewBase` que representa un flujo de vídeo de una cámara. Esta clase extiende la clase de Android `SurfaceView`, por lo que las instancias de `CameraBridgeViewBase` pueden ser parte de la jerarquía de vistas. Además, una instancia de `CameraBridgeViewBase` puede mandar eventos a cualquier escuchador que implemente una de las dos interfaces `CvCameraViewlistener` o `CvCameraViewlistener2`. A menudo, el escuchador será una actividad, como es el caso del ejemplo propuesto.

Tanto `CvCameraViewlistener` como `CvCameraViewlistener2` proporcionan un mecanismo de callback para manejar el arranque y la parada de un flujo de vídeo, así como manejar la captura de cada fotograma. Cada vez que hay un nuevo fotograma disponible se ejecuta el método `onCameraFrame()`. Las diferencias entre ambos son las siguientes:

- `CvCameraViewlistener` siempre recibe una imagen RGBA[1] en color como un objeto de la clase Mat de OpenCV.
- `CvCameraViewlistener2` recibe un objeto de la clase `CvCameraViewFrame` de OpenCV. Dicha clase permite almacenar una imagen en formato nativo de Android. A partir de este objeto es posible obtener una imagen en color o en gris de OpenCV mediante los métodos:

  `rgba()`

  `gray()`

Como `CameraBridgeViewBase` (clase de OpenCV) es una clase abstracta, es necesaria una implementación de la misma. OpenCV proporciona una implementación:

`JavaCameraView`

Recordemos que para que el dispositivo pueda ejecutar aplicaciones que usen OpenCV es necesario tener instalada en el mismo la aplicación OpenCV Manager. Para soportar la interacción entre dicha aplicación y las aplicaciones cliente (nuestros programas) OpenCV proporciona una clase abstracta llamada BaseLoaderCallback. Esta clase declara un método de callback que se ejecuta cuando OpenCV comprueba que la librería está disponible. Típicamente, este callback es el lugar adecuado para inicializar otros objetos de OpenCV tales como CameraBridgeViewBase.

12. Abre el fichero AndroidManifest.java y añade en la etiqueta `<activity>` los dos atributos subrayados:

```
<activity android:name="MainActivity"
          android:label="@string/app_name"
          android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
          android:screenOrientation="landscape"
          android:configChanges="keyboardHidden/orientation">
```

El primer atributo aplica un tema a la actividad donde no se muestra la barra de título y además se ocupa toda la pantalla. El segundo fuerza la orientación de la pantalla a horizontal. Para comprender el tercero, recuerda que cuando cambia la configuración (idioma, orientación de la pantalla ...), una actividad es destruida y vuelta a ser creada cargando los recursos alternativos adecuados. En este tercer atributo estamos indicando que, aunque se saque/oculte el teclado o se cambie la orientación de la pantalla, no queremos que se reinicie la actividad.

13. Ejecuta el proyecto en un dispositivo con cámara y con la aplicación OpenCV Manager instalada. La aplicación ha de mostrar en pantalla la señal de vídeo captada por la cámara. Aparentemente no es un gran reto. No obstante, a diferencia del programa Cámara que suelen traer los dispositivos, tenemos un punto en el código donde podemos acceder a los píxeles para analizarlos.

---

[1] RGBA es un formato de imagen con cuatro valores por píxel, tres para RGB y un cuarto denominado Alfa que se suele usar para la transparencia.

1.4.2. Configurar la cámara y su resolución

Los dispositivos móviles actuales suelen disponer de varias cámaras que podremos usar indistintamente en nuestras aplicaciones. Además, disponen de la posibilidad de ajustar la resolución de captura o ajustar automáticamente el foco. A lo largo de este apartado veremos el código necesario para ajustar estos parámetros en nuestra aplicación OpenCV sobre Android. Este código lo incorporaremos al proyecto base para que así esté disponible en todas las aplicaciones que hagamos a partir de este.

Ejercicio: Selección de la cámara y resolución (Base2)

1. Abre el proyecto base.
2. Pulsa sobre la carpeta res y selecciona File > New > Android Resource File. En Resource Type, selecciona Menu y en File, Menu.
3. Abre el fichero anterior y reemplaza su contenido por el siguiente:

```
<?xml version="1.6" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/cambiarCamara"
          android:icon="@android:drawable/ic_menu_camera"
          android:title="Cambiar cómara"/>
    <item android:id="@+id/cambiarResolucion"
          android:icon="@android:drawable/ic_menu_view"
          android:title="Cambiar resolucion">
        <menu>
            <item android:id="@+id/resolucion_866x666"
                  android:title="866x666"/>
            <item android:id="@+id/resolucion_646x486"
                  android:title="646x486"/>
            <item android:id="@+id/resolucion_326x246"
                  android:title="326x246"/>
        </menu>
    </item>
</menu>
```

4. En algunos dispositivos, y dependiendo de la versión de Android, es posible que no se tenga una tecla para mostrar el menú. Añade el siguiente código en la actividad principal para que el menú se muestre al tocar en cualquier punto de la pantalla.

```
public boolean onTouchEvent(MotionEvent event) {
    openOptionsMenu();
    return true;
}
```

5. Añade el siguiente import en la actividad del proyecto:

```
import android.hardware.Camera;
```

6. Añade las siguientes declaraciones en la actividad del proyecto:

```
private int indiceCamara; // 0-> camara trasera; 1-> camara frontal
private int cam_anchura =320;// resolucion deseada de la imagen
private int cam_altura=240;
private static final String STATE_CAMERA_lNDEX = "cameralndex";
```

7. Añade el siguiente código:

```
public void onSavelnstanceState(Bundle savedlnstanceState) {
    // Save the current camera index.
    savedlnstanceState.putlnt(STATE_CAMERA_INDEX, indiceCamara);
    super.onSavelnstanceState(savedlnstanceState);
}
```

8. Añade las siguientes líneas en el método onCreate():

```
if (savedlnstanceState != null) {
    indiceCamara = savedlnstanceState.getlnt(STATE_CAMERA_lNDEX, 0);
} else {
    indiceCamara = CameraBridgeViewBase.CAMERA_ID_BACK;
}
cameraView.setCameralndex(indiceCamara);
```

Estas líneas son necesarias porque para cambiar de cámara es necesario recrear la actividad principal

9. Añade también el siguiente de código:

```
@Override public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    Menulnflater inflater = getMenulnflater();
    inflater.inflate(R.menu.menu, menu);
    return true;
}

@Override public boolean onOptionsltemSelected(Menultem item) {
    switch (item.getltemld()) {
    case R.id.cambiarCamara:
        if (indiceCamara == CameraBridgeViewBase.CAMERA_ID_BACK) {
            indiceCamara = CameraBridgeViewBase.CAMERA_ID_FRONT;
        } else indiceCamara = CameraBridgeViewBase.CAMERA_ID_BACK;
        recreate();
        break;
    case R.id.resolucion_800x600:
        cam_anchura = 800;
        cam_altura = 600;
        reiniciarResolucion();
        break;
    case R.id.resolucion_640x480:
        cam_anchura = 640;
        cam_altura = 480;
        reiniciarResolucion();
        break;
```

```
    case R.id.resolucion_320x240:
       cam_anchura = 320;
       cam_altura = 240;
       reiniciarResolucion();
       break;
    }
    String msg= "W="+lnteger.toString(cam_anchura)+ " H= " +
                  lnteger.toString(cam_altura) + " Cam= " +
                  lnteger.toBinaryString(indiceCamara);
    Toast.makeText(MainActivity.this, msg ,
                  Toast.lENGTH_lONG).show();
    return true;
}

public void reiniciarResolucion() {
    cameraView.disableView();
    cameraView.setMaxFrameSize(cam_anchura, cam_altura);
    cameraView.enableView();
}
```

La idea del código anterior es permitir seleccionar una opción de resolución y si queremos la cámara delantera o trasera. Como la opción de cambio de cámara requiere recrear la actividad, es necesario almacenar el número de cámara en un bundle.

10. No todas las resoluciones son posibles. La resolución que solicitamos es solo eso, una solicitud. Luego Android elige de entre las disponibles la que es más similar. Para poder acceder a los valores reales, modifica la función onCameraViewStarted().

```
public void onCameraViewStarted(int width, int height) {
    cam_altura = height; //Estas son las que se usan de verdad
    cam_anchura = width;
}
```

11. Modifica el método onManagerConnected para que fije la resolución una vez se haya cargado la librería OpenCV. Si no se indica la resolución, por defecto se considera la resolución máxima.

```
public void onManagerConnected(int status) {
    switch (status) {
    case loaderCallbacklnterface.SUCCESS:
       log.i(TAG, "OpenCV se cargo correctamente");
       cameraView.setMaxFrameSize(cam_anchura, cam_altura);
       cameraView.enableView();
       break;
    default:
       log.e(TAG, "OpenCV no se cargo");
           Toast.makeText(MainActivity.this, "OpenCV no se cargo",
           Toast.lENGTH_lONG).show();
           finish();
       break;
    }
}
```

12. Ejecuta el proyecto. Pulsa en la tecla de menú y prueba a cambiar la resolución y la cámara de la captura.

1.4.3. Seleccionar la entrada desde fichero

El reconocimiento de imágenes es un proceso extremadamente delicado. Tendrás que escoger los algoritmos adecuados y ajustar decenas de parámetros. Pequeños cambios en la captura o en las condiciones de iluminación pueden hacer que el programa deje de funcionar. Por esta razón, la fase de pruebas es especialmente larga y compleja. Para ayudarnos en esta fase podría ser interesante disponer de una entrada a nuestra aplicación que fuera siempre la misma. De esta forma vas a poder ejecutar el programa repetidas veces usando la misma entrada hasta que el resultado sea satisfactorio.

Con este propósito vamos a proporcionar una colección de ficheros con fotografías de señales de limitación de velocidad y diferentes condiciones de adquisición, pero el programa ha de estar preparado para tomar como entrada estos ficheros. En el siguiente ejercicio se explica cómo realizarlo.

Ejercicio: Seleccionar la entrada a reconocer desde un fichero de imagen (Base3)

1. Abre el proyecto base.
2. Crea la carpeta res/raw y copia en ella los ficheros que podrás descargarte del siguiente link: www.androidcurso.com/. Busca la sección referida al libro y dentro, ficheros OpenCV.
3. En el fichero res/menu.xml añade el siguiente ítem de menú dentro de la etiqueta <menu>.</menu>:

```
<item android:id="@+id/cambiar_entrada"
      android:icon="@android:drawable/ic_menu_gallery"
      android:title="Cambiar entrada">
    <menu>
        <item android:id="@+id/entrada_camara"
              android:title="camara"/>
        <item android:id="@+id/entrada_ficherol"
              android:title="ficherol"/>
        <item android:id="@+id/entrada_fichero2"
              android:title="fichero2"/>
    </menu>
</item>
```

4. Añade las siguientes declaraciones en la actividad del proyecto:

```
private int tipoEntrada = 0; // 0 -> cámara 1 -> fichero1 2 -> fichero2
Mat imagenRecurso_;
boolean recargarRecurso = false;
```

5. Añade también el siguiente código dentro del `switch` del método `onOptionsltemSelected()`:

```
case R.id.entrada_camara:
    tipoEntrada = 0;
    break;
case R.id.entrada_fichero1:
    tipoEntrada = 1;
    recargarRecurso = true;
    break;
case R.id.entrada_fichero2:
    tipoEntrada = 2;
    recargarRecurso = true;
    break;
```

6. Reemplaza el código del método `onCameraFrame()` por el siguiente:

```
public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
    Mat entrada;
    if (tipoEntrada == 0) {
        entrada = inputFrame.rgba();
    } else {
        if(recargarRecurso == true) {
        imagenRecurso_ = new Mat();
//Poner aqui el nombre de los archivos copiados
        int RECURSOS_FlCHEROS[] = {0, R.raw.fichero1, R.raw.fichero2};
            Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
                                RECURSOS_FlCHEROS[tipoEntrada]);
//Convierte el recurso a una Mat de OpenCV
           Utils.bitmapToMat(bitmap, imagenRecurso_);
           lmgproc.resize(imagenRecurso_, imagenRecurso_,
                                    new Size(cam_anchura, cam_altura));
            recargarRecurso = false;
        }
    entrada = imagenRecurso_;
    }
    Mat salida = entrada.clone();
    return salida;
}
```

Como las imágenes que tenemos en los recursos pueden tener un tamaño que no coincida con el de captura de la cámara. En ese caso, deberemos re-escalarlas para que tengan el mismo tamaño que las capturadas por la cámara. La instrucción de OpenCV `resize()` realiza esa función.

En este ejemplo se han introducido dos funciones de OpenCV, una para copiar imágenes y otra para redimensionarlas. Un poco más adelante se explicará con más detalle. De momento es suficiente saber que una Mat es el tipo de datos donde se almacena una imagen en OpenCV, y que en este ejemplo lo que se consigue es simplemente que la salida sea una copia de la imagen.

7. Ejecuta la aplicación y verifica que al seleccionar fichero1 o fichero2 se muestra la imagen correspondiente en la pantalla.

1.4.4. Guardar el resultado

En este apartado veremos cómo podemos guardar el resultado de nuestro procesamiento de imágenes en un archivo con el fin de poder depurar algoritmos o generar informes. El objetivo de los cambios que realizaremos será que cuando se seleccione en el menú la opción correspondiente, se guarden dos imágenes, una que sea la original que capturó la cámara y otra el resultado de nuestro procesamiento.

Ejercicio: Guardar el resultado en fichero de imagen (Base4)

1. Abre el proyecto base o continúa con el anterior.
2. En el fichero res/menu.xml añade el siguiente ítem de menú dentro de la etiqueta <menu>.</menu>:

```
<item android:id="@+id/guardar_imagenes"
      android:icon="@android:drawable/ic_menu_gallery"
      android:title=" Guardar imagenes ">
</item>
```

3. Añade las siguientes declaraciones en la actividad del proyecto:

```
private boolean guardarSiguientelmagen = false;
```

4. Añade también el siguiente código dentro del `switch` del método `onOptionsltemSelected()`:

```
case R.id.guardar_imagenes:
    guardarSiguientelmagen = true;
    break;
```

5. Añade al método `onCameraFrame()`, a continuación de la orden en que se calcula `salida`, lo siguiente:

```
public Mat onCameraFrame(CvCameraViewFrame inputFrame) {

    Mat salida = entrada.clone();
    if (guardarSiguientelmagen) {//Para foto salida debe ser rgba
        takePhoto(entrada, salida);
        guardarSiguientelmagen = false;
    }
    if(tipoEntrada > 0) {
//Es necesario que el tamaño de la salida coincida con el real de captura
        lmgproc.resize(salida, salida, new Size(cam_anchura , cam_altura));
    }
    return salida;
}
```

6. Añade el siguiente método:

```
private void takePhoto(final Mat input, final Mat output) {
    // Determina la ruta para crear los archivos
```

```
    final long currentTimeMillis = System.currentTimeMillis();
    final String appName =  getString(R.string.app_name);
    final String galleryPath = Environment
           .getExternalStoragePublicDirectory(
                  Environment.DIRECTORY_PICTURES).toString();
    final String albumPath = galleryPath + "/" + appName;
    final String photoPathIn = albumPath + "/In_" + currentTimeMillis + ".png";
    final String photoPathOut = albumPath + "/Out_" + currentTimeMillis
           + ".png";
    // Asegurarse que el directorio existe
    File album = new File(albumPath);
    if (!album.isDirectory() && !album.mkdirs()) {
       log.e(TAG, "Error al crear el directorio " + albumPath);
       return;
    }
    // Intenta crear los archivos
    Mat mBgr = new Mat();
    if (output.channels() == 1)
       Imgproc.cvtColor(output, mBgr, Imgproc.COLOR_GRAY2BGR, 3);
    else
       Imgproc.cvtColor(output, mBgr, Imgproc.COLOR_RGBA2BGR, 3);
    if (!Imgcodecs.imwrite(photoPathOut, mBgr)) {
       log.e(TAG, "Fallo al guardar " + photoPathOut);
    }
    if (input.channels() == 1)
       Imgproc.cvtColor(input, mBgr, Imgproc.COLOR_GRAY2BGR, 3);
    else
       Imgproc.cvtColor(input, mBgr, Imgproc.COLOR_RGBA2BGR, 3);
    if (!Imgcodecs.imwrite(photoPathIn, mBgr))
       log.e(TAG, "Fallo al guardar " + photoPathIn);
    mBgr.release();
    return;
}
```

7. Abre el fichero AndroidManifest.java y añade dentro de la etiqueta `<manifest>`:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
```

   De esta forma la aplicación tendrá permiso para guardar las imágenes en la memoria externa.

8. Ejecuta la aplicación y prueba a guardar una imagen.

9. Abre un explorador de archivos y busca la carpeta /Pictures y dentro el nombre de la aplicación. Encontrarás un fichero llamado In xxx.png y otro llamado Out xxx.png.

1.4.5. La clase Mat de OpenCV

La clase Mat es la forma que tiene OpenCV de representar una imagen. Un objeto de tipo Mat consta de dos partes:

- Cabecera: son un conjunto de metadatos que permiten manejar las imágenes. Entre la información que contiene la cabecera está:

  - Anchura y altura en píxeles de la imagen.
  - Información sobre el tipo de dato de los píxeles: uchar, int, float …
  - Información sobre el número de canales de la imagen (uno si es monocroma o tres si es en color).
  - Puntero a los píxeles: dirección de memoria donde están almacenados los píxeles.
- Píxeles: zona de memoria donde físicamente residen los píxeles.

Para crear un objeto de tipo Mat en Java debemos escribir:

```
Mat img1 = new Mat(); //Imagen vacía
Mat img2 = new Mat( 200, 100, CvType.CV_64FC1); //200 filas x 100 columnas
de tipo float de 64 bits.
```

Un aspecto que al principio causa confusión es cuando escribimos algo como:

```
Mat img1 = new Mat( 200, 100, CvType.CV_8UC3); //200x100 tipo unsigned int
de 8 bpp y 3 canales
Mat img2 = img1;
```

El operador "=" podría hacernos pensar que lo que estamos realizando es una copia de la imagen 1. En realidad, esta instrucción crea una nueva referencia al mismo objeto de tipo Mat que `img1`. Es decir, si modificáramos posteriormente los píxeles de `img1`, también estaríamos modificando los píxeles de `img2` y viceversa.

Si realmente lo que se quiere es hacer una copia de los píxeles en otra zona de memoria con el fin de preservar la imagen original, si realizamos cambios en la imagen, deberíamos escribir:

```
Mat img3 = img1.clone(); //Creamos una copia con nuevos píxeles
```

Un aspecto interesante de OpenCV es que permite crear una nueva cabecera de imagen que simule que tenemos una imagen que es un trozo de la original sin copiar los píxeles. El hecho de ahorrarnos la copia de los píxeles hace que estos métodos sean muy eficaces.

```
Mat m1 = new Mat( 200, 100, CvType.CV_8UC3);
Mat m2 = m1.row(5);   // Nueva cabecera mismos pixels
Mat m3 = m1.colRange(27,33); // Nueva cabecera mismos pixels
Mat m4 = m1.submat (int rowStart, int rowEnd, int colStart, int colEnd);
// Nueva cabecera mismos pixels
```

De ese modo, si cambiamos los píxeles de `m2`, también estaremos cambiando los píxeles de la fila 5 de `m1`.

Una posible aplicación de `submat()` es procesar un trozo de una imagen grande. Supongamos que, de alguna manera, hemos localizado un círculo rojo en la imagen, y queremos analizar su interior para ver si es una señal de limitación de velocidad. Si en vez de procesar la imagen completa seguimos nuestro análisis tomando un rectángulo de tamaño adecuado, dado que la cantidad de píxeles del trozo es mucho menor, el procesamiento será mucho más rápido:

Algunos aspectos básicos adicionales que conviene conocer son:

Cómo averiguar el número de canales de una imagen:

```
int nchannels = m1.channels();//Consultar num canales 1:monocromo, 3:color
```

Saber extraer un canal (componente de color) de una imagen:

```
Mat m2 = new Mat();
Core.extractChannel(m1,1,m2); //m2: imagen monocroma, canal verde de m1
```

Saber descomponer una imagen de color en sus componentes:

```
java.util.list<Mat> mv; //lista de Mats
Core.split(m1,mv); //Divide imagen en color en tantos Mats como canales
```

Cambiar de formato una imagen, por ejemplo, pasarla de color a gris:

```
lmgproc.cvtColor(m1,m2,lmgproc.COlOR_BGR2GRAY);
lmgproc.cvtColor(m1,m2,lmgproc.COlOR_RGBA2GRAY);
```

Cambiar tipo de dato de una imagen, por ejemplo, de `unsigned char` a `double`:

```
Mat m1 = new Mat( 200, 100, CvType.CV_8UC1);
Mat m2 = new Mat();
m1.convertTo(m2, CvType.CV_64FC1);
```

Preguntas de repaso: Introducción a OpenCV

1.4.6. Determinar la orientación del dispositivo

Si queremos reconocer señales, es importante saber la orientación de la cámara respecto al plano del suelo. Independientemente de cómo orientemos el teléfono,

la imagen en la pantalla la veremos correctamente. Es necesario un procedimiento para determinar si la imagen está invertida o no, ya que si la señal está boca abajo, no podremos leerla.

Hay dos posibilidades:

- Usar la información del acelerómetro para, por software, rotar adecuadamente la imagen.
- Colocar el teléfono siempre en la misma orientación. Para ello es importante saber qué esquina considerará nuestro programa que es la superior izquierda.

El ejercicio que se propone consiste en dibujar un pequeño cuadrado blanco en la esquina superior izquierda de la imagen, para de ese modo situar el teléfono en la posición correcta.

Figura 5. Resultado a obtener con el programa para determinar la orientación del teléfono.

Ejercicio: Identificar la orientación del teléfono.

1. Haz una copia del proyecto Base4 (que incluye poder seleccionar cámara, resolución, procesar archivos y guardar resultados). Llamemos al nuevo proyecto "Orientacion".
2. Sobre dicha copia modifica el método `onCameraFrame()` para que, en vez de copiar simplemente la entrada hacia la salida, se rellene un pequeño cuadrado blanco en la esquina superior izquierda. Elimina la línea que contenía `entrada.clone()` y reemplázala por:

```
Mat esquina = entrada.submat(0,10,0,10); //Arriba-izquierda
esquina.setTo(new Scalar(255,255,255));
Mat salida = entrada;
```

Nota: Pulsa **Alt-Intro** en **Android Studio** para que se añadan automáticamente los paquetes que faltan en la sección `import`.

La imagen `esquina` es una referencia a los píxeles de la esquina superior-izquierda de la imagen original. Aparentemente es una imagen distinta, pero físicamente los píxeles son los mismos que los de `entrada`.

Con el método `esquina.setTo()` lo que hacemos es poner todos los píxeles de `esquina` en un valor constante, en este caso blanco. Dado que los píxeles de esquina son físicamente los mismos que los de `entrada`, resulta que es como si hubiéramos dibujado un cuadrado blanco sobre dicha imagen.

Finalmente, la imagen de salida es una referencia a la imagen de entrada (sobre la que se ha dibujado el cuadrado blanco). No se están copiando los píxeles físicamente, solo estamos creando una referencia de imagen.

3. Ejecuta el programa en el dispositivo e intenta recordar la orientación que hace que el cuadrado blanco aparezca arriba a la izquierda. Te será de utilidad más adelante.

1.4.7. Separar el procesamiento de las imágenes

El programa básico desarrollado hasta el momento consta de una única clase. En ella se mezclan los aspectos de pura programación Android con los de análisis de imágenes.

Resulta más claro separar ambos aspectos de la aplicación. En este apartado veremos cómo agrupar todo el código de análisis de imagen en una clase diferenciada de la clase de la actividad.

Adicionalmente el hecho de separar el análisis de imagen tendrá la ventaja de que todas las imágenes temporales de trabajo que vayamos generando pueden ser miembros de la clase, y no precisarán que se reserven y liberen cada nuevo fotograma.

Ejercicio: Separación análisis de imagen (Base 5).

1. Haz una copia del proyecto Base4 (que incluye poder seleccionar cámara, resolución, procesar archivos y guardar resultados) y renómbralo como Base5.
2. Genera una nueva clase haciendo clic con el botón derecho en el explorador de proyectos y seleccionando Nueva > Clase. Llámala, por ejemplo, Procesador.
3. Añade en dicha clase un método llamado `procesa()` que simplemente copie la salida a la entrada:

```
public class Procesador {
    public Procesador() {} //Constructor
    public Mat procesa(Mat entrada) {
        Mat salida = entrada.clone();
        return salida;
    }
}
```

4. Añade en la actividad principal una variable de la clase Procesador:

```
Procesador procesador;
```

5. Inicializa dicha variable en el método `onCameraViewStarted()`. Esto debe realizarse en esta función porque es en esa función cuando tenemos cargada la librería OpenCV. **Realizar la inicialización en el constructor de la actividad daría un error** si en dicho constructor se intenta hacer cualquier proceso de OpenCV.

```
procesador = new Procesador();
```

6. En la función `onCameraFrame()` reemplaza la llamada a clone() por:

```
Mat salida = procesador.procesa(entrada);
```

7. Ejecuta el programa y comprueba que funciona como Base4.

1.4.8. Procesar imágenes monocromas

A veces interesa procesar imágenes en blanco y negro. La ventaja de hacerlo es que tienen una tercera parte de los píxeles y, por tanto, son más rápidas de analizar.

Ejercicio: Procesar imágenes monocromas (Base5mono).

1. Haz una copia del proyecto Base5 (que incluye poder seleccionar cámara, resolución, procesar archivos y guardar resultados) y renómbralo como Base5mono.
2. Sobre dicha copia modifica el método onCameraFrame() para que la imagen de entrada sea monocroma:

```
Mat entrada = inputFrame.gray();
```

3. Prueba el programa y comprueba que se ven imágenes grises.

1.4.9. Procesamiento de las imágenes en fases

En los siguientes apartados se describen diversas técnicas para el procesamiento de imágenes. En un problema real será necesario combinar varias de estas técnicas para obtener el objetivo final. Por ejemplo, para abordar la detección de señales de tráfico se van a realizar cinco fases que corresponden a:

Orden	Fase	Objetivo
1	Transformación de intensidad	Obtener una imagen monocroma donde las intensidades de los píxeles se distribuyan de forma conveniente a nuestro objetivo
2	Operador local	Obtener una imagen donde el valor de cada píxel dependa de los píxeles cercanos. Por ejemplo, para resaltar los contornos de los objetos.
3	Binarización	Obtener una imagen donde los píxeles solo tengan valores 1 o 0.

4	Segmentación	Localizar los diferentes objetos que aparecen en la imagen. Por ejemplo, localizar círculos rojos.
5	Reconocimiento	Identificar los tipos de objetos encontrados. Por ejemplo, se trata de una señal de 60 o de 80.

Figura 6. Secuencias de fases que utilizaremos para la detección de señales de tráfico.

Los siguientes apartados coinciden con cada una de estas fases y en ellos se describen diferentes alternativas de procesado que podemos aplicar.

Cada una de estas técnicas puedes probarlas en un proyecto diferente, tal y como se propone en los ejercicios paso a paso. En el siguiente ejercicio se sugiere otra alternativa de trabajo. Consiste en recopilar estas técnicas en un único proyecto, de forma que desde las preferencias del mismo se pueda combinar qué fases quieres realizar y qué técnica concreta se aplica en cada una de las fases. Cada alumno es libre de organizar su trabajo de la forma que considere más adecuada.

El siguiente diagrama muestra la organización del programa BaseFases:

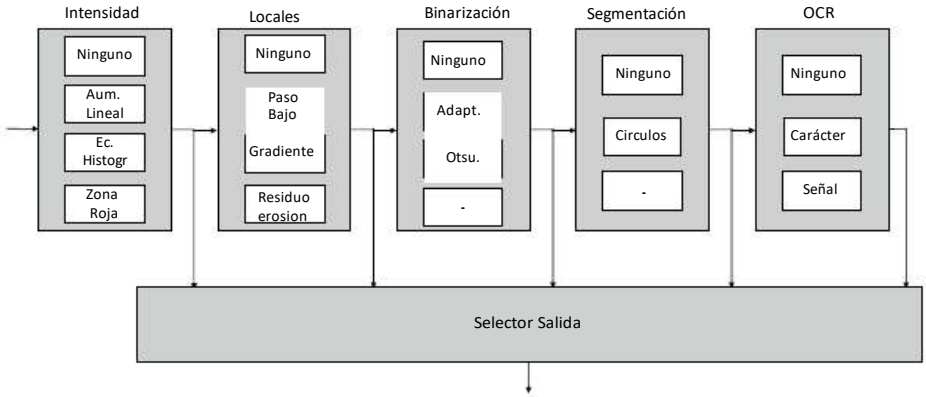


Figura 7. Diagrama de bloques del programa BaseFases. Se puede elegir una opción por bloque y seleccionar qué se quiere ver como salida.

Ejercicio: Análisis de imagen en fases (BaseFases).

1. Haz una copia del proyecto Base5 y renómbralo como BaseFases.
2. Reemplaza el código de la clase `Procesador` por:

```
public class Procesador {
    private Mat gris;
    private Mat salidaintensidad;
    private Mat salidatrlocal;
    private Mat salidabinarizacion;
    private Mat salidasegmentacion;
    private Mat salidaocr;

    public enum Salida {ENTRADA, INTENSIDAD, OPERADOR_LOCAL, BINARIZACION,
```

```
        SEGMENTACION, RECONOCIMIENTO
    }
    public enum TipoIntensidad {SIN_PROCESO, LUMINANCIA,
        AUMENTO_LINEAL_CONSTRASTE, EQUALIZ_HISTOGRAMA, ZONAS_ROJAS}
    public enum TipoOperadorLocal {SIN_PROCESO, PASO_BAJO, PASO_ALTO,
        GRADIENTES}
    public enum TipoBinarizacion {SIN_PROCESO, ADAPTATIVA, MAXIMO}
    public enum TipoSegmentacion {SIN_PROCESO}
    public enum TipoReconocimiento {SIN_PROCESO}

    private Salida mostrarSalida;
    private TipoIntensidad tipoIntensidad;
    private TipoOperadorLocal tipoOperadorLocal;
    private TipoBinarizacion tipoBinarizacion;
    private TipoSegmentacion tipoSegmentacion;
    private TipoReconocimiento tipoReconocimiento;

    public Procesador() {
        mostrarSalida = Salida. INTENSIDAD;
        tipoIntensidad = TipoIntensidad. LUMINANCIA;
        tipoOperadorLocal = TipoOperadorLocal.SIN_PROCESO;
        tipoBinarizacion = TipoBinarizacion.SIN_PROCESO;
        tipoSegmentacion = TipoSegmentacion.SIN_PROCESO;
        tipoReconocimiento = TipoReconocimiento.SIN_PROCESO;
        salidaIntensidad = new Mat();
        salidaTrLocal = new Mat();
        salidaBinarizacion = new Mat();
        salidaSegmentacion = new Mat();
        salidaOCR = new Mat();
        gris = new Mat();
    }
}
```

El primer tipo enumerado se corresponde con las diferentes fases en las que se divide el proceso. Vamos a definir la variable `mostrarSalida` de este tipo, de manera que podremos configurar el alcance del proceso a realizar. Es decir, si tiene el valor `BINARIZACION` indicamos que solo queremos realizar el proceso hasta esta fase, mostrándose como salida el resultado tras la binarización. Los siguientes seis enumerados definen diferentes alternativas de procesado para cada una de las fases. El valor `SIN_PROCESO` indica que se ha de pasar a la siguiente fase sin realizar ninguna modificación en la imagen. Se han introducido solo algunos valores. Deberás añadir nuevos valores cada vez que se plantee un nuevo tipo de procesado. Para poder configurar el tipo de procesado a realizar en cada fase se han declarado las variables `tipoXXX`. En el constructor de la clase se inicializan las variables.

3. Añade el siguiente método a la clase `Procesador`:

```
public Mat procesa(Mat entrada) {
        if (mostrarSalida == Salida.ENTRADA) {
            return entrada;
        }
// Transformación intensidad
```

```
        switch (tipoIntensidad) {
            case SIN_PROCESO:
                salidaintensidad = entrada;
                break;
            case LUMINANCIA:
                lmgproc.cvtColor(entrada, salidaintensidad,
                                        lmgproc.COLOR_RGBA2GRAY);
                break;
            case AUMENTO_LINEAL_CONSTRASTE:
                lmgproc.cvtColor(entrada, gris, lmgproc.COLOR_RGBA2GRAY);
                aumentolinealConstraste (gris); //resultado en salidain-
                                                              tensidad
                break;
            case EQUALIZ_HISTOGRAMA:
                lmgproc.cvtColor(entrada, gris, lmgproc.COLOR_RGBA2GRAY);
//Eq. Hist necesita gris
                lmgproc.equalizeHist(gris, salidaintensidad);
                break;
            case ZONAS_ROJAS:
                zonaRoja (entrada); //resultado en salidaintensidad
                break;
            default:
                salidaintensidad = entrada;
        }
        if (mostrarSalida == Salida.INTENSIDAD) {
            return salidaintensidad;
        }
// Operador local
        switch (tipoOperadorlocal) {
            case SIN_PROCESO:
                salidatrlocal = salidaintensidad;
                break;
            case PASO_BAJO:
                pasoBajo(salidaintensidad); //resultado en salidatrlocal
                break;
        }
        if (mostrarSalida == Salida.OPERADOR_LOCAL) {
            return salidatrlocal;
        }
// Binarización
        switch (tipoBinarizacion) {
            case SIN_PROCESO:
                salidabinarizacion = salidatrlocal;
                break;
            default:
                salidabinarizacion = salidatrlocal;
                break;
        }
        if (mostrarSalida == Salida.BINARIZACION) {
            return salidabinarizacion;
        }
// Segmentación
        switch (tipoSegmentacion) {
```

```
            case SIN_PROCESO:
                salidasegmentacion = salidabinarizacion;
                break;
        }
        if (mostrarSalida == Salida.SEGMENTACION) {
            return salidasegmentacion;
        }

// Reconocimiento OCR
        switch (tipoReconocimiento) {
            case SIN_PROCESO:
                salidaocr = salidasegmentacion ;
                break;
        }
        return salidaocr;
    }
```

El método `procesa()` va a ir llamando a las diferentes fases del proceso según la configuración especificada. Observa cómo antes de cada fase se comprueba el valor de `mostrarSalida` para ver si tenemos que terminar el proceso en ese punto. Para cada una de las fases se ha añadido un **`switch`** donde se comprueba el tipo de proceso a realizar. Hay que hacer algún comentario sobre la primera fase, transformación de intensidad. Vamos a suponer que la entrada a esta fase es una imagen en color (RGB), pero la salida ha de ser siempre una imagen en tonos de grises. Por esta razón se ha añadido la cláusula **`default:`** donde la imagen `entrada`, en RGB, se convierte en `salida`, en tonos de grises. Observa los dos primeros **`case`** de este **`switch`**. Los dos comienzan convirtiendo la entrada en tonos de grises, dado que el tipo de proceso así lo requiere. Se diferencian en que el primero llama a un método creado en nuestro proyecto (lo implementamos en el siguiente punto), mientras que el segundo llama a un método de la librería OpenCV. El resto del código sigue una estructura similar. Tendrás que ir completándolo a medida que avances en el capítulo.

4. Sitúa el cursor al final de la clase y con el botón derecho escoge Source > Generate Getters and Setters... Selecciona todas las variables de la clase.

5. Para implementar los diferentes de tipo de procesamiento te proponemos ir creando diferentes funciones dentro de la clase en `Procesador`. De momento dichas funciones simplemente devolverán la imagen de entrada como salida:

```
void zonaRoja(Mat entrada){ //Ejemplo para ser rellenado en curso
    salidaintensidad = entrada;
}
void aumentolinealConstraste(Mat entrada) { //Ejemplo para ser rellenado
   salidaintensidad = entrada;
}
void pasoBajo(Mat entrada) { //Ejemplo para ser rellenado
    salidatrlocal entrada;
}
```

6. Ejecuta el proyecto. La salida aparecerá en tonos de grises.

Ejercicio: Selección de fases a realizar desde preferencias.

En el ejercicio anterior declaramos siete variables en la clase Procesador para configurar las diferentes fases del proceso. En este ejercicio vamos a añadir preferencias a la aplicación para que el usuario pueda cambiar estos valores y guardarlos de forma permanente.

1. Crea un nuevo recurso de preferencias res/xml/preferencias.xml con el siguiente código:

```
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android" >
  <PreferenceCategory android:title="Ajustes básicos">
    <CheckBoxPreference
     android:key="pantalla_partida"
     android:title="Pantalla partida"
     android:summary="Muestra a la izquierda la entrada y a la derecha la salida"
     android:defaultValue="false"/>
      <listPreference
     android:key="salida"
     android:title="Salida a Mostrar"
     android:summary="Muestra el proceso hasta la fase indicada"
     android:entries="@array/tipoSalida"
     android:entryValues="@array/valorSalida"
     android:defaultValue="RECONOClMlENTO"/>
 </PreferenceCategory>
 <PreferenceCategory android:title="Configurar Fases del proceso">
   <listPreference
     android:key="intensidad"
     android:title="Transformación lntensidad"
     android:summary="Obtiene imagen de grises mejorando distribución intensidades"
     android:entries="@array/tipolntensidad"
     android:entryValues="@array/valorlntensidad"
     android:defaultValue="SlN_PROCESO"/>
  <listPreference
     android:key="operador_local"
     android:title="Operador local"
     android:summary="Aplica un operador local"
     android:entries="@array/tipoOperadorlocal"
```

```
      android:entryValues="@array/valorOperadorlocal"
      android:defaultValue="SlN_PROCESO"/>
  <listPreference
      android:key="binarizacion"
      android:title="Binarización"
      android:summary="Obtiene una imagen binaria a partir de una de grises"
      android:entries="@array/tipoBinarizacion"
      android:entryValues="@array/valorBinarizacion"
      android:defaultValue="SlN_PROCESO"/>
  <listPreference
      android:key="segmentacion"
      android:title="Segmentación"
      android:summary="localiza objetos en la imagen"
      android:entries="@array/tipoSegmentacion"
      android:entryValues="@array/valorSegmentacion"
      android:defaultValue="SlN_PROCESO"/>
  <listPreference
      android:key="reconocimiento"
      android:title="Reconocimiento"
      android:summary="ldentifica los objetos encontrados"
      android:entries="@array/tipoReconocimiento"
      android:entryValues="@array/valorReconocimiento"
      android:defaultValue="SlN_PROCESO"/>
  </PreferenceCategory>
</PreferenceScreen>
```

Nota: La preferencia `pantalla_partida` se utilizará en uno de los ejercicios siguientes.

2. Crea un nuevo recurso de preferencias res/values/arrays.xml con el siguiente código:

```
<resources>
<string-array name="tipoSalida">
  <item>Entrada</item>
  <item>Transformación de lntensidad</item>
  <item>Operador local</item>
  <item>Binarización</item>
  <item>Segmentación</item>
  <item>Reconocimiento</item>
</string-array>
<string-array name="valorSalida">
  <item>ENTRADA</item>
  <item>lNTENSlDAD</item>
  <item>OPERADOR_lOCAl</item>
  <item>BlNARlZAClON</item>
  <item>SEGMENTAClON</item>
  <item>RECONOClMlENTO</item>
</string-array>
<string-array name="tipoIntensidad">
  <item>Sin proceso</item>
  <item>Aumento lineal contraste</item>
  <item>Ecualización de histograma</item>
  <item>luminancia</item>
  <item>Zonas Rojas</item>
```

```
</string-array>
<string-array name="valorIntensidad">
  <item>SlN_PROCESO</item>
  <item>AUMENTO_llNEAl_CONSTRASTE</item>
  <item>EQUAllZ_HlSTOGRAMA</item>
  <item>lUMlNANClA</item>
  <item>ZONAS_ROJAS</item>
</string-array>
<string-array name="tipoOperadorLocal">
        <item>Sin proceso</item>
        <item>Filtro Paso Bajo</item>
    </string-array>
    <string-array name="valorOperadorLocal">
        <item>SlN_PROCESO</item>
        <item>PASO_BAJO</item>
    </string-array>
    <string-array name="tipoBinarizacion">
        <item>Sin proceso</item>
    </string-array>
    <string-array name="valorBinarizacion">
        <item>SlN_PROCESO</item>
    </string-array>
    <string-array name="tipoSegmentacion">
        <item>Sin proceso</item>
    </string-array>
    <string-array name="valorSegmentacion">
        <item>SlN_PROCESO</item>
    </string-array>
    <string-array name="tipoReconocimiento">
        <item>Sin proceso</item>
    </string-array>
    <string-array name="valorReconocimiento">
        <item>SlN_PROCESO</item>
    </string-array>
</resources>
```

3. Crea la clase `Preferencias` con el siguiente código:

```
public class Preferencias extends PreferenceActivity {
    @SuppressWarnings("deprecation")
    @Override
    protected void onCreate(Bundle savedlnstanceState) {
        super.onCreate(savedlnstanceState);
        addPreferencesFromResource(R.xml.preferencias);
    }
}
```

Nota: Desde el nivel de API 11, el método `addPreferencesFromResource()` se ha marcado como obsoleto. En lugar de usar la clase `PreferenceActivity`, se recomienda utilizar `PreferenceFragments`. No obstante, se ha preferido utilizar la primera clase dado que requiere menos código.

4. Registra la nueva actividad en AndroidManifest.xml.

5. Abre el fichero res/menu/menu.xml y añade el siguiente ítem de menú:

```
<item    android:id="@+id/preferencias"
         android:icon="@android:drawable/ic_menu_preferences"
         android:title="Preferencias"/>
```

6. En `MainActivity`, dentro del método `onOptionsItemSelected()`, añade el siguiente código:

```
case R.id.preferencias:
    Intent i = new Intent(this, Preferencias.class);
    startActivity(i);
    break;
```

7. Añade al final del método `onCameraViewStarted()` el siguiente código:

```
PreferenceManager.setDefaultValues(this, R.xml.preferencias, false);
SharedPreferences preferencias = PreferenceManager
                          .getDefaultSharedPreferences(this);
pantallaPartida = (preferencias.getBoolean("pantalla_partida", true));
String valor = preferencias.getString("salida", "ENTRADA");
procesador.setMostrarSalida(Salida.valueOf(valor));
valor = preferencias.getString("intensidad", "SIN_PROCESO");
procesador.setTipoIntensidad(TipoIntensidad.valueOf(valor));
valor = preferencias.getString("operador_local", "SIN_PROCESO");
procesador.setTipoOperadorlocal(TipoOperadorlocal.valueOf(valor));
valor = preferencias.getString("binarizacion", "SIN_PROCESO");
procesador.setTipoBinarizacion(TipoBinarizacion.valueOf(valor));
valor = preferencias.getString("segmentacion", "SIN_PROCESO");
procesador.setTipoSegmentacion(TipoSegmentacion.valueOf(valor));
valor = preferencias.getString("reconocimiento", "SIN_PROCESO");
procesador.setTipoReconocimiento(TipoReconocimiento.valueOf(valor));
```

8. Declara la siguiente variable al principio de la clase:

```
private boolean pantallaPartida = false;
```

   Se utilizará en un próximo ejercicio.

9. Ejecuta el proyecto y comprueba que puedes acceder a las preferencias a través del menú. Si seleccionas en Salida a mostrar el valor Entrada, se mostrará la imagen en color. En cualquier otra fase se mostrará en grises.

1.4.1 O. Conclusión

Tras los pasos anteriores tendremos un programa básico que será capaz de:

1. Capturar imágenes de la cámara y mostrar el resultado.
2. Seleccionar cualquiera de las dos cámaras del dispositivo.
3. Procesar archivos prealmacenados como recursos para observar efectos sobre ciertas imágenes o para depuración.
4. Cambiar la resolución de captura de la cámara.
5. Guardar en imágenes la pareja: entrada de la cámara y salida.

6. Separar el análisis de imagen en una clase y todo lo relacionado con Android en otra.

Adicionalmente, hemos visto cómo procesar imágenes monocromas. Este programa básico será el punto de partida de todos los ejemplos siguientes.

Preguntas de repaso: Aplicación Básica OpenCV

CAPÍTULO 2.

Visión artificial: preproceso

Por ANTONIO ALBIOL

En este capítulo veremos las técnicas de preproceso de imagen. Estas pueden corregir problemas de iluminación y enfatizar ciertas zonas de las imágenes, como los contornos, las zonas oscuras, las zonas claras, etc.

Normalmente el preproceso es el primer paso en cualquier análisis automático de imágenes y también suele ser el que involucra mayor coste computacional. Por tanto, entender bien la función de los distintos operadores y las cuestiones principales relacionadas con la eficiencia computacional serán vitales para el desarrollo de aplicaciones de análisis de imagen en Android.

Objetivos:

- Aplicar las técnicas de aumento lineal constante y ecualización de histograma para corregir problemas de iluminación de forma automática.
- Conocer diferentes opciones para transformar una imagen en color en una monocroma.
- Cómo localizar zonas de color rojo o cualquier otro color en una imagen.
- Utilizar operadores locales para aplicar filtros paso-bajo y paso-alto.
- Detectar los contornos de una imagen utilizando gradientes.
- Aprender a utilizar los operadores morfológicos.

2.1. Transformaciones de Intensidad

Vamos a comenzar a ver las técnicas de procesamiento de imágenes propiamente dichas. En este primer punto veremos lo que se conocen como transformaciones de intensidad. Un ejemplo de transformación de intensidad se muestra en la siguiente figura:

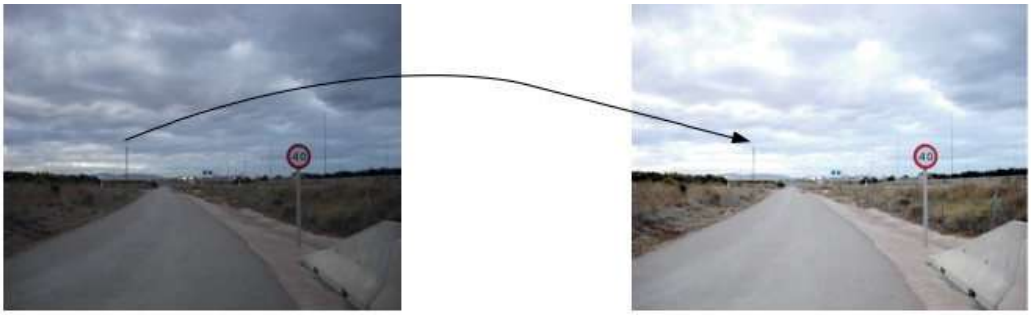


Figura 8. Ejemplo de transformación de intensidad.

Dada una imagen, el resultado de una transformación de intensidad es otra imagen, del mismo tamaño que la original, donde cada píxel de la imagen de salida depende únicamente del mismo píxel de la imagen de entrada. Como cada punto de la imagen de entrada se transforma en un punto de la imagen de salida, también se suele hablar de operadores puntuales.

Las transformaciones de intensidad vienen caracterizadas por una función que determina la intensidad del punto de salida en función de la intensidad de entrada. El aspecto de dicha función puede ser como el que se muestra en la curva:

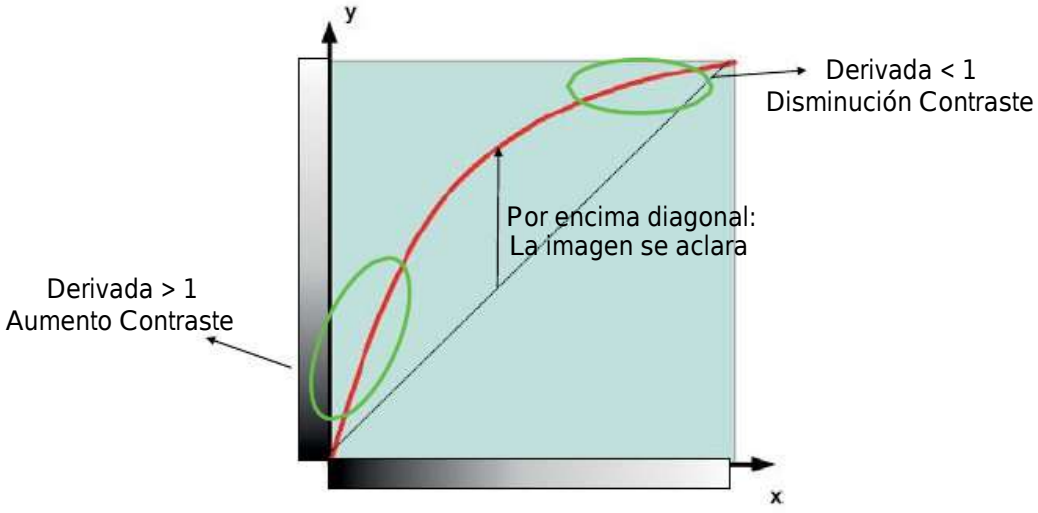


Figura 9. Curva entrada salida de una transformación de intensidad.

La forma de la curva determina totalmente el tipo de transformación de intensidad:

- En el rango de intensidades donde la curva está por encima de la identidad (diagonal), los píxeles de la imagen de salida serán más claros que en la entrada.

- En el rango de intensidades donde la pendiente de la curva sea mayor que uno, aumentará el contraste. El contraste de dos píxeles se define como su diferencia de intensidad. En otras palabras, si dos píxeles de la imagen de entrada tienen una diferencia de intensidad de 10 en la imagen de entrada, y la pendiente es mayor que uno, en la imagen de salida tendrán una diferencia de intensidad mayor de 10.
- Normalmente la curva será monótona creciente. Esta condición garantiza que si un píxel es más claro que otro en la imagen de entrada, también lo será en la imagen de salida.

La principal utilidad de las transformaciones de intensidad es corregir problemas de iluminación para:

- Poder visualizarla mejor.
- Facilitar su análisis automático.

La forma de la curva de entrada/salida se puede elegir de forma manual (como por ejemplo se hace en programas como Photoshop o Gimp). Sin embargo, para nuestra aplicación lo que resulta más interesante es la posibilidad de determinar dicha curva de entrada/salida de manera automática.

Para ello es necesario hacer uso de una herramienta llamada Histograma.

2.1.1. Histogramas

Un histograma, no es más que una función que nos muestra cuántos píxeles tienen cada nivel de intensidad. Así, por ejemplo, en imágenes normales con 256 niveles de intensidad, el histograma será un vector de 256 enteros, de tal manera que el elemento 19 del histograma nos indica cuántos píxeles tienen una intensidad igual a 19.

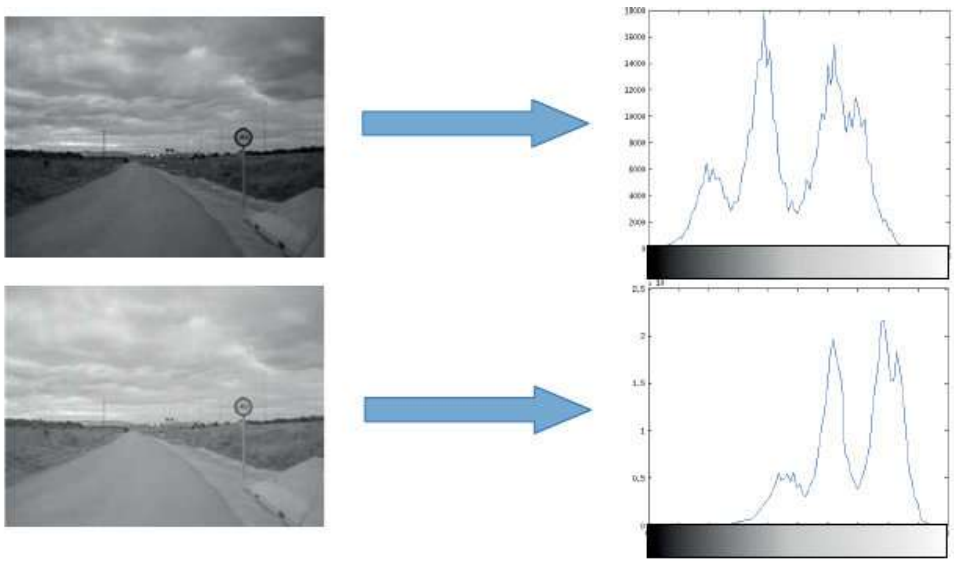


Figura 10. Imágenes y sus correspondientes histogramas a la derecha.

El histograma de una imagen puede ser de ayuda en la adquisición de la misma o para determinar si la imagen está bien hecha. Por ejemplo, en la imagen inferior de la Figura 10, el histograma nos indica que apenas hay píxeles con intensidades pequeñas. En los siguientes apartados veremos dos aplicaciones del uso de histogramas para determinar la curva de entrada/salida:

- Aumento lineal del contraste.
- Ecualización del histograma.

2.1.2. Aumento lineal del contraste

El objetivo del aumento lineal del contraste es lograr que el rango de intensidades de la imagen cubra todo el margen disponible (0-255).

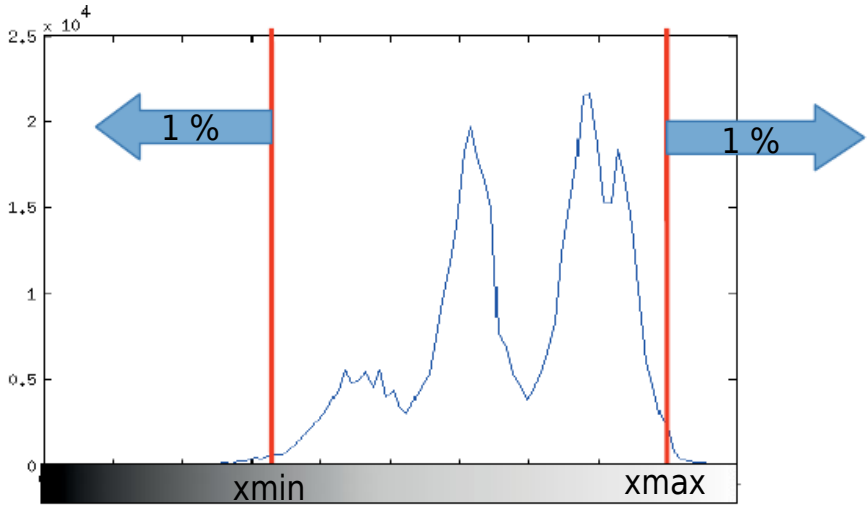


Figura 11. Histograma y determinación de xmin y xmax.

Para ello, a partir del histograma, y fijando un porcentaje de saturación, Psat, (el 1 % en la **Figura** 11), se seleccionan los valores xmin y xmax como aquellos que dejan por debajo y por encima solo al Psat por ciento de los píxeles. Una vez determinados estos valores, la curva de entrada/salida se construye como se muestra en la Figura 12.

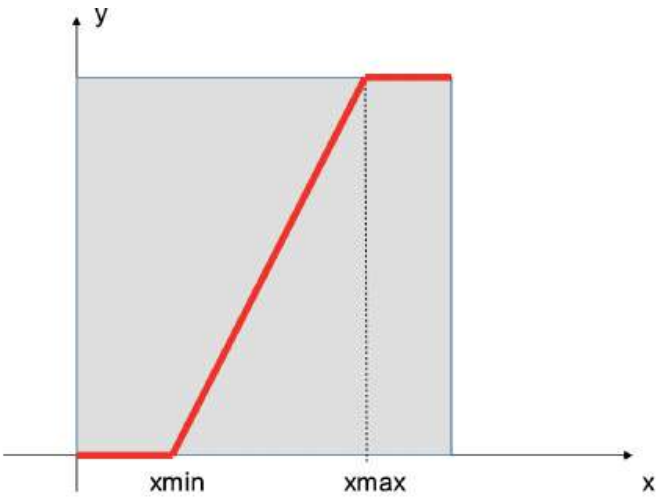


Figura 12. Curva entrada/salida para aumento lineal del contraste.

Los valores por debajo de xmin se transformarán en negro y los que estén por encima de xmax en blanco. El resto (la mayor parte) aumentarán su contraste, ya que la recta entre xmin y xmax tienen una pendiente mayor que la unidad. Obsérvese que tras la transformación tendremos una imagen que ocupará todo el rango de intensidades (a diferencia de lo que sucedía en la imagen original).

Ilustraremos con un ejemplo el funcionamiento del aumento lineal del contraste. En la parte izquierda de la Figura 13 vemos una imagen tal y como la capturó una cámara cuya misión era leer matrículas de vehículos. La exposición de la imagen es correcta, ya que el histograma aparece centrado. Sin embargo, resulta evidente que la imagen tiene poco contraste. El motivo del bajo contraste es que la cámara estaba tomando la imagen a través de un cristal sucio donde estaba incidiendo el sol. Un fenómeno muy parecido al que sucede si apuntamos la cámara de nuestro dispositivo Android a través del parabrisas de un coche.

Tras calcular el histograma y determinar xmin y xmax, es posible corregir la imagen **automáticamente** para obtener la imagen de la derecha. Nótese cómo la nueva imagen sí que cubre todo el margen de intensidades disponibles (ver histograma).

Insistimos en la idea de que la transformación se determine automáticamente pues en la práctica tendremos del orden de 15-30 imágenes por segundo y no es cuestión de ponerse a ajustar curvas manualmente.

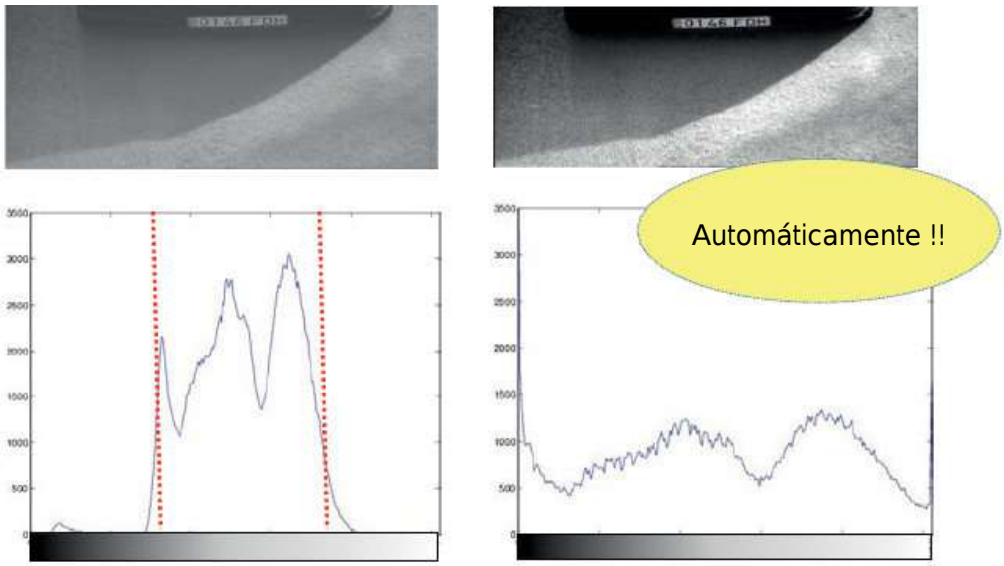


Figura 13. Ejemplo de aumento lineal de contraste.

Ejercicio: Aumento lineal del contraste.

1. Haz una copia del proyecto Base5mono.
2. En la clase `Procesador` coloca el siguiente código:

```
public class Procesador {
    MatOfint canales;
    MatOfint numero_bins;
    MatOfFloat intervalo;
    Mat hist;
    List <Mat> imagenes;
    float[] histograma;

    Procesador() {
        canales = new MatOfint(0) ;
        numero_bins = new MatOfint(256);
        intervalo = new MatOfFloat(0,256);
        hist = new Mat();
        imagenes = new ArrayList<Mat>() ;
        histograma = new float[256];
    }

    public Mat procesa(Mat entrada) {
        Mat salida = new Mat();
        imagenes.clear(); //Eliminar imagen anterior si la hay
        imagenes.add(entrada); //Añadir imagen actual
        imgproc.calcHist(imagenes, canales, new Mat(), hist,
                numero_bins, intervalo);
        //Lectura del histograma a un array de float
        hist.get(0, 0, histograma);
```

```
        //Calcular xmin y xmax
        int total_pixeles = entrada.cols()*entrada.rows();
        float porcentaje_saturacion = (float) 0.05;
        int pixeles_saturados = (int) (porcentaje_saturacion
                                                    * total_pixeles);
        int xmin=0;
        int xmax = 255;
        float acumulado = 0f;
        for(int n=0; n < 256; n++) { //xmin
            acumulado = acumulado + histograma[n];
            if(acumulado > pixeles_saturados) {
                xmin = n;
                break;
            }
        }
        acumulado = 0;
        for(int n = 255; n >=0; n--) { //xmax
            acumulado = acumulado + histograma[n];
            if(acumulado > pixeles_saturados) {
                xmax = n;
                break;
            }
        }
        //Calculo de la salida
        Core.subtract(entrada, new Scalar(xmin) , salida);
        float pendiente = ((float) 255.0) / ((float) (xmax-xmin));
        Core.multiply(salida, new Scalar(pendiente) , salida);
        return salida;
    }
}
```

En esta clase lo primero que nos encontramos son una serie de variables que hacen falta para calcular el histograma. Dichas variables se inicializan en el constructor. La función que calcula el histograma, `calcHist()`, recibe como argumentos:

1. Una lista de imágenes. En vez de recibir una imagen, permite que le pongamos una lista de imágenes para calcular el histograma conjunto. Como en nuestro caso queremos calcular el histograma de cada imagen, lo que hacemos es que en cada nuevo fotograma vaciamos la lista e introducimos la entrada actual.

```
imagenes.clear(); //Eliminar imagen anterior si la hay
imagenes.add(entrada); //Añadir imagen actual
```

2. Una lista de canales. Si las imágenes son monocromas, solo tienen un canal. Si son en color, tienen tres. En el caso de color, podemos calcular el histograma de un cierto canal (por ejemplo, de la componente verde) o de todos conjuntamente. En este argumento le indicamos de qué canal queremos calcular el histograma. En nuestro caso siempre será el "0" por ser imágenes monocromas.

```
        canales = new MatOfint(0) ;
```

La lista de canales se indica con una `MatOfint`, que es una clase derivada de `Mat` en la que los píxeles son números enteros de 32 bits. La inicializamos para que contenga un único entero que es el "0".

3. Una máscara, por si queremos calcular el histograma de ciertas partes de la imagen. En el caso de querer calcular el histograma de toda la imagen, se indica con una imagen vacía.

4. A continuación, indicamos dónde queremos dejar el resultado (`hist`).

5. Seguidamente indicamos el número de elementos del histograma y el rango de valores. El número de elementos coincide con el número de intensidades diferentes que tenemos, 256. El rango de intensidades es de 0 a 255. Nosotros hemos indicado:

```
numero_bins = new MatOfint(256);
intervalo = new MatOfFloat(0,256);
```

porque el primer valor es inclusive y el último exclusive.

A continuación se accede al histograma usando hist.get(). Los dos bucles que vienen a continuación sirven para determinar `xmin` y `xmax`.

Finalmente se calcula la salida aplicando la siguiente ecuación:

Salida = (entrada – xmin) * pendiente

que se corresponde con la gráfica de la Figura 12, teniendo en cuenta que las operaciones de suma/resta y multiplicación saturan los valores en el rango 0-255.

3. Si ejecutáramos el programa tal y como está, veríamos una imagen distinta de la que captura la cámara, aunque no nos sería fácil darnos cuenta. Con el fin de poder evaluar el efecto de lo que acabamos de hacer añadiremos una nueva función que generará una imagen donde la mitad izquierda será la entrada y la derecha la salida. De esa manera, viendo las dos mitades simultáneamente podremos hacernos una idea del efecto. Añade una nueva función en la clase Procesador:

```
void mitadMitad(Mat entrada, Mat salida) {
    if (entrada.channels() > salida.channels())
        imgproc.cvtColor(salida, salida, imgproc.COLOR_GRAY2RGBA);
    if (entrada.channels() < salida.channels())
        imgproc.cvtColor(entrada, entrada, imgproc.COLOR_GRAY2RGBA);
    //Representar la entrada en la mitad izquierda
    Rect mitad_izquierda = new Rect();
    mitad_izquierda.x = 0; mitad_izquierda.y = 0;
    mitad_izquierda.height = entrada.height();
    mitad_izquierda.width = entrada.width()/2;
    Mat salida_mitad_izquierda = salida.submat( mitad_izquierda );
    Mat entrada_mitad_izquierda = entrada.submat( mitad_izquierda );
    entrada_mitad_izquierda.copyTo(salida_mitad_izquierda);
}
```

En este código se puede ver el uso de la clase `Rect`, que es la que emplea OpenCV para definir una región rectangular, y de la función `submat`. También se ha introducido la función `copyTo`, que permite copiar los píxeles de una imagen a otra. Recuerda que `salida_mitad_izquierda` comparte los píxeles físicamente con `salida`.

4. Finalmente, modifica la actividad principal para llamar a esta función:

```
Mat salida = procesador.procesa(entrada);
procesador.mitadMitad(entrada, salida);
```

5. Prueba el programa. Deberías observar algo similar a la imagen siguiente cuando apuntes la cámara a una imagen con poco contraste. En el caso que se muestra, se aplicó la cámara a una mesa de madera

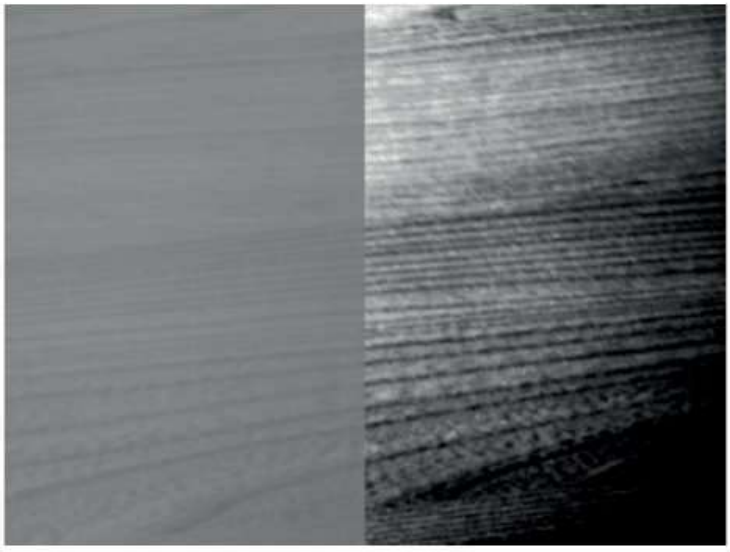


Figura 14. Resultado del aumento lineal de contraste automático.

2.1.3. Ecualización del histograma

La ecualización de histograma es otra técnica para mejorar el contraste de las imágenes. A diferencia de la anterior, en que la curva de entrada/salida tenía forma de recta, en la ecualización de histograma la curva de entrada/salida puede tener cualquier forma (monótona creciente). Se basa en tratar de lograr que la imagen de salida tenga una distribución lo más plana posible.

Curiosamente, la forma que debe tener la curva de entrada/salida coincide con el histograma acumulado desde 0 y adecuadamente normalizado:

$$f(X) = \frac{255}{S} \sum_{X_i=0}^{X} H(X_i)$$

Donde S es el número total de píxeles de la imagen:

$$S = \sum_{x=0}^{255} H(x)$$

En la Figura 15 se muestra el resultado de aplicar la ecualización de histograma a una imagen.

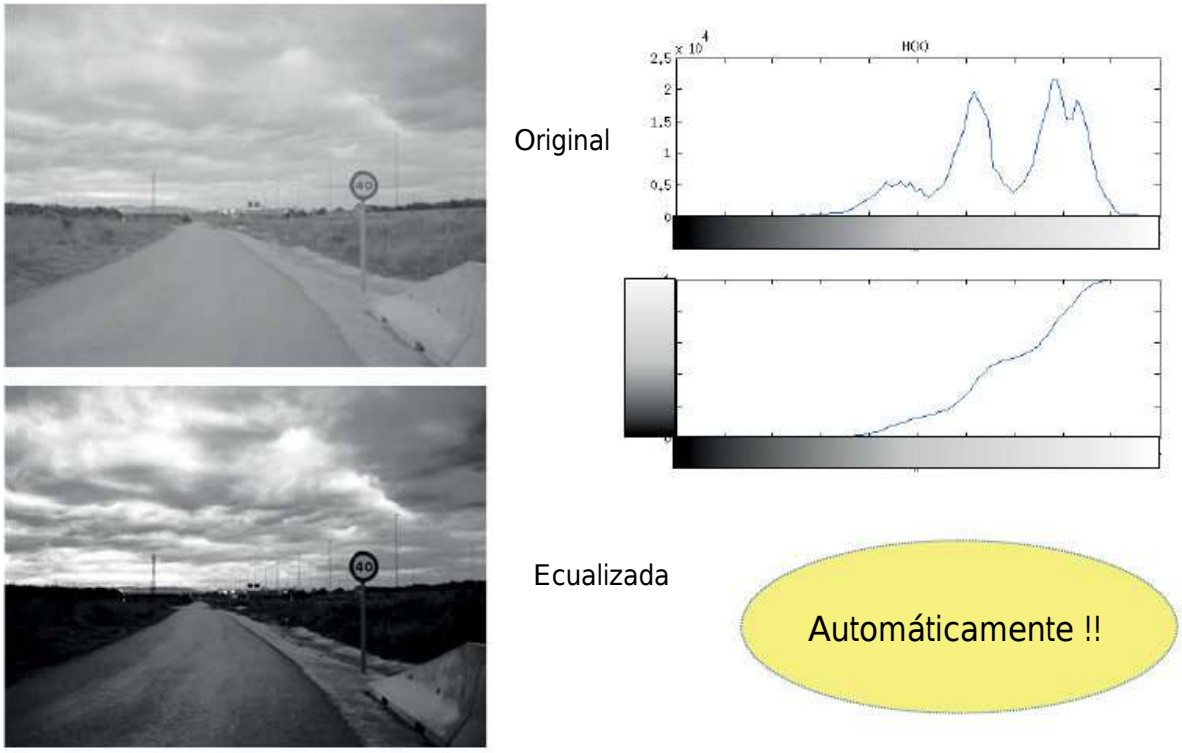


Figura 15. Ejemplo de ecualización de histograma. A la izquierda imágenes original y resultante. A la derecha tenemos el histograma de la imagen de entrada y la curva de entrada/salida obtenida automáticamente a partir del histograma.

Ejercicio: Ecualización de histograma .

1. Haz una copia del proyecto de aumento lineal de contraste.
2. En la clase Procesador, elimina las variables que se necesitaban para calcular el histograma, vacía el constructor y actualiza la función `procesa()`:

```
Procesador() { //Constructor
}

public Mat procesa(Mat entrada) {
    Mat salida = new Mat();
    imgproc.equalizeHist(entrada, salida);
    return salida;
}
```

3. Prueba el programa sobre un objeto que tenga poco contraste. Deberías observar algo similar a lo que se muestra en la siguiente figura:

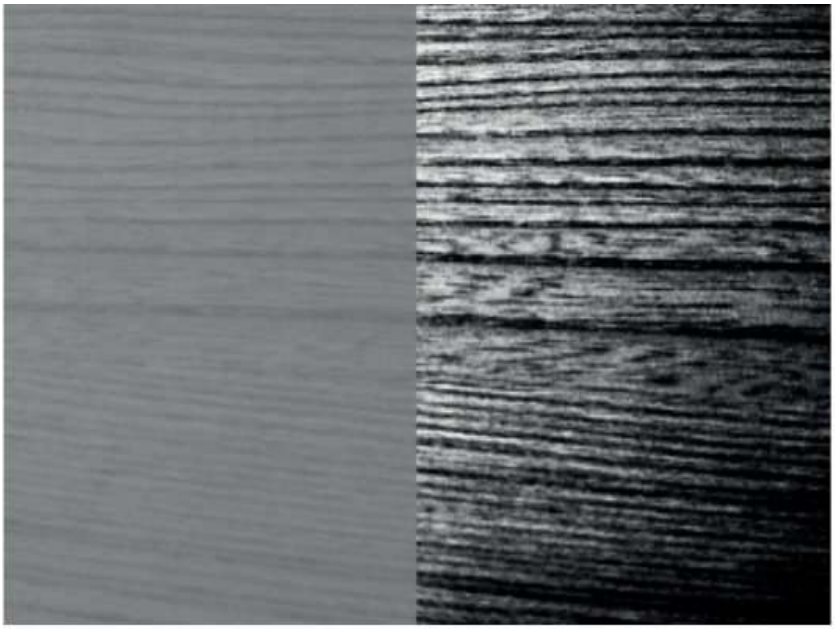


Figura 16. Comparación entrada-salida de la ecualización de histograma.

2.1.4. Del color al monocromo

Hemos visto cómo una imagen en color puede verse como la unión de tres imágenes monocromas, que es lo que se denominan componentes o canales. Una imagen en color ocupa tres veces más memoria que si es monocroma. Además, como tiene tres veces más píxeles, cualquier operación que intentemos realizar sobre una imagen en color será en general tres veces más costosa computacionalmente. Ahora bien, cabe preguntarse ¿en qué medida se podría reducir la imagen en color a una imagen monocromática reteniendo la información suficiente como para lograr nuestro propósito? Vamos a ver en este apartado distintas maneras de obtener una imagen monocroma a partir de una de color, cada una de ellas con ciertas propiedades. Las alternativas que veremos serán:

- La luminancia.
- Alguna de las componentes R, G o B.
- Alguna combinación lineal de las componentes R,G, B.
- Alguna combinación no lineal de las componentes R, G, B.

2.1.4.1. La luminancia

La luminancia corresponde a una combinación de las componentes RGB, que aproximadamente es proporcional a la percepción de brillo del ojo humano. Cuando vemos una imagen en blanco y negro en la televisión, lo que vemos es la luminancia

$$Y = 0{,}3R + 0{,}59\,G + 0{,}11\,B$$


Figura 17. Imagen en color y su correspondiente luminancia.

Es interesante observar que en la imagen monocroma se aprecian perfectamente:

- los números sobre el fondo blanco,
- la circunferencia del disco blanco de la señal.

Como veremos más adelante, estos elementos son los que nos van a permitir buscar la señal en la imagen.

Es interesante observar que en nuestros programas en Android es posible obtener directamente la luminancia a partir de la imagen nativa de Android usando el método `.gray()`.

2.1.4.2. Las componentes de color R, G y B

Una alternativa más simple que el cálculo de la luminancia podría ser tomar directamente una de las tres componentes de color, pero ¿cuál? Podría parecer a primera vista que, dado que el borde de la señal es rojo, podría ser interesante tomar esta componente.

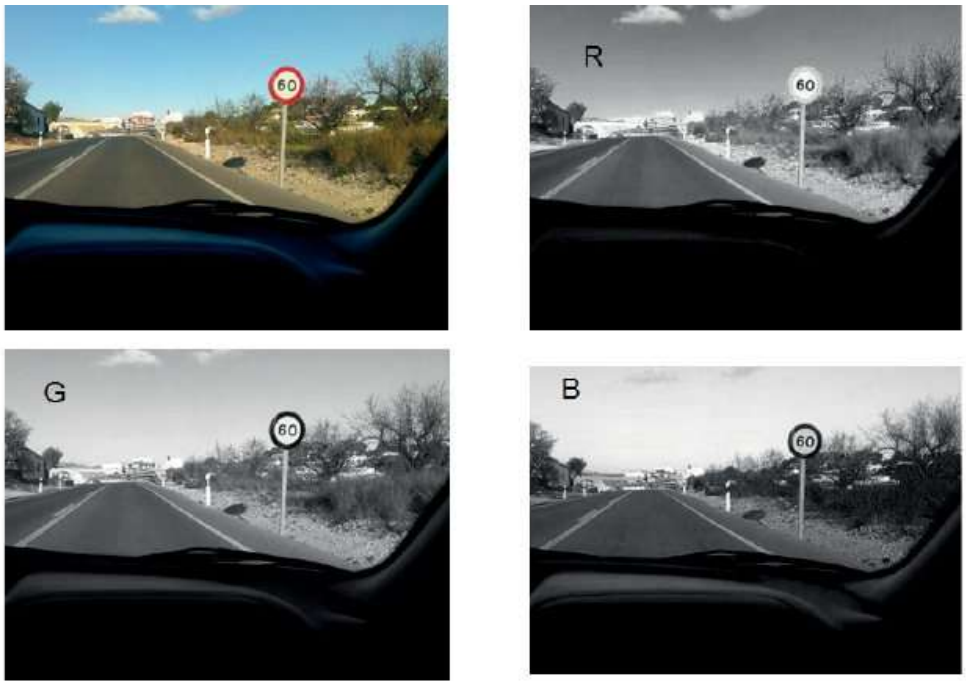


Figura 18. Componentes RGB de una imagen en color.

Como podemos ver en la Figura 18, resulta que en la componente roja es donde menos se aprecia un contraste entre el borde rojo de la señal y el disco blanco. Ello se debe a que los píxeles de color blanco tienen un valor de R=G=B que es alto. **Por tanto, la componente roja no permite diferenciar bien el borde de la señal de su interior.**

Cualquiera de las otras dos componentes, G, o B, permitirían obtener un mejor contraste entre el borde de la señal y el disco, o entre el disco y los números.

2.1.4.3. Combinación lineal de las componentes R, G y B

Los dos casos anteriores, la luminancia y una de las componentes R, G o B, constituyen casos particulares de una combinación lineal genérica de las componentes RGB:

$$e = a_R R + a_G G + a_B B$$

Podríamos pensar en otras combinaciones lineales que enfatizaran las características que tienen los objetos que vamos buscando. Una posibilidad para detectar zonas rojas sería:

$$e = R - Y = 0,7R - 0,59G - 0,11B$$

La idea detrás de esta elección es que los colores grises o poco saturados donde R = G = B, la característica C será nula. En las zonas rojas donde la componente G y B serán pequeñas, la característica tomará un valor grande, mientras que las zonas de otros colores, como por ejemplo azules o verdes, en los que la componente roja sea pequeña, la cantidad C tomará valores negativos. Normalmente, los valores negativos se ponen a cero para evitar problemas con números que son enteros sin signo de 8 bits (unsigned char).

Figura 19. Imagen en color y la combinación R-Y.
Los valores negativos de la resta se han saturdo a cero.

2.1.4.4. Combinación no lineal de las componentes R, G y B

En el apartado anterior hemos visto cómo realizar combinaciones lineales de las componentes de color con el fin de resaltar el borde rojo de la señal. También es posible realizar combinaciones no lineales de dichas componentes. Veremos dos posibilidades.

La primera posibilidad se basa en el hecho de que en el borde de la señal, rojo, habrá una componente (la roja) que será elevada mientras que las otras serán reducidas. Por el contrario en las blancas, negras o grises (asfalto) las tres componentes serán similares. Por tanto, se podría usar la siguiente característica para discriminar zonas coloreadas.

$$C = \max(R, G, B) - \min(R, G, B)$$

En las zonas rojas, el máximo será elevado y el mínimo reducido, por lo que la resta será elevada. En las zonas grises el máximo y el mínimo serán similares, por lo que la resta será aproximadamente nula. La Figura 20 muestra el resultado de esta idea.

Figura 20. Combinación no lineal de las componentes RGB con el fin de discriminar entre zonas coloreadas y zonas grises.

El problema del método anterior es que produciría valores altos siempre y cuando tuviéramos colores puros (rojo, verde, azul ‒) independientemente de cuál fuera el color. Es posible obtener otra característica que tenga en cuenta el hecho de que lo que buscamos es rojo:

$$C = R - \max(G, B)$$

Usando la anterior combinación de características RGB, solo cuando se de el hecho de que R sea elevado y ambos, B y G sean reducidos, obtendremos valores altos. La Figura 21 ilustra lo que se obtiene al usar esta característica, donde queda patente cómo se realzan las zonas rojas de la imagen.

Figura 21. Imagen en color y R -max(G.B)

2.1.5. Funciones OpenCV relevantes

En este apartado enumeraremos algunas funciones de OpenCV relacionadas con las ideas que acabamos de explicar. Para obtener más información, se recomienda consultar la documentación oficial de OpenCV[1].

- `calcHist()`: calcula el histograma de una imagen.
- `equalizeHist()`: obtiene la imagen resultado de la ecualización del histograma.
- `threshold()`: dada una imagen monocromática, aplica un umbral a los píxeles. Tiene varias opciones: un umbral fijo, un umbral calculado automáticamente usando Otsu, etc.
- `adaptiveThreshold()`: genera una imagen binaria, aplicando un umbral diferente a cada píxel de la imagen. Se explicará más adelante en qué consiste.
- `cvtColor()`: permite transformar entre distintos espacios de color. Cuando se pasa de RGB a YUV, la componente V es proporcional a R-Y.

[1] http://docs.opencv.org/java/

- `addWeighted()`: permite calcular combinaciones lineales de imágenes. Ver ayuda sobre el tema de las saturaciones (es decir, lo que pasa si el resultado es menor que 0 o mayor de 255).
- `add()`, `subtract()`: sumar y restar imágenes. Ver ayuda sobre el tema de las saturaciones.
- `split()`: dada una imagen en color, obtiene tres componentes.
- `merge()`: combina tres componentes de color en una imagen en color.

Ejercicio: Detección de zonas rojas

En este ejercicio implementaremos un procedimiento que destaque las zonas rojas de la escena. En otras palabras, implementaremos el código que nos genere una imagen donde los píxeles rojos en la imagen de entrada produzcan valores elevados, y donde los píxeles no rojos produzcan valores pequeños.

1. Haz una copia del programa Basico5.
2. Sobre dicha copia, modifica la clase Procesador.

```
public class Procesador {
    Mat red;
    Mat green;
    Mat blue;
    Mat maxGB;
    public Procesador() { //Constructor
        red = new Mat();
        green = new Mat();
        blue = new Mat();
        maxGB = new Mat();
    }

    public Mat procesa(Mat entrada) {
        Mat salida = new Mat();
        Core.extractChannel(entrada, red, 0);
        Core.extractChannel(entrada, green, 1);
        Core.extractChannel(entrada, blue, 2);
        Core.max(green, blue, maxGB);
        Core.subtract( red , maxGB , salida );
        return salida;
    }
}
```

3. Este programa recibe como entrada una imagen en color y genera una imagen monocroma como resultado. En la actividad principal añade la siguiente línea, justo antes de `return` :

```
if(salida.channels() == 1)
    imgproc.cvtColor(salida, salida, imgproc.COLOR_GRAY2RGBA);
return salida;
```

4. Prueba el programa apuntando a objetos de color rojo. El resultado debería ser similar al que se muestra en la Figura 22.

Figura 22. Resultado esperado del programa que detecta zonas rojas.

Práctica: Detección de zonas verdes.

1. Sobre la base del programa que acabas de crear, realiza una copia y modifícala adecuadamente para realzar las **zonas verdes** en vez de las rojas.

2.1.6. Resumen de las transformaciones de intensidad

En este apartado hemos visto:

- Cómo modificar automáticamente la intensidad de las imágenes para lograr mejor contraste.
- Cómo pasar de una imagen en color a una imagen monocroma, que ocupando la tercera parte de memoria, contenga o incluso realce la información que queremos detectar.
- Un breve listado de las funciones de OpenCV relacionadas con los conceptos vistos.

Preguntas de repaso: Transformaciones de intensidad

2.2. Preproceso: Operadores locales

En el capítulo anterior vimos cómo podíamos modificar una imagen con el fin de visualizarla mejor o para realzar ciertos atributos de color. Todos los métodos vis-

tos tenían en común el hecho de que la intensidad de un punto de la imagen de salida dependía únicamente del mismo punto de la imagen de entrada.

Hay ocasiones en las que lo que se desea es crear una imagen de salida, de modo que la intensidad en un punto de la misma dependa de cómo sea la imagen de entrada en un entorno del mismo punto. A este tipo de operadores se les llama operadores locales. En otras palabras, para calcular el valor de un punto de la imagen de salida, un operador local realiza ciertas operaciones con los píxeles alrededor de dicho punto en la imagen de entrada.

Figura 23. Concepto de operador local.

Algunas de las acciones que se pueden lograr usando operadores locales son:

- Realizar un promediado local.
- Detectar los contornos de los objetos.
- Mejorar la nitidez de la imagen.
- Realzar/eliminar objetos claros/oscuros de un cierto tamaño.

Para comprender mejor la idea de cómo funcionan los operadores locales consideremos la **Figura** 23. En dicha figura se ilustra cómo se calcula un píxel de la imagen de salida resaltado en azul. Sobre la imagen de entrada se sitúa una máscara centrada en la posición para la que estamos intentando calcular el valor de la salida. Dicha máscara consiste en una serie de coeficientes (1/9 en este caso) mostrados en color rojo en la figura. A continuación se multiplica cada coeficiente por el píxel de la imagen que recae debajo sumando finalmente el resultado de todos los productos. Para calcular todos los valores de los píxeles de la imagen de salida se va desplazando la máscara sobre la imagen de entrada.

Figura 24. Ejemplo de operador local. La salida en un píxel es la media de una región de tamaño 3 x 3 de la entrada.

Existen dos tipos básicos de operadores locales:

- Lineales: la operación que se realiza sobre los píxeles es una combinación lineal de los mismos. El ejemplo visto en la figura anterior corresponde a este tipo.
- No lineales: en este caso la operación que se realiza con los píxeles debajo de la máscara es de otro tipo. Algunos ejemplos de estos operadores son:
    - Máximo: el valor de la salida es el máximo de los valores de la entrada debajo de la máscara.
    - Mínimo: el valor de la salida es el mínimo de los valores de la entrada debajo de la máscara.
    - Mediana: el valor de la salida es la mediana de los valores de la entrada debajo de la máscara. Recordemos que la mediana es el valor que resulta en el medio cuando se ordenan los valores de menor a mayor.

2.2.1. Filtros lineales

En el apartado anterior hemos visto cómo operan los filtros lineales. Dependiendo del valor de los coeficientes, la operación realizada por el filtro tendrá una utilidad u otra:

- Promediadores (filtros paso-bajo): calculan un promedio local de los píxeles. Provocan un emborronamiento de la imagen.
- Filtros paso-alto: sirven para realzar los cambios de intensidad en la imagen.
- Realzador de contorno: producen imágenes más nítidas.
- Gradientes: determinan el modo en que varía la intensidad alrededor de cada punto. Pueden servir también como detectores bordes.

Seguidamente veremos cada uno de estos tipos de filtros lineales.

2.2.1.1. Filtros promediadores. Filtros paso-bajo

Como hemos comentado, su principal utilidad es realizar un promediado local de las intensidades de los píxeles. El valor de la salida en un punto puede verse como la media de los píxeles situados alrededor de la misma coordenada en la imagen de entrada.

Tienen como propiedad común que la suma de todos los coeficientes de la máscara debe ser uno para que el filtrado no altere el brillo medio de la imagen.

Si bien los coeficientes pueden diseñarse por diferentes métodos, en la práctica se utilizan dos tipos de filtros paso-bajo:

- Planos: todos los coeficientes de la máscara son iguales. Suelen usarse aquellos en los que la forma de la máscara es rectangular y el tamaño impar. La razón para el **tamaño impar** es que de ese modo la máscara se puede centrar exactamente alrededor de un punto. Es el caso de la figura anterior. La razón de la forma rectangular es que, si el filtro es plano y la máscara tiene forma rectangular, es posible una **implementación muy rápida** del filtrado que facilita el funcionamiento en tiempo real.
- Gaussianos: los coeficientes del filtro siguen la ecuación de una gaussiana

$$h(x, y) = K\exp\left(-\frac{x^2 + y^2}{2a^2}\right)$$

El grado de promediado (y de emborronamiento) se controla con el tamaño de la máscara en el caso de los filtros planos y con el parámetro σ en el caso gaussiano. En el caso gaussiano, la extensión de la máscara es infinita. En la práctica, no obstante, decrece muy rápidamente para valores de (x,y) mucho mayores que 3 σ y se suelen emplear aproximaciones finitas. En la Figura 25 puede observarse en 3D la forma que toman los coeficientes de la máscara para el caso de un promediador plano y uno gaussiano.

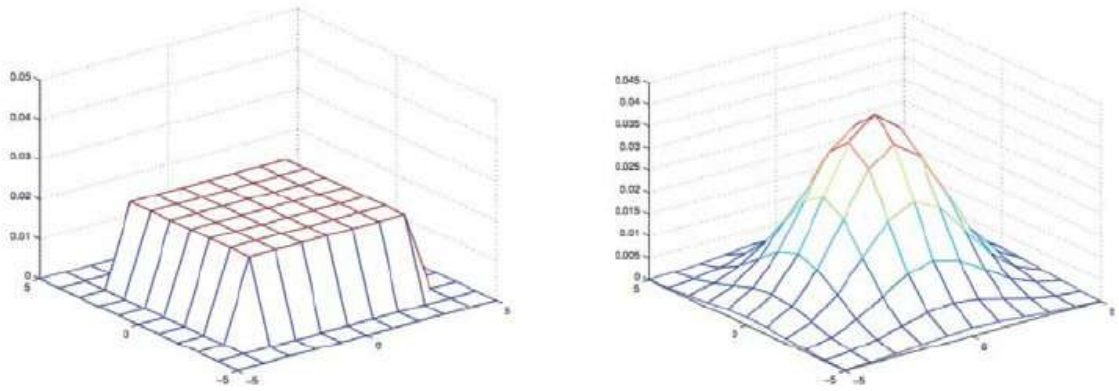


Figura 25. Forma de la máscara en promediadores planos y gaussianos. Izquierda plano de 7 x 7. Derecha, máscara gaussiana con a = 2 truncada a 11 x 11.

El mayor inconveniente de los filtros gaussianos es que son más costosos computacionalmente. No obstante, los filtros gaussianos proporcionan mejores resultados debido a que a la hora de promediar ponderan más los píxeles próximos al centro de la máscara. Se puede considerar que un filtrado plano con una máscara de MxM produce el mismo promediado que un filtro gaussiano con

$$a = \frac{M}{3.5}$$

En la Figura 26 se compara el resultado de filtrar una misma imagen usando un filtro plano y otro gaussiano cuya σ se ajusta a la ecuación anterior.

Aparentemente el aspecto de las dos imágenes es similar. Sin embargo podemos observar en el detalle cómo el suavizado plano ha producido un artefacto: en realidad en la escena hay dos ramas horizontales que se cruzan con dos verticales y en la imagen de la derecha parece haber una tercera rama central.

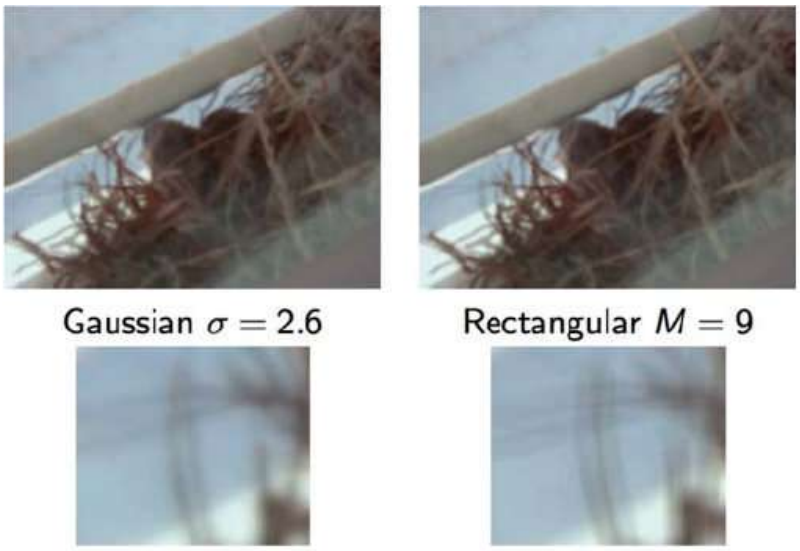


Figura 26. Comparación entre suavizado plano y gaussiano.

Aspectos de implementación. Separabilidad

Un aspecto que conviene tener en cuenta es ¿cuántas operaciones hay que realizar para calcular cada punto de la salida? Es inmediato darse cuenta que, en principio, tendremos que realizar tantos productos y sumas como puntos tenga la máscara. Por tanto, si la máscara tiene un tamaño M x M realizaremos M^2 operaciones por píxel.

Tanto en el caso de filtros planos como gaussianos es posible descomponer la máscara como un vector fila por otro columna. En la figura siguiente se ilustra la descomposición para un filtro plano de 3 x 3.

	1/3	1/3	1/3
1/3	1/9	1/9	1/9
1/3	1/9	1/9	1/9
1I3	1/9	1/9	1/9

En el caso de que la máscara sea gaussiana, también se puede realizar dicha descomposición:

$$h(x, y) = K\exp\left(\frac{x^2 + y^2}{2a^2}\right) = \left[\sqrt{K}\exp\left(-\frac{x^2}{2a^2}\right)\right]\left[\sqrt{K}\exp\left(\frac{y^2}{2a^2}\right)\right]$$

En el caso de que la máscara se pueda descomponer de este modo, y en el caso de suavizadores planos y gaussianos sí es posible, se puede realizar el filtrado procediendo primero con una máscara horizontal (de una fila) y filtrando de nuevo el

resultado obtenido con una máscara vertical (de una columna). En la Figura 27, se ilustra el proceso. Para obtener la imagen filtrada (abajo-derecha) se realiza primero un suavizado horizontal (arriba-derecha) y en un segundo paso se filtra este resultado con un filtro vertical para obtener la imagen final. El mismo resultado se habría obtenido si primero se realiza un suavizado vertical (abajo-izquierda) y luego el horizontal.

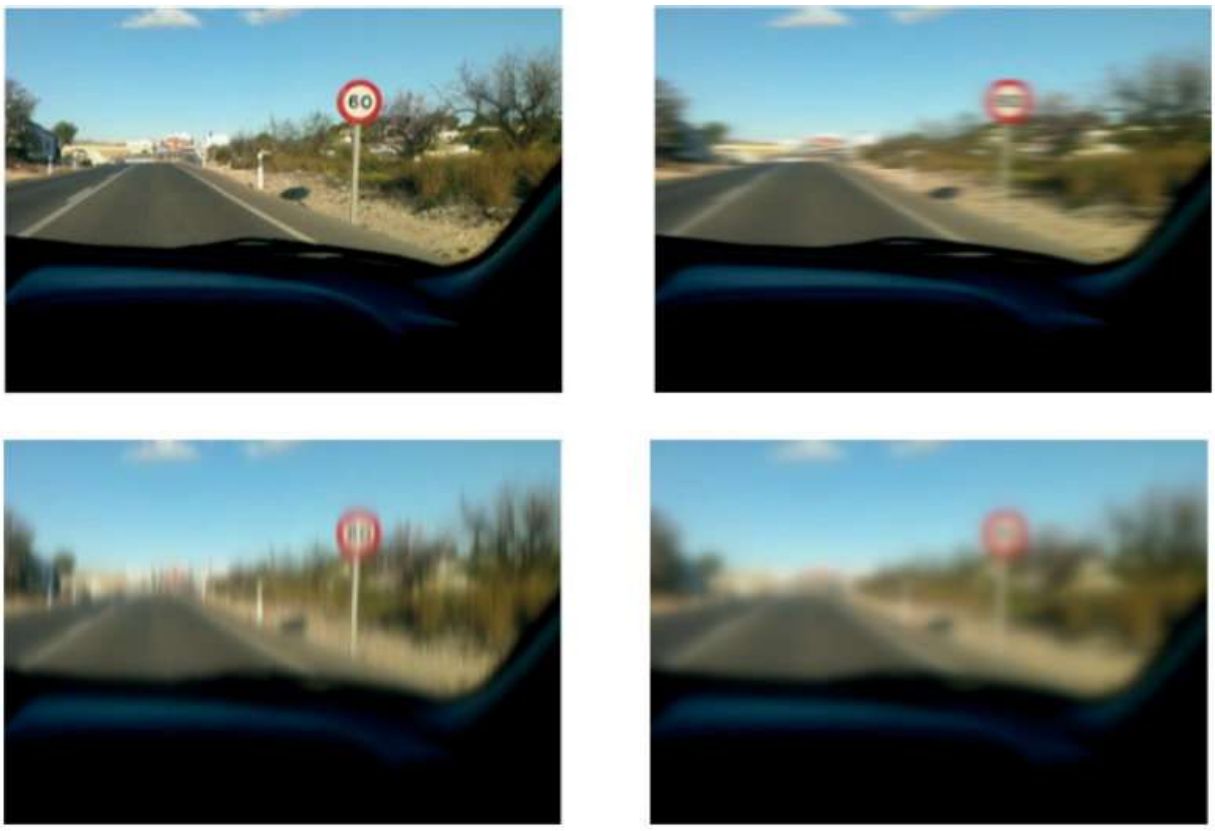


Figura 27. Filtrado usando separabilidad. Arriba-izquierda: imagen Original. Arriba-derecha: resultado de filtrar en horizontal. Abajo-izquierda: resultado de filtrar en vertical. Abajo-derecha: suavizado final

El uso de la separabilidad permite reducir el número total de operaciones por píxel de M^2 a $2M$. Si tratamos de estimar cómo puede ser el ahorro, en el caso de una máscara de 11 x 11 pasaríamos de 121 ops/.píxel a 22 ops./píxel, es decir, una ganancia por un factor aproximadamente igual a 6.

Estos detalles son vitales cuando se quieren realizar operaciones en tiempo real o sobre hardware de prestaciones limitadas.

Aspectos de implementación: imágenes en color

Otro aspecto a tener en cuenta es el de las imágenes en color. Cuando se filtran imágenes en color que constan de tres canales (R,G y B), al filtrar lo que se tiene que hacer es repetir el mismo proceso para cada una de las componentes de color. Esto implica que **filtrar una imagen en color sea tres veces más costoso computacionalmente que filtrar una imagen monocroma**. Por ello, en la prácti-

ca, salvo que tengamos una buena razón para hacerlo, preferiremos filtrar versiones monocromáticas de las imágenes en vez de su versión en color.

Aspectos de implementación: funciones OpenCV

Como hemos comentado anteriormente, filtrar es una de las operaciones más costosas computacionalmente que tendremos que llevar a cabo. Siempre que lo tengamos que hacer, deberemos buscar la manera más eficiente de hacerlo.

Realizar bucles sobre los píxeles y los elementos de la máscara en Java resultaría extremadamente lento. Cuando tengamos que realizar un filtrado usaremos funciones de la librería OpenCV, que está escrita en C y perfectamente optimizada. Las funciones que conviene conocer son:

- Filtrado plano: `imgproc.blur()` ,`imgproc.boxFilter()`
- Filtrado Gaussiano: `imgproc.gaussianBlur()`
- Filtrado con máscaras genéricas: `imgproc.filter2D()`, `imgproc.sepFilter2D()`

Para más detalles sobre estas funciones, consulta la documentación oficial[2].

2.2.1.2. Filtros paso-alto

Se puede decir que los filtros paso-bajo, promediadores, que hemos visto en el apartado anterior, eliminan información de los detalles de la imagen y la emborronan.

En ciertas ocasiones son esos detalles los que nos interesan. La manera más sencilla de obtenerlos es restarle a la imagen original la imagen resultante del suavizado. La Figura 28 ilustra cómo obtener el filtrado paso-alto como diferencia entre la imagen original y el filtrado paso-bajo.

[2] http://docs.opencv.org/java/

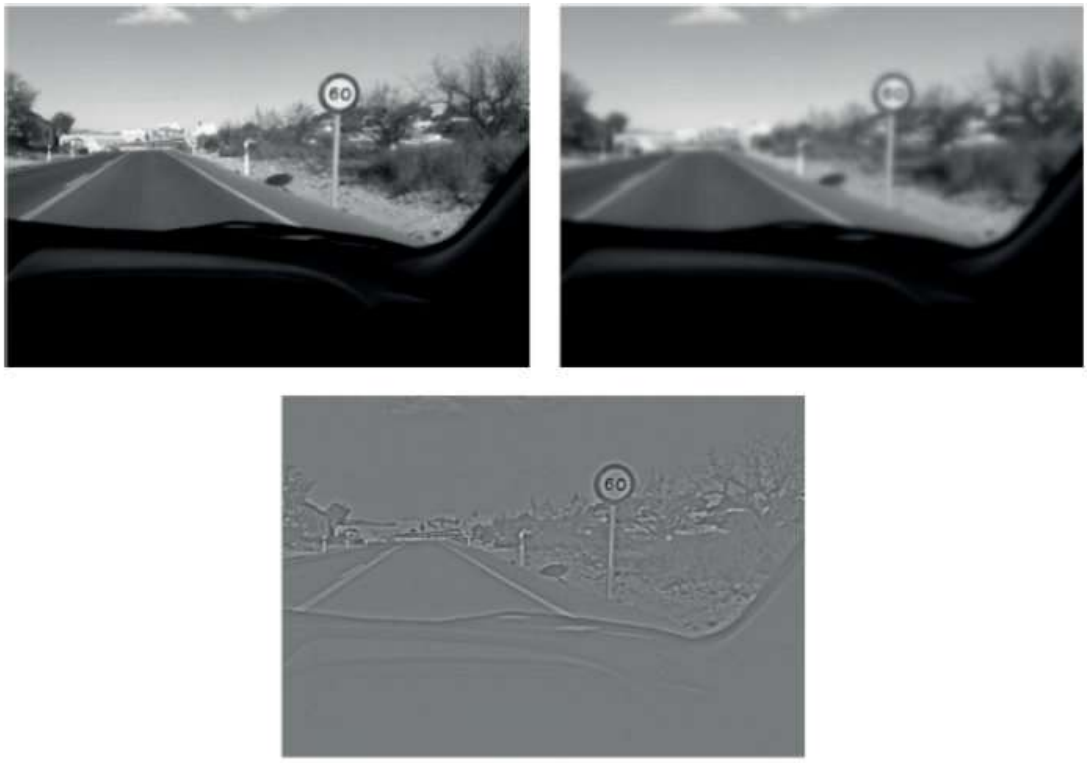


Figura 28. Filtrado paso-alto como diferencia entre la imagen original y la imagen suavizada.

A la hora de calcular la resta entre las imágenes hay que ser cuidadoso, ya que dicha diferencia puede tomar valores positivos y negativos. En la imagen que se muestra en la Figura 28 se ha añadido 128 al valor de la diferencia con el fin de poder visualizarla adecuadamente. Los píxeles grises representan valores nulos de la imagen paso-alto. Los que son más oscuros que 128 corresponden a valores negativos, mientras que los que son más claros a valores positivos.

Para calcular el filtrado paso-alto en OpenCV escribiríamos:

```
//Mat original;
Mat paso_bajo = new Mat();
Size s=new Size(7,7);
imgproc.blur( original, paso_bajo, s);
Mat paso_alto= new Mat();
Core.subtract( original, paso_bajo, paso_alto );
```

Ahora bien, cuando el tipo de datos de los píxeles en OpenCV es unsigned char (caso usual), al hacer la resta de cada píxel, OpenCV efectúa una saturación del resultado, de modo que para evitar problemas los valores negativos se truncan a cero. En el código anterior aparece la clase `Size`. Dicha clase la emplea OpenCV para referirse al tamaño de una imagen o región. Tiene dos campos: `width`, que contiene la anchura de la zona, y `height`, que contiene la altura.

Para obtener un resultado como el de la Figura 28 se debería convertir la imagen a un tipo de dato que admita números negativos (float, por ejemplo):

```
Mat original_float = new Mat();
```

```
original.convertTo( original_float , CvType.CV_32FC1);
Mat paso_bajo_float = new Mat();
Size s=new Size(7,7);
imgproc.blur( original_float, paso_bajo, s);
Mat paso_alto_float = new Mat();
Core.subtract( original_float, paso_bajo_float, paso_alto_float );
```

No obstante, a veces nos interesarán solo los detalles de un signo. En ese caso, si deseamos la parte positiva del filtrado paso-alto bastará con realizar la el resto (OpenCV mantendrá a cero las diferencias negativas). Si lo que deseamos fuera la parte negativa del filtrado paso-alto, bastará con restar en orden inverso:

```
//Mat original;
Mat paso_bajo = new Mat();
Size s=new Size(7,7);
imgproc.blur( original, paso_bajo, s);
Mat paso_alto_pos = new Mat();
Core.subtract( original, paso_bajo, paso_alto_pos);
Mat paso_alto_neg = new Mat();
Core.subtract( paso_bajo, original, paso_alto_neg);
```

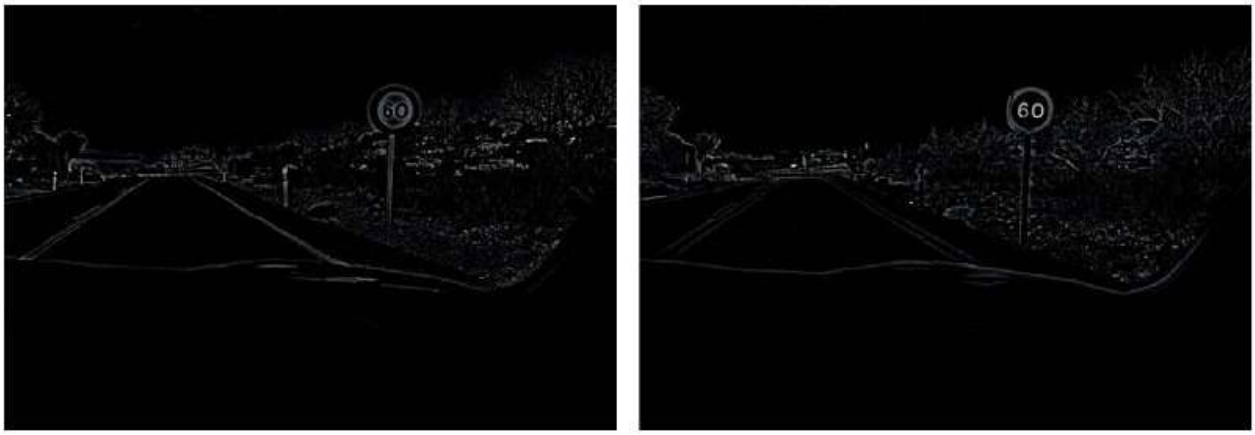


Figura 29: Parte positiva (izquierda) y negativa (derecha) del filtrado paso-alto.

Es interesante observar la aplicación del filtrado paso-alto para nuestro problema de detección de señales. Obsérvese, por ejemplo, cómo los números de la señal, que se habían borrado parcialmente en la imagen paso-bajo, se muestran en la imagen paso-alto. En la imagen paso-alto positiva, los números aparecen como zonas negras rodeadas de valores positivos. En la imagen paso-alto negativa, los números se muestran como zonas claras rodeadas por píxeles negros. El hecho de que los números aparezcan claramente diferentes de lo que les rodea permitirá, como veremos más adelante, detectarlos fácilmente.

2.2.1.3. Realce de contornos

Acabamos de ver cómo la imagen paso-alto contiene los detalles finos de la imagen. Si dichos detalles finos se suman de nuevo a la imagen original, lo que se

obtiene es una imagen con eoss detalles amplificados. En otras palabras, se logra una imagen con unos contornos más definidos.

Debe tenerse precaución, ya que esta operación no solo amplifica los detalles de la imagen sino el ruido, por lo que puede producir imágenes más ruidosas.

Realzada = Original + α Paso-alto

En la Figura 30 se muestra el efecto del realce de contornos:

Figura 30. Imagen original (izquierda) y con contornos realzados (derecha).

Sobre la gestión de memoria

Un aspecto a tener en cuenta en las aplicaciones de análisis de imagen, es que cuando termina una función en la que hemos hecho `new Mat()`, debería liberarse la memoria inmediatamente al salir de la función (destructor de la clase Mat). Sin embargo, el garbage collector de Android retrasa esta acción hasta que la CPU esté relativamente libre. Como analizar las imágenes es costoso, puede suceder que el programa se detenga por falta de memoria al no haberse liberado las imágenes anteriores. Hay dos posibles soluciones:

- Que todas las imágenes de trabajo temporal pertenezcan al objeto procesador. De ese modo el programa no tiene la necesidad de reservar memoria cada nueva imagen.

```
Public class Procesador {
    private Mat tmpimg;

    public Procesador() {
        tmpimg = new Mat();
    }

    public Mat procesa(Mat entrada) {
         ...
        tmpimg = 2* entrada;
        return entrada;
    }
}
```

- Que antes de abandonar el método, liberemos la memoria explícitamente usando el método `release()` de la clase `Mat`.

```
Public class Procesador {
    private

    public Procesador() {
    }

    public Mat procesa(Mat entrada) {
        Mat tmpimg = new Mat();
         ...
        tmpimg = 2* entrada;
        ...
        tmpimg.release();
        return entrada;
    }
}
```

Ejercicio: Obtener imagen que resalte los números de la señal

En este ejercicio implementaremos un filtrado paso-alto en el que la mayor parte del resultado sean píxeles negros y en el que los números de una señal aparezcan claros.

1. Haz una copia del programa "Base5mono".
2. Sobre dicha copia, en la clase `Procesador`, añade la siguiente variable de trabajo e inicialízala en el constructor:

```
Mat  paso_bajo;
public Procesador() {
    paso_bajo= new Mat();
}
```

3. Modifica la función `procesa()` de la clase `Procesador` para realizar el filtrado pedido añadiendo las líneas siguientes.

```
public Mat procesa(Mat entrada) {
    Mat salida = new Mat();
    int filter_size = 17;
    Size s=new Size(filter_size,filter_size);
    imgproc.blur(entrada, paso_bajo, s);
        // Hacer la resta. Los valores negativos saturan a cero
    Core.subtract(paso_bajo, entrada, salida);
        //Aplicar Ganancia para ver mejor. La multiplicacion satura
    Scalar ganancia = new Scalar(2);
    Core.multiply(salida, ganancia, salida);
```

```
    return salida;
}
```

En primer lugar, se realiza un filtrado paso-bajo usando la función blur().

Seguidamente realizamos la resta con la imagen original para obtener el filtrado paso-alto. Obsérvese que al hacer la resta, los valores negativos de la diferencia se saturan a cero.

A continuación, multiplicamos el resultado por 2 para visualizar mejor el resultado. Este producto es con saturación, lo que quiere decir que si al multiplicar excedemos el valor máximo de 255, el resultado será 255. Obsérvese también el uso de `Scalar` para realizar la multiplicación por una constante.

4. Prueba el programa realizado sobre el dispositivo:
   - Sobre una imagen de una imagen de una señal impresa.
   - Sobre texto manuscrito en una hoja en blanco.
5. Prueba a variar tamaño filter size, por ejemplo, a 11, 17, 21_, y compara los resultados que se obtienen. Observa también si cambia el número de imágnes por segundo.
6. Con la opción del menú, guarda algunos pares entrada-salida y analiza los resultados.

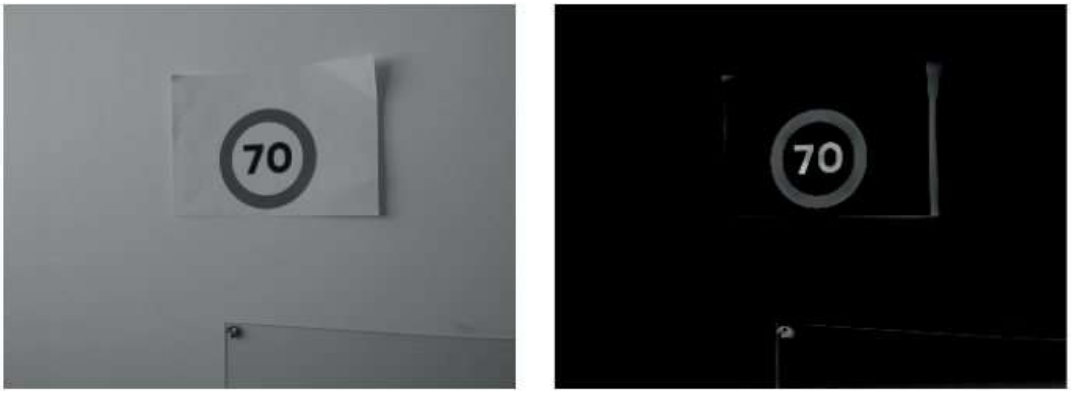


Figura 31. Imagen original y resultado del filtrado paso-alto.

Práctica: Filtros lineales

Partiendo del ejemplo anterior:

1. Realiza el filtrado paso-bajo en vez del paso-alto. Básicamente tendrás que quitar algunas instrucciones.
2. Realiza el filtrado paso-alto negativo, pero empleando un promediador gaussiano equivalente al empleado en el código anterior.
3. Calcula el filtrado paso-alto positivo.

Preguntas de repaso: Filtros Lineales

2.2.1.4. Gradientes

Cuando uno maneja funciones unidimensionales, surge el concepto de derivada. La derivada nos indica cómo varía la función localmente. Así, si la derivada es cero, estaremos en una zona donde la función es aproximadamente constante. Si la derivada es positiva, indicará que la función es creciente en ese punto, y lo contrario si es negativa.

Si se desea extender la idea a funciones multidimensionales, surge el concepto de gradiente. En el caso de dos dimensiones, el gradiente en cada punto es un vector.

- La primera componente del vector nos indica cómo varía la imagen en un punto al movernos hacia la derecha: será positivo si se hace más clara, negativo si se hace más oscura o cero si no varía. A esto se le llama derivada parcial en la dirección x.
- La segunda componentes nos indica cómo varía la imagen en un punto al movernos hacia abajo: será positivo si se hace más clara, negativo si se hace más oscura o cero si no varía. A esto se le llama derivada parcial en la dirección y.

$$\overrightarrow{Grad}(x, y) = \left[G_x(x, y), G_y(x, y)\right]$$

Obsérvese que **el gradiente en cada punto es un vector**. Dicho vector se puede representar en sus componentes cartesianas, como acabamos de ver, o en forma polar, con un módulo y un ángulo.

Los gradientes tienen muchas aplicaciones en el análisis de imagen. El que vamos a usar nosotros tiene que ver con la detección de transiciones rápidas en la intensidad de la imagen. En la Figura 32 se muestra una señal unidimensional que consiste en una zona de valor uno rodeada de valores nulos. Si fuera una imagen, lo que veríamos sería algo claro rodeado por una zona oscura. Una manera de detectar la «frontera» entre la zona clara y oscura es mediante el gradiente. Para ello, basta con tomar el módulo del gradiente. Dicho módulo tendrá un valor elevado en las zonas donde haya transiciones rápidas y será prácticamente nulo en el resto.

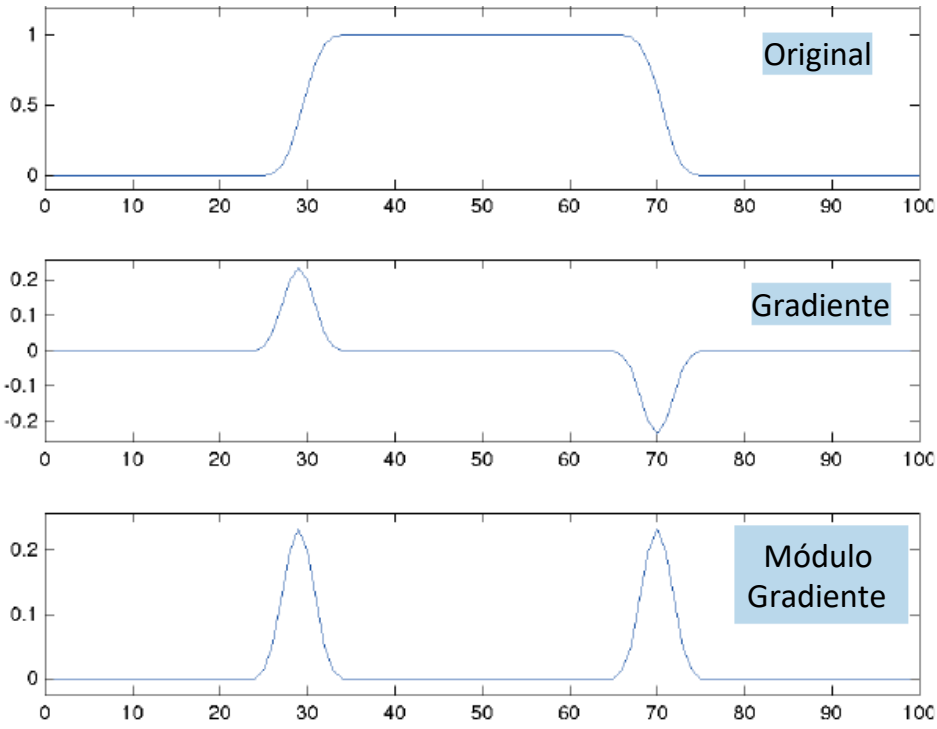


Figura 32. Gradientes en una dimensión. Uso para detectar transiciones.

Para extender la anterior idea al caso bidimensional lo que hay que hacer es reemplazar el módulo de la derivada por el módulo del vector gradiente:

$$\left|\overrightarrow{Grad}\,I(x,y)\right| = \sqrt{G_x^2(x,y) + G_y^2(x,y)}$$

El módulo del gradiente será muy pequeño en las zonas uniformes de la imagen y tomará valores altos en las zonas donde se produzcan transiciones rápidas. En la Figura 33 se ilustra el aspecto del módulo del gradiente. Obsérvese que el resultado de esta operación contiene un conjunto de puntos brillantes justo en la frontera del círculo rojo de la señal. Esto será de utilidad, como veremos más adelante, para detectar la ubicación de la señal.

En una imagen de una señal, el nivel de intensidad será muy diferente dependiendo de la iluminación de la escena, el tiempo de exposición, etc. Por tanto, resulta imposible determinar dónde está la señal en base a su color. Sin embargo, en una imagen de una señal, siempre aparecerá una transición rápida en el borde del círculo rojo que dará lugar a un valor alto del módulo del gradiente. Este hecho, junto con la propiedad de que los puntos brillantes del módulo del gradiente forman un círculo, será la clave para detectar la señal.

Figura 33. Imagen original y módulo del gradiente

Implementación de los gradientes

Hemos dicho que el gradiente corresponde a unas derivadas parciales, pero ¿cómo calculamos derivadas en una imagen digital? La respuesta es mediante aproximación con diferencias finitas. Así:

$$G_x(x, y) = \frac{f(x+1, y) - f(x-1, y)}{2}$$

$$G_y(x, y) = \frac{f(x, y+1) - f(x, y-1)}{2}$$

Esto corresponde a un filtrado lineal con una máscara como la siguiente (gradiente horizontal):

0	0	0
-1	0	1
0	0	0

Hay veces que se emplean máscaras más sofisticadas y que proporcionan prestaciones ligeramente mejores como:

-1	0	1
-2	0	2
-1	0	1

Esta máscara, por ejemplo, recibe el nombre de máscara de Sobel y calcula derivadas en sentido horizontal a la vez que un cierto suavizado en sentido vertical.

Gradientes en OpenCV

La función para calcular gradientes en OpenCV recibe el nombre de `Sobel()`.
Dicha función permite calcular derivadas en la dirección que se desee:

```
// Mat  original
Mat Gx = new Mat();
Mat Gy = new Mat();
imgproc.Sobel( original, Gx, CvType.CV_32FC1 , 1, 0);
                                                //Derivada primera rto x
imgproc.Sobel( original, Gy, CvType.CV_32FC1 , 0, 1);
                                                //Derivada primera rto y
```

Además de la imagen de entrada y de salida, se tiene que indicar el tipo de dato que deseamos para la imagen de salida (float en el ejemplo de arriba), ya que los gradientes son de nuevo cantidades con signo y si usáramos el tipo de dato por defecto, unsigned char, tendríamos problemas con los valores negativos. Los dos últimos argumentos se refieren a si queremos derivar según x o según y, y el orden de la derivada (esta misma función permite calcular derivadas segundas).

Si lo que nos interesa es el módulo del gradiente, podemos calcularlo así:

```
Mat ModGrad = new Mat();
Mat AngGrad = new Mat();
Core.cartToPolar( Gx , Gy , ModGrad, AngGrad);
```

Si no nos interesara calcular el ángulo de los vectores gradiente, podríamos ahorrarnos calcular un arco-tangente por píxel calculando solo el módulo:

```
// Mat original
Mat Gx2 = new Mat();
Mat Gy2 = new Mat();
Core.multiply(Gx, Gx , Gx2); //Gx2 = Gx*Gx elemento a elemento
Core.multiply(Gy, Gy , Gy2); //Gy2 = Gy*Gy elemento a elemento
Mat ModGrad2 = new Mat();
Core.add( Gx2 , Gy2, ModGrad2);
Mat ModGrad = new Mat();
Core.sqrt(ModGrad2,ModGrad);
```

Adicionalmente, dependiendo de lo que queramos hacer, a veces es posible prescindir de la raíz cuadrada y ahorrarnos operaciones[3].

Ejercicio: Detección de contornos usando gradientes

En este ejercicio, implementaremos un filtrado de gradiente para detectar los bordes (transiciones rápidas) en una imagen monocromática. El objetivo es calcular el módulo del gradiente de cada fotograma que capture la cámara. La imagen que se visualizará será similar a la de la Figura 33.

1. Haz una copia del programa Base5mono.

[3] No debe confundirse la operación `multiply()` que acabamos de ver con `Gx * Gx`. Esta segunda fórmula calcula el producto en sentido matricial, mientras que `multiply()` lo hace multiplicando elemento a elemento.

2. Sobre dicha copia, modifica en la clase `Procesador` la función `procesa()` para que se realice el filtrado pedido. En este ejercicio deberás ser tú el que escriba el código. Todas las instrucciones necesarias las has visto anteriormente en los ejemplos, revísalos. Los pasos que debes seguir son:
    - Calcular la derivada horizontal (como flotante) usando un filtro de Sobel.
    - Calcular la derivada vertical (como flotante) usando un filtro de Sobel.
    - Calcular el módulo del gradiente (como flotante)
    - Convertir el resultado a unsigned char. En este punto se tendrá una imagen monocromática de unsigned char.
    - Si necesitas usar imágenes auxiliares, decláralas en la clase Procesador e inicialízalas en el constructor, como se vio en el ejemplo del Filtrado paso-alto.
3. Prueba el programa realizado sobre el dispositivo:
    - Sobre una señal impresa. Guarda una imagen de lo que se observa.
    - Apuntando el teléfono en cualquier dirección. Deberás observar una imagen con fondo negro y líneas en blanco.
4. Anota el número de imágenes por segundo.

Práctica: Gradiente componente verde y rojo.

En el ejercicio anterior se ha calculado el módulo del gradiente sobre la luminancia. Realiza copias del programa anterior y modifícalas para:

1. Calcular el gradiente de la componente verde.
2. Calcular el gradiente de la componente roja.

En ambos casos, ejecuta el programa sobre la señal sobre una señal impresa. Compara la intensidad que tiene la línea en la frontera del círculo rojo dependiendo de si se usa la luminancia, la componente verde o la componente roja.

2.2.1.5. Conclusión sobre filtros lineales

A modo de conclusión, podemos afirmar que los filtros lineales producen resultados que dependen de las relaciones de intensidad entre los píxeles de un vecindario.

En la práctica, dichas relaciones resultan más interesantes que los niveles de intensidad de los píxeles considerados aisladamente. Así, por ejemplo, no es posible saber qué nivel de intensidad tendrán los números de la señal (depende mucho de la iluminación, la cámara, etc.). Sin embargo, lo que sí es seguro es que serán más oscuros que el fondo de la misma. Del mismo modo, sabemos que entre el borde rojo de la señal y el fondo blanco de la misma existirá una variación

rápida de la intensidad. Este tipo de relaciones son las que enfatizan (o atenúan) los filtros lineales y de ahí su utilidad.

2.2.2. Filtros morfológicos

Una clase de filtros con unos principios de funcionamiento totalmente diferentes son los llamados filtros morfológicos. Los filtros morfológicos son operadores locales del estilo de los mostrados en la Figura 23, en los que la operación que se realiza con los píxeles debajo de la máscara es tomar el máximo o el mínimo.

En este contexto se denomina **elemento estructurante** a la máscara que se desplaza por la imagen de entrada.

Existen dos tipo básicos de filtros morfológicos:

- **Erosión:** cuando la operación que se realiza con los píxeles bajo el elemento estructurante es tomar la intensidad mínima.
- **Dilatación:** cuando la operación que se realiza con los píxeles bajo el elemento estructurante es tomar la intensidad máxima.

Este tipo de filtros solo se aplican sobre imágenes monocromáticas (de grises) y no tienen sentido con imágenes en color.

A continuación veremos cada uno de estos filtros y su posible utilidad en nuestro problema de reconocimiento de señales.

2.2.2.1. Erosión

Cuando la operación que se realiza con los píxeles consiste en tomar el mínimo de los mismos, la operación se denomina **erosión**.

Una primera observación que se puede hacer es que cada píxel del resultado de una erosión es de una intensidad menor o igual que el mismo píxel de la imagen de entrada.

Se denomina erosión porque los objetos claros de la imagen se reducen de modo similar a lo que ocurre con la erosión de una roca.

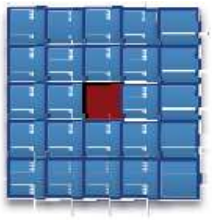


Figura 34. Elemento estructurante cuadrado de 5 x 5 que muestra su centro.

La Figura 34 muestra un elemento estructurante (EE) cuadrado de 5 x 5. En dicho EE se ha resaltado el píxel central también llamado centro o punto de anclaje (anchor point). Aunque es posible elegir cualquier punto como centro, normalmente dicho punto se corresponde con el centro del EE, y nosotros lo haremos siempre así.

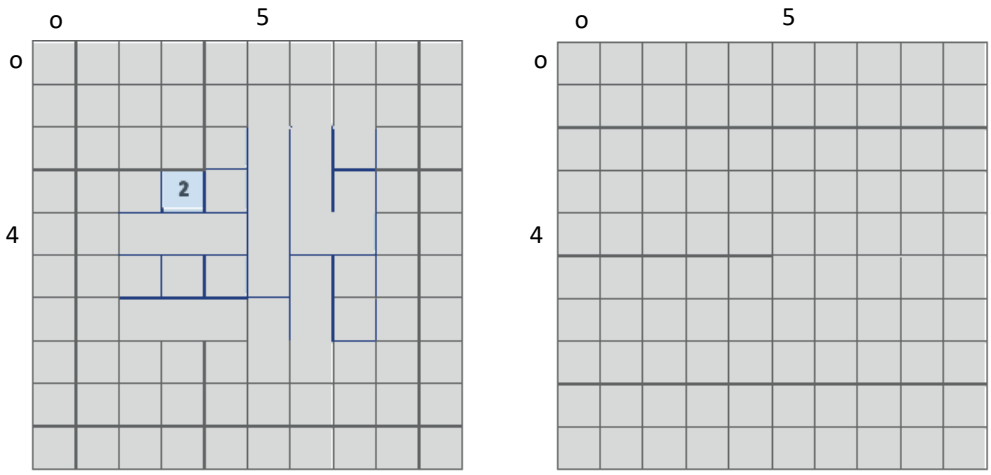


Figura 35. Cálculo de la erosión. Izquierda, imagen de entrada. Derecha, resultado de erosión en la posición (5,4).

En la Figura 35 se ilustra el proceso del cálculo de la erosión en un punto. El EE se va desplazando sobre la imagen original y calculamos el mínimo de los píxeles que caen bajo el EE. El resultado se coloca en la imagen de salida en la misma posición que ocupa el centro.

Para entender los efectos de la erosión consideremos una imagen binaria como la que se muestra en la Figura 36. En dicha imagen podemos distinguir tres casos:

- El EE está enteramente sobre una zona oscura. El mínimo de los píxeles de la zona oscura será el valor de cualquiera de ellos. Por tanto, el píxel de salida no cambiará tras la erosión.
- El EE está enteramente sobre una zona clara. El mínimo de los píxeles será en este caso un valor grande y, por tanto, de nuevo el píxel de salida no cambiará con la erosión.
- El EE está centrado en un píxel claro pero «toca» píxeles oscuros. En ese caso, al tomar el mínimo, este resulta ser un valor oscuro, y por tanto el píxel de salida pasará **de claro a oscuro.**

Nótese que los píxeles claros que se convierten en oscuros son aquellos que están próximos a píxeles oscuros. De alguna manera es como si erosionáramos los píxeles claros del borde, de ahí el nombre del operador. La cantidad de píxeles que pasan de claro a oscuro dependerá del tamaño del elemento estructurante y de su forma:

- Si el EE fuera un segmento vertical, se erosionarían puntos de la parte superior e inferior.
- Si el EE fuera un segmento horizontal, se erosionarían puntos de la derecha y la izquierda.
- La cantidad de píxeles que se erosionan depende del tamaño del EE. Suponiendo un EE vertical de tamaño 5:
    - Se eliminan 2 píxeles blancos de la parte inferior de cada mancha blanca.

- Se eliminan 2 píxeles blancos de la parte superior de cada mancha blanca.
- Avanzado: si el centro del EE estuviera desplazado, es posible eliminar una cantidad distinta por arriba y por abajo.

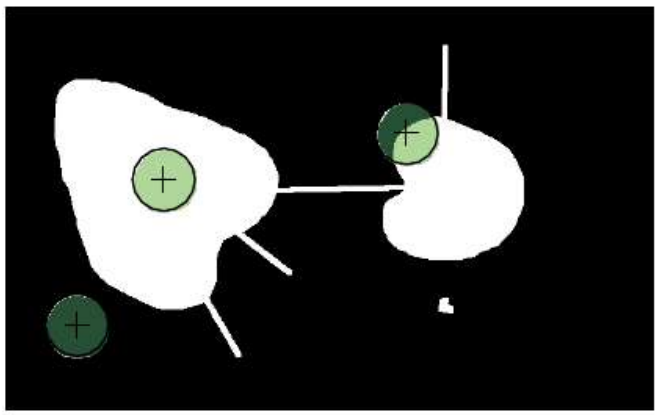


Figura 36. Distintas posiciones del EE durante la erosión de una imagen binaria.

Nótese que los únicos píxeles que permanecen en blanco son aquellos suficientemente alejados del borde del objeto, tal y como muestra la Figura 37**.**

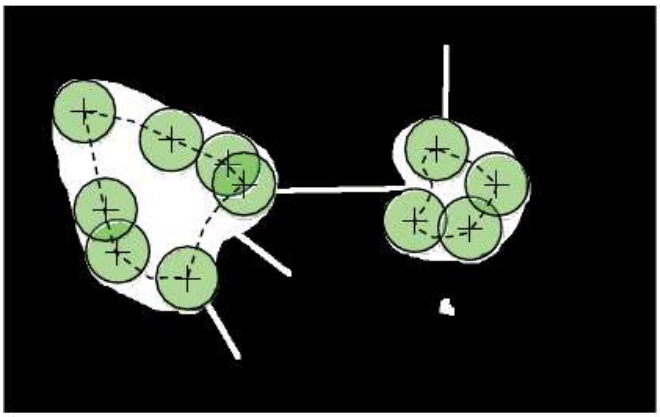


Figura 37. Puntos que permanecen blancos tras una erosión.
Los que están fuera de la línea punteada pasarán a ser negros.

Cuando en vez de tener imágenes tan simples tenemos imágenes normales, el efecto es similar pero cambiando el término blanco por color más claro y el término negro por color más oscuro.

Figura 38. Erosión de imágenes de grises.

En la Figura 38 podemos apreciar el resultado de aplicar una erosión con un EE cuadrado en distintas zonas de la imagen:

- La parte oscura del disco de la señal y los números invaden el fondo blanco de la misma.
- Las zonas que son aproximadamente planas (constantes), como el coche blanco o el asfalto, permanecen casi invariables.
- Los detalles claros pequeños (similar a la isla del ejemplo binario), como las pegatinas del contenedor, se han eliminado.
- Las líneas blancas anchas de la calzada se reducen.

2.2.2.2. Dilatación

La dilatación se considera el operador dual de la erosión. Se obtiene tomando el máximo de los píxeles bajo cada posición del EE como se muestra en la Figura 39. Cálculo de la dilatación en un punto.

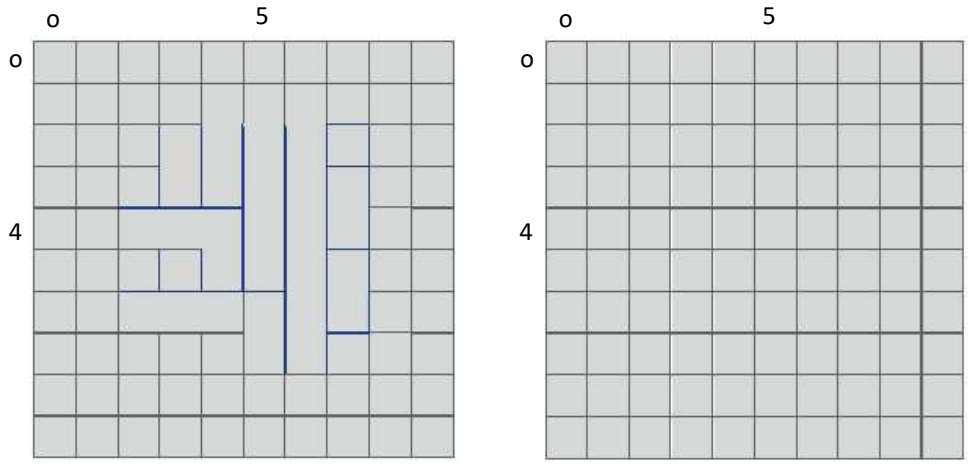


Figura 39. Cálculo de la dilatación en un punto.

El efecto que produce sobre una imagen es justamente el contrario del de la erosión, es decir, la dilatación expande las zonas claras de la imagen hacia las zonas oscuras.

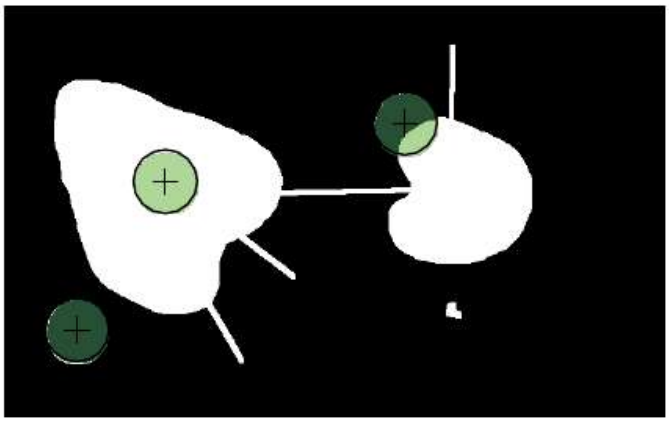


Figura 40. Tres casos para la dilatación.

Al igual que sucedía con la erosión, podemos examinar qué sucede al aplicar la dilatación a la misma imagen binaria de antes. Podemos distinguir tres casos:

- El EE está enteramente sobre una zona oscura. El máximo de los píxeles de la zona oscura será el valor de cualquiera de ellos. Por tanto, el píxel de salida no cambiará tras la dilatación.
- El EE está enteramente sobre una zona clara. El máximo de los píxeles será en este caso un valor elevado y, por tanto, de nuevo el píxel de salida no cambiará con la dilatación.
- El EE está centrado en un píxel oscuro pero «toca» píxeles claros. En ese caso, al tomar el máximo, este resulta ser un valor claro y por ello el píxel de salida pasará **de oscuro a claro.**

Nótese que los píxeles oscuros que se convierten en claros son aquellos que están próximos a píxeles claros. De alguna manera es como si dilatáramos los píxeles claros del borde, de ahí el nombre del operador.

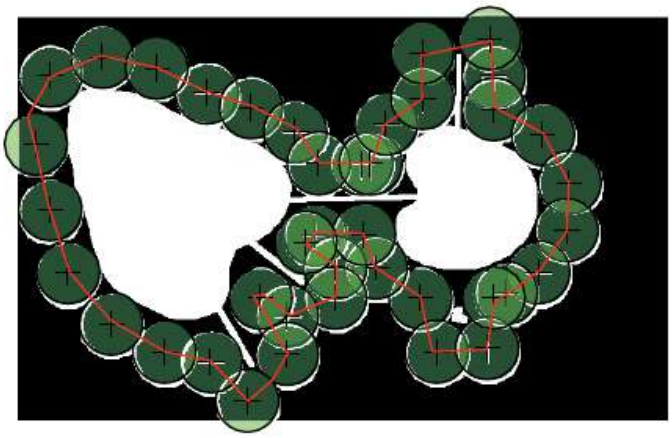


Figura 41. Dilatación. Todos los puntos dentro de la línea roja serán blancos en el resultado de la dilatación.

En la Figura 41 se muestra el efecto de la dilatación sobre una imagen binaria. Solo los puntos en los que el EE esté enteramente sobre píxeles negros permane-

cerán negros (el exterior de la línea roja). Todos los puntos en los que el EE tenga debajo algún punto blanco se convertirán en blanco (interior de la línea roja).

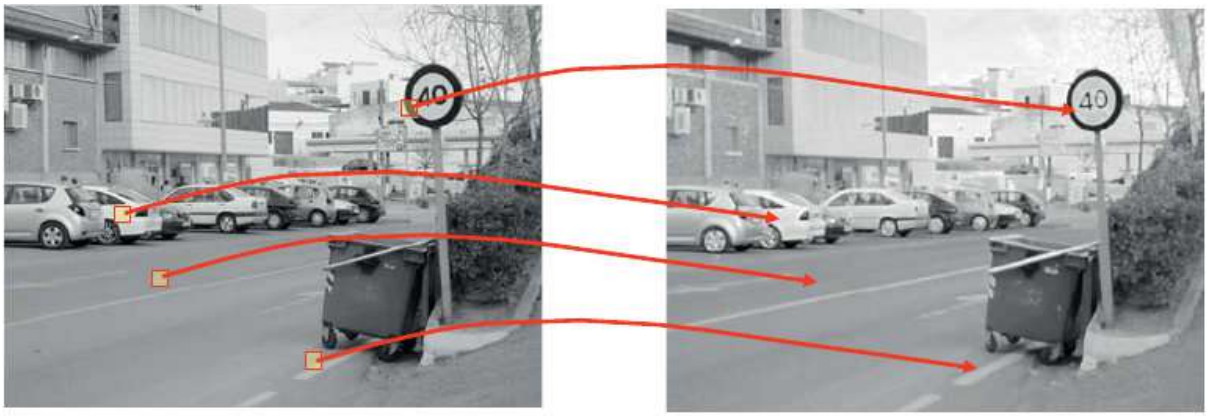


Figura 42. Dilatación de una imagen monocromática.

En el caso de que la imagen no sea tan simple como una imagen binaria, sino que sea una imagen monocromática como la de la Figura 42, podemos observar el efecto que produce la dilatación con un EE cuadrado del tamaño mostrado en la figura prestando atención a distintas zonas:

- Cuando estamos en una zona plana, como por ejemplo el coche blanco o el asfalto gris, el resultado de la dilatación prácticamente coincide con la imagen original.
- Cuando estamos en una zona oscura próxima a una zona clara, como es el borde de la señal, vemos cómo la zona clara «avanza» hacia la zona oscura y la reduce de tamaño.
- Lo mismo sucede, por ejemplo, con los números de la señal. Se han "adelgazado" (oscuros) en favor del fondo (claro). Si el tamaño del EE hubiera sido ligeramente más elevado, podría haber ocurrido que los objetos oscuros (números) se hubieran eliminado totalmente.
- Los objetos claros (línea sobre el asfalto) se hacen más gruesos.

2.2.2.3. Residuos: gradientes morfológicos

En los apartados anteriores hemos visto cómo la dilatación propaga valores claros hacia las zonas más oscuras adyacentes, mientras que la erosión hace lo mismo pero con los valores oscuros hacia zonas más claras.

Recordemos también que los píxeles pueden:

- Permanecer casi constantes si están en una zona plana.
- Permanecer constantes si son más claros que sus vecinos y estamos haciendo una dilatación.
- Permanecer constantes si son más oscuros que sus vecinos y estamos haciendo una erosión.
- Oscurecerse y tomar el valor del mínimo bajo el EE si realizamos una erosión y cerca del píxel existe uno más oscuro.

- Aclararse y tomar el valor del máximo bajo el EE si realizamos una dilatación y cerca del píxel existe uno más claro.

Si realizásemos la diferencia entre la imagen original y el resultado de la erosión/dilatación, los píxeles que no cambian, los que corresponden a los tres primeros casos de la lista anterior, tendrían un valor de la diferencia nulo o muy pequeño. Sin embargo, cuando estemos en una situación como la de los últimos dos puntos, resultaría que al hacer la resta tendríamos un valor elevado.

Nótese también que al hacer la resta, es posible saber su signo:

- La diferencia dilatación - original será siempre positiva, ya que cualquier píxel del resultado de la dilatación es mayor o igual que el mismo píxel de la imagen original.
- La diferencia original - erosión será siempre positiva, ya que cualquier píxel de la erosión será siempre menor que el mismo píxel en la imagen de entrada.

A este tipo de diferencias se las denomina **residuos**. Cuando el EE que se utiliza es un pequeño cuadrado de (3 x 3, 5 x 5) lo que se obtiene con los residuos son líneas claras finas en las transiciones claro-oscuro, de ahí el nombre de gradiente.

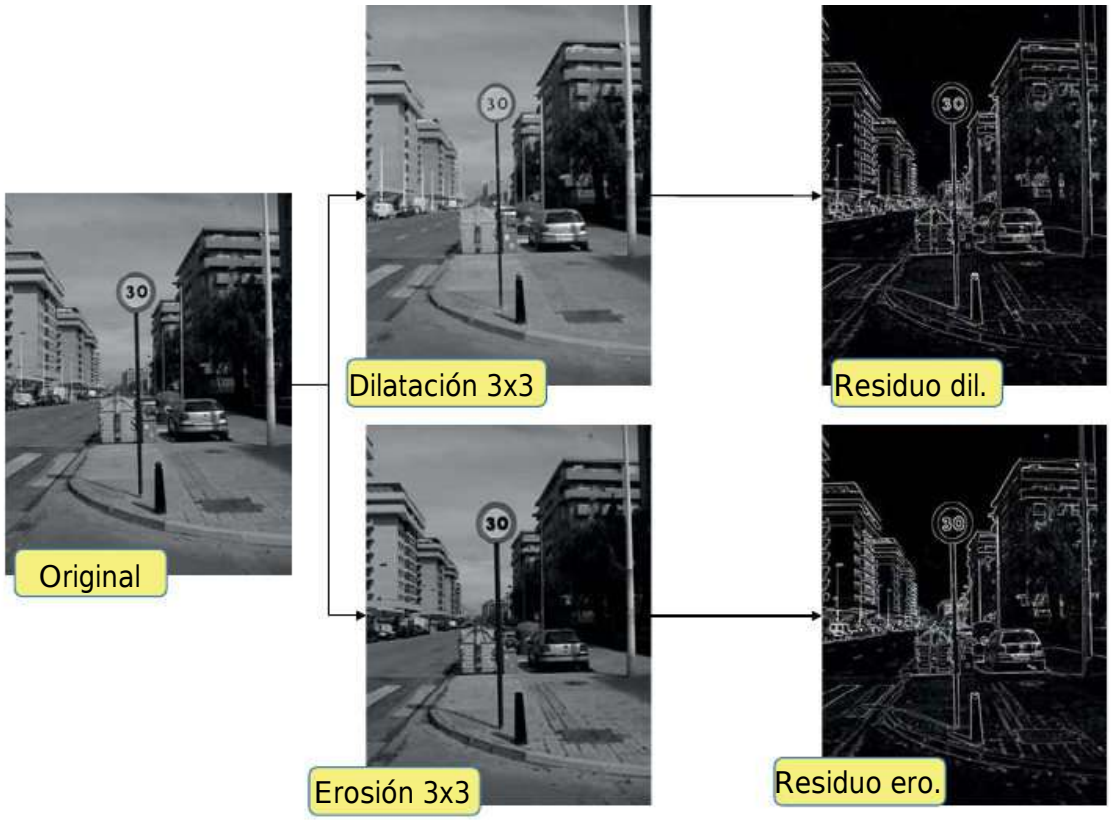


Figura 43. Gradientes morfológicos por erosión y dilatación.

La Figura 43 ilustra el proceso de cálculo del gradiente morfológico. Vemos cómo la dilatación expande las zonas claras (los números aparecen más delgados, así como el borde de la señal) mientras la erosión hace los números y el bor-

de de la señal más gruesos. Al calcular el residuo observamos una serie de líneas claras justo en las transiciones. El nivel de gris de estas líneas claras se corresponde con la diferencia de intensidad en la imagen original. Si la imagen estuviera perfectamente enfocada, la anchura de dichas líneas sería de un píxel. Aunque aparentemente tanto el gradiente por dilatación como por erosión son similares, si nos fijamos en la Figura 44 podemos observar que en el caso del gradiente por dilatación, las líneas blancas se sitúan sobre las zonas oscuras de la transición claro-oscuro, mientras que en el caso del gradiente por erosión, las líneas claras se sitúan sobre la parte clara de la transición claro-oscuro.

Para facilitar el análisis posterior, interesará que cada línea blanca no toque a las otras, es decir, que por ejemplo para pasar del contorno de un número al de otro haya necesariamente en medio píxeles negros. En un caso como el de la imagen que se muestra en las figuras, dará lo mismo cuál de los dos gradientes empleemos. Sin embargo, cuando el tamaño de las imágenes sea pequeño (porque la señal esté lejos), los contornos aparecerán más próximos entre sí. Si nos fijamos de nuevo en la Figura 44, nos daremos cuenta de que el residuo de la dilatación nos mantiene más separados los contornos. Por ese motivo será el que preferiremos.

Dilatación - Original

Original - Erosión

Figura 44. Detalle para comparar los dos tipos de gradientes morfológicos.

2.2.3. Operadores morfológicos en OpenCV

Las funciones de erosión y dilatación las tenemos implementadas en OpenCV (`dilate()` y `erode()`). Las funciones requieren como parámetros de entrada:

- La imagen a filtrar y
- El elemento estructurante,

y devuelven como salida el resultado de la erosión/dilatación. Para crear fácilmente el EE existe una función llamada `getStructuringElement()`. Se recomienda consultar la documentación para ver las posibilidades de estas funciones.

```
double tam = 3;
```

```
Mat SE = imgproc.getStructuringElement(imgproc.MORPH_RECT, new
Size(tam,tam));
Mat gray_dilation = new Mat(); // Result
Mat gray_erosion = new Mat(); // Result
imgproc.dilate(gray, gray_dilation, SE ); // 3x3 dilation
imgproc.erode(gray, gray_erosion, SE ); // 3x3 erosion
```

Para calcular los residuos, simplemente debemos restar a la imagen original el resultado de la dilatación/erosión teniendo en cuenta el signo de la resta:

```
Mat dilation_residue = new Mat();
Mat erosion_residue = new Mat();
Core.subtract(gray_dilation, gray, dilation_residue);
Core.subtract(gray, gray_erosion, erosion_residue);
```

2.2.3.1. Consideraciones sobre coste computacional

Suponiendo que el EE fuera un cuadrado de 9 x 9, podría pensarse que el coste computacional de calcular la erosión en un punto de la imagen sería de 81 comparaciones para buscar el mínimo de los 81 píxeles que recaen bajo el EE en cada posición. Sin embargo, al igual que sucedía con los filtros lineales, una erosión de 9 x 9 se puede calcular mediante separación como una erosión con un EE vertical de tamaño 9 seguida de otra horizontal del mismo tamaño. En ese caso pasaríamos de 81 comparaciones a 2 x 9 = 18 comparaciones.

Adicionalmente los filtros morfológicos se pueden implementar de forma recursiva de modo que, independientemente del tamaño del EE, una erosión/dilatación con un EE lineal (horizontal o vertical) puede implementarse con una media de tres comparaciones por píxel.

Por todo ello, este tipo de operadores se puede considerar como eficientes computacionalmente.

Ejercicio: Detección de contornos con gradientes morfológicos

En este ejercicio implementaremos un filtrado de gradiente por dilatación para detectar los bordes (transiciones rápidas) en una imagen monocromática. La imagen que se visualizará será similar a la de la Figura 43.

1. Haz una copia del programa Base5mono.
2. Sobre dicha copia, modifica la función `procesa()` para que se realice el filtrado pedido. En este ejercicio deberás ser tú el que escriba el código. Todas las instrucciones necesarias se han visto anteriormente en los ejemplos, revísalos. Los pasos que debes seguir son:
   a. Calcular la dilatación con un EE cuadrado de tamaño 3 x 3.
   b. Calcular el residuo de la dilatación prestando atención al orden en que se realiza la resta.

3. Prueba el programa realizado sobre el dispositivo:
   - Sobre una imagen de una señal impresa, debes observar el contorno circular del borde de la señal, así como el contorno de los números.
   - Apuntando el teléfono en cualquier dirección, debes observar una imagen con fondo negro y líneas en blanco en los contornos.
4. **OPCIONAL:** si colocas la imagen de la señal suficientemente cerca de la cámara, en la zona del número 0, se podrán observar dos líneas claras próximas (tal y como se ven en la Figura 45). Esto puede complicar el reconocimiento del dígito. Para evitar que suceda esto, se puede modificar el programa que acabas de realizar de modo que ambas líneas se unan dando lugar a un trazo grueso. Para ello es suficiente con que la dilatación se realice con un EE cuyo tamaño sea superior al ancho del trazo del carácter.
   - Modifica el programa anterior cambiando el tamaño del EE a un valor de 11 x 11.
   - Vuelve a ejecutar el programa sobre la imagen de la señal. Observarás que ahora las líneas blancas son más gruesas y que en los números ya no es posible distinguir dos contornos en el cero sino un trazo grueso

Figura 45. Efecto del cambio del tamaño de la dilatación en el residuo.

2.2.3.2. Conclusión sobre operadores morfológicos

Los filtros morfológicos constituyen una alternativa a los filtros lineales para muchas aplicaciones. La principal diferencia que presentan respecto a los filtros lineales es que actúan de manera selectiva sobre las partes claras u oscuras de la imagen.

Para el cálculo de gradientes constituyen una alternativa de menor coste computacional que las clásicas como el filtro de Sobel. Adicionalmente, es posible controlar la posición de las líneas de gradiente con precisión (situarlas sobre la parte oscura o clara de la transición).

Preguntas de repaso: Morfología

CAPÍTULO 3

Visión artificial: Segmentación y reconocimiento

Por ANTONIO ALBIOL

En este capítulo analizaremos las herramientas necesarias para localizar las señales de tráfico en una escena y cómo leer de manera automática los números en las señales. Para ello será necesario realizar una segmentación de la imagen, que no es otra cosa que dividirla en regiones y analizarlas en búsqueda de aquellas que cumplan ciertos criterios como la circularidad, la presencia del color rojo en el borde, etc. Finalmente analizaremos un método sencillo pero efectivo para reconocer los dígitos en las señales.

Objetivos:

- Describir qué es la binarización y conocer las principales alternativas para elegir un umbral.
- Aprender a utilizar las técnicas de segmentación disponibles en OpenCV.
- Conocer cómo podemos extraer características de un objeto que nos ayuden a identificarlo.
- Describir las etapas usadas en el proceso de reconocimiento.
- Aplicar estos conocimientos para implementar un OCR que reconozca dígitos.

3.1. Binarización

Para poder detectar correctamente las señales son necesarias dos fases:

- Localizar las zonas de la imagen que pueden contener la señal.
- Analizar esas zonas para leer el número que contienen.

Hasta el momento, lo que hemos visto en apartados anteriores eran operaciones que, dada una imagen, generaban otra imagen. Esta fase es lo que se denomina preproceso, y consiste en generar imágenes donde resulte más sencillo observar aquello que buscamos. Las figuras siguientes ilustran ejemplos en los que se busca enfatizar lo rojo o las transiciones.

Figura 46. Ejemplo de preproceso para enfatizar zonas rojas

Figura 47. Ejemplo de preproceso para enfatizar transiciones.

Ambos casos tienen en común el hecho de que partimos de una imagen, tal y como la capta la cámara, y obtenemos una imagen donde:

- es oscuro lo que no me interesa,
- es más claro lo que me interesa.

El siguiente paso a dar consiste en realizar una binarización. La binarización consiste simplemente en aplicar un umbral, y poner a 1 los píxeles que lo superen y a 0 los que no lo hagan. La binarización ya se ha comentado en apartados anteriores. La idea de la binarización es esencialmente la de tomar una decisión 0/1 sobre si un punto es de una determinada manera o no (rojo/no-rojo). Si bien conceptualmente es muy sencilla, tiene una pequeña dificultad práctica, que es:

¿Cómo elegir el umbral?

Para un mismo resultado del preproceso, en función del umbral que apliquemos, obtendremos diferentes imágenes binarias. En la Figura 48 se ilustran los diferentes resultados que se obtienen con distintos umbrales. Normalmente existirá un margen de posibles umbrales que nos servirán, es decir, la elección del umbral es importante, pero tenemos un cierto margen para elegir el valor. En otras palabras, no es vital acertar con el valor exacto. Dicho lo anterior, también es cierto que un umbral mal elegido hace imposible continuar el proceso de detección. En la Figura 48 observamos distinto número de manchas blancas según el umbral elegido. Ahora bien, en ambos casos podemos ver un círculo negro. ¿Cómo tendría que haber sido el umbral para que no hubiera sido válido? Si el umbral hubiera sido demasiado pequeño, el interior del círculo negro habría contenido puntos blancos. Si hubiera sido demasiado alto, la frontera blanca hubiera podido partirse o incluso desaparecer. Obsérvese que en la medida en que el preproceso proporcione grandes diferencias entre el interior del círculo y su frontera, la elección del umbral será más robusta.

En este apartado veremos diferentes métodos para elegir el umbral:

- umbral fijo,
- umbral dependiente de la estadística de la imagen,
- umbral por Otsu,
- umbral adaptativo.

A continuación, explicaremos brevemente en qué consiste cada uno de estos métodos.

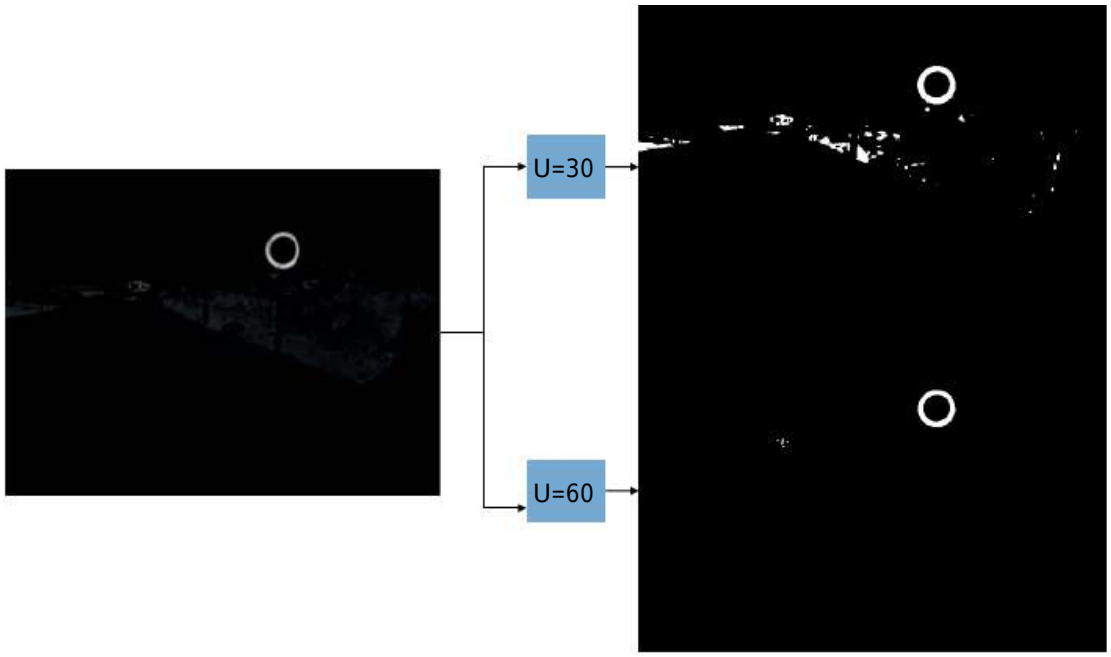


Figura 48. Distintos resultados binarios al aplicar distintos umbrales sobre la misma imagen.

3.1.1. Umbralización fija

Es la más sencilla. Simplemente elegimos un valor de umbral, en base a nuestra experiencia previa, o por prueba y error, y lo aplicamos. Puede tener el problema de que, dependiendo de la iluminación, un umbral fijo a veces proporcione buenos resultados y otras no. Las soluciones para esto son:

- Probar varios umbrales fijos y mirar si para cada uno de ellos se detectan señales. Tiene el inconveniente de que todos los pasos que sigan a la binarización se repetirán tantas veces como umbrales empleemos.
- Realizar alguna transformación de intensidad que normalice la imagen como el aumento lineal de contraste o la ecualización de histograma.

3.1.2. Umbralización dependiente de la imagen

Una alternativa un poco más inteligente que la anterior consistiría en analizar los píxeles de la imagen y elegir el umbral en función de dicho análisis. El tipo de análisis que se suele emplear es:

- Determinar el máximo (M) del resultado del preproceso: simplemente se busca la mayor intensidad.
- Determinar el máximo en sentido estadístico del resultado del pre-proceso: es lo que hicimos con el aumento lineal del contraste. Se busca aquel valor que solo se rebasa en un cierto porcentaje de los píxeles. En general proporciona mejores resultados que el máximo absoluto, ya que no depende de un solo píxel. Se determina con el histograma.
- Determinar la media (m): como en el resultado del preproceso la mayor parte de los píxeles serán oscuros, la media nos da una idea de cómo es la amplitud de esos píxeles oscuros.

Una vez determinado el máximo (M) o la media (m), se suele introducir un margen de seguridad. Así, por ejemplo, elecciones razonables del umbral serían:

- U = M/4 (donde 4 se puede sustituir por otro valor similar).
- U = m * 1,8 (donde 1,8 se puede sustituir por otro valor normalmente mayor que uno).
- U = m + 20 (donde 20 es la diferencia que esperamos que tengan las partes claras del pre-proceso respecto al fondo; se puede cambiar por otro valor similar como 10, 30 ...).

La Figura 49 y la Figura 50 muestran cómo eligiendo el umbral en función del máximo se puede obtener un buen umbral en ambos casos.

Figura 49. Dos imágenes con diferentes iluminaciones y sus respectivos resultados del preproceso para detectar zonas rojas.

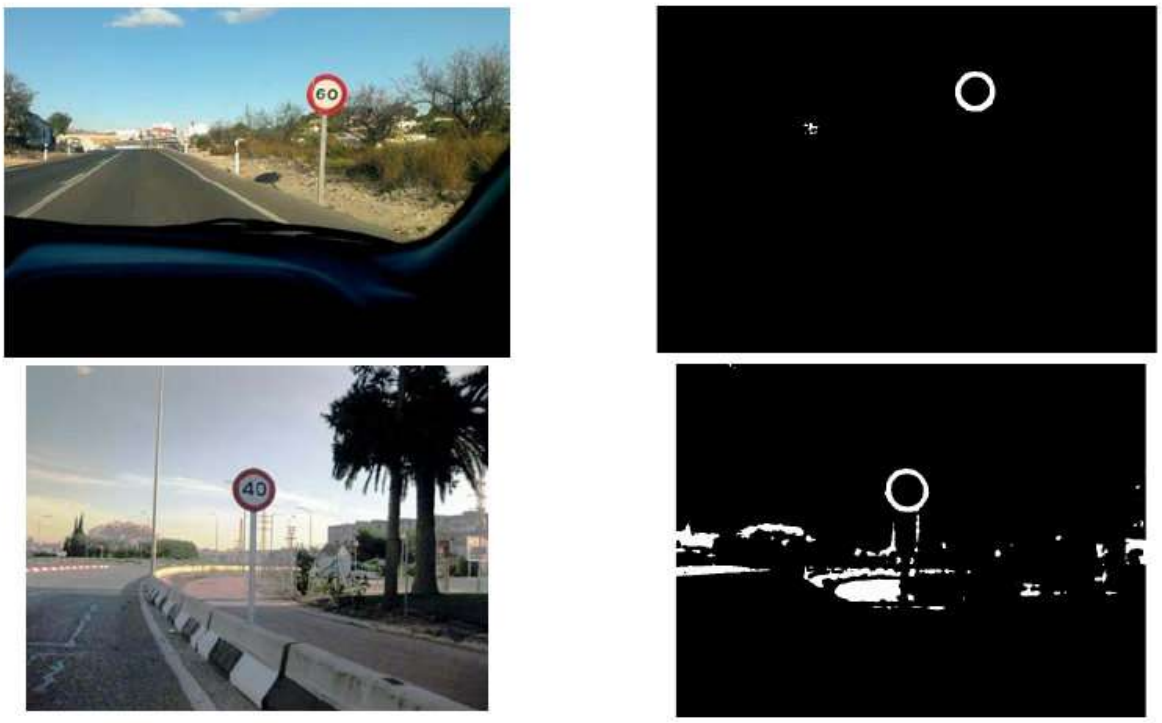


Figura 50. Resultado de la binarización al aplicar en ambos casos un umbral U = M/4.

Ejercicio: Binarización de las Zonas Rojas.

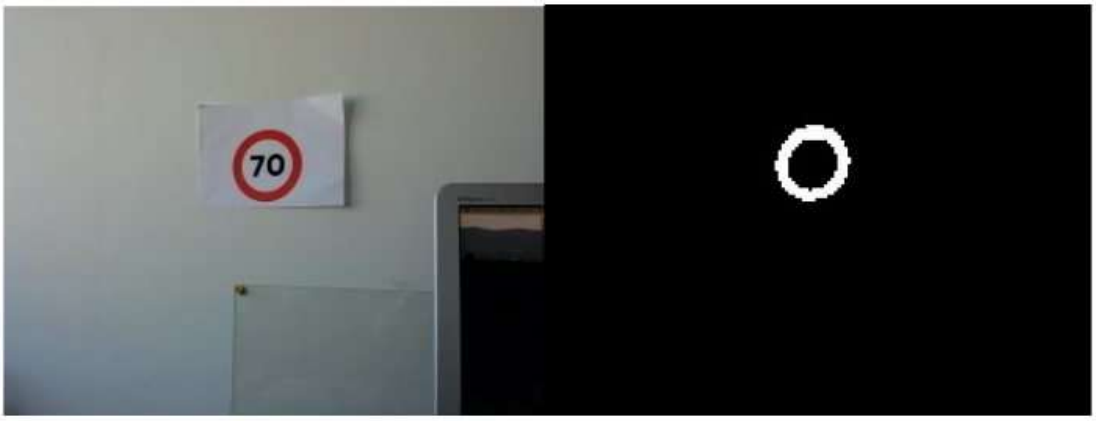
En este ejercicio partiremos del programa que detectaba objetos rojos y binarizaremos el resultado usando el máximo de la imagen para obtener el umbral.

1. Haz una copia del programa zona roja.
2. Sobre dicha copia modifica la función `procesa()`para que se realice la binarización pedida. Las líneas a añadir se incluyen en el siguiente bloque. El umbral lo determinaremos como el máximo dividido por 4.

```
Core.MinMaxlocResult minMax = Core.minMaxloc(salida);
int maximum = (int) minMax.maxVal;
int thresh = maximum / 4;
Imgproc.threshold(salida, salida, thresh, 255, Imgproc.THRESH_BINARY);
```

3. Prueba el programa realizado sobre el dispositivo. Deberás observar algo similar a lo que se muestra en la figura.

Para buscar el máximo hemos usado la función `minMaxloc()` que determina el máximo y el mínimo. Dicha función devuelve el resultado en una clase llamada `MinMaxlocResult` que contiene ambos valores.



3.1.3. Umbralización por Otsu

El método de Otsu es útil cuando la imagen a binarizar es aproximadamente bimodal. Una imagen bimodal es aquella en la que predominan dos valores de intensidades. El caso típico de esta situación en nuestra aplicación sería el interior de la señal, donde tenemos píxeles claros del fondo y píxeles oscuros de los números. La Figura 51 ilustra una imagen bimodal real y su histograma. La idea del método de Otsu consiste en proporcionar un umbral que esté aproximadamente en el centro de los dos picos del histograma. Para ello, calcula una cantidad llamada separabilidad que mide, para cada posible umbral, cómo se separaría lo claro de lo oscuro. El umbral se elige como aquel valor que maximiza la separabilidad. Para los detalles de este método puede se puede consultar:

http://en.wikipedia.org/wiki/Otsu's_method

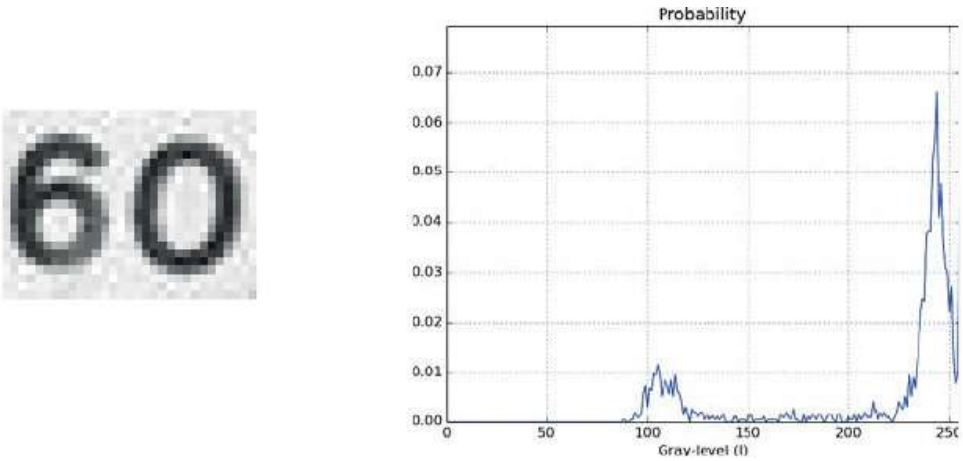


Figura 51. Imagen bimodal y su histograma.

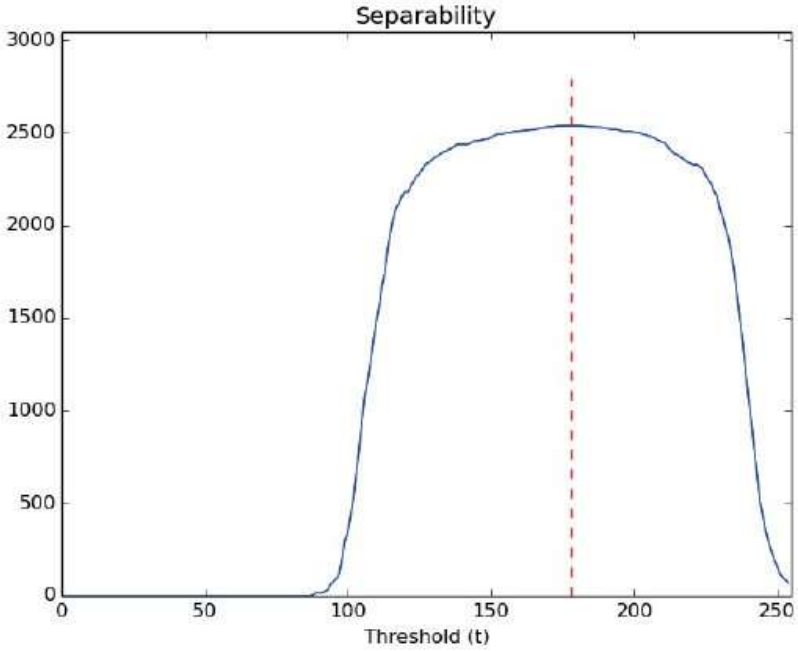


Figura 52. Separabilidad en el método de Otsu.

Ejercicio: Binarización usando Otsu.

En este ejercicio partiremos del programa base monocromático y le añadiremos una umbralización usando Otsu. Para probarlo, necesitaremos una imagen bimodal. Lo que haremos será apuntar la cámara a una hoja blanca que contenga un número negro.

1. Haz una copia del programa Base5mono.

2. Sobre dicha copia modifica la función `procesa()`para que se realice la binarización pedida. Las líneas a añadir se incluyen en el siguiente bloque:

```
Imgproc.threshold(entrada, salida, 0, 255,Imgproc.THRESH_OTSU 1
        Imgproc.THRESH_BINARY);
```

3. Prueba el programa realizado sobre el dispositivo intentando enfocar la cámara a una imagen bimodal. Deberás observar algo similar a lo que se muestra en la figura. Cuando en el tipo de umbralización le indicamos que queremos emplear Otsu, el valor del umbral proporcionado, 0 en este caso, es ignorado.

Figura 53. Imagen de entrada bimodal y resultado de la umbralización automática con Otsu.

3.1.4. Umbralización adaptativa

La umbralización adaptativa consiste en aplicar un umbral diferente en cada zona de la imagen. La idea es similar a la vista en el apartado 5.1.2, donde elegíamos el umbral en función de la media. La diferencia es que en vez de emplear la media global de la imagen empleamos la media local alrededor de cada punto. Recordemos ahora que los filtros suavizadores calculan una media local alrededor de cada punto. Este método tiene dos parámetros:

- El tamaño de la zona en la que se calcula la media local.
- El margen de seguridad, ε.

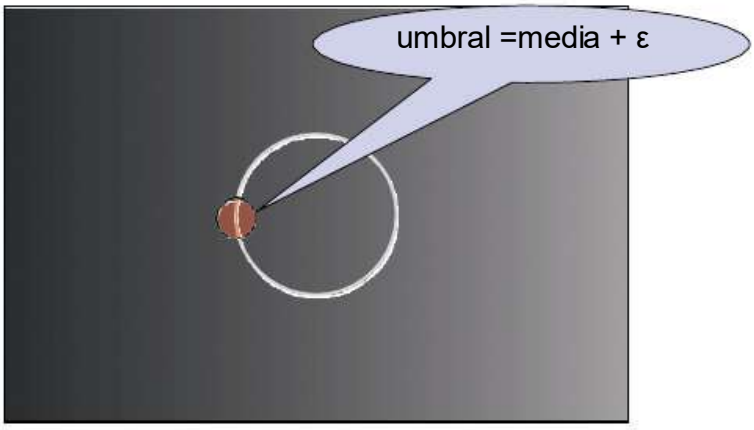


Figura 54. Principios de la umbralización adaptativa.

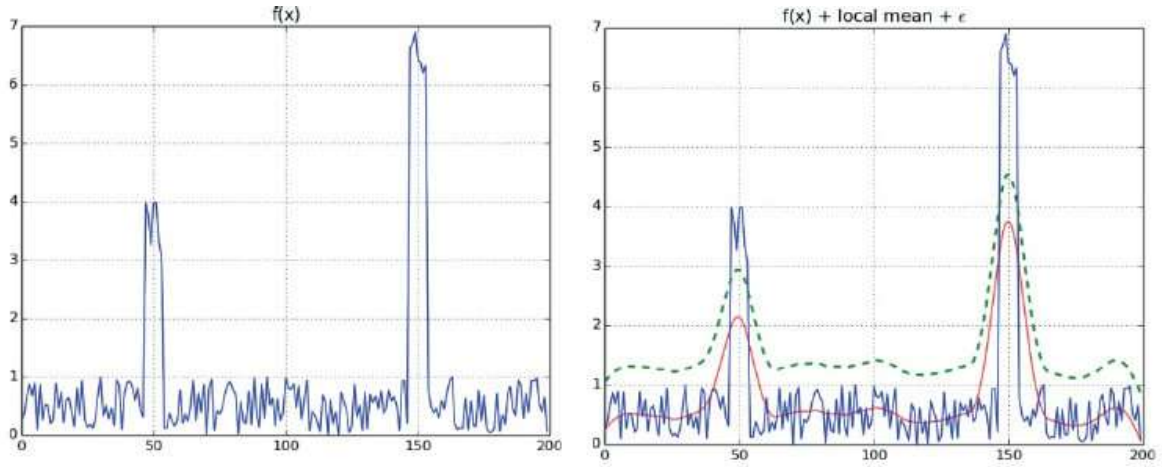


Figura 55. Ejemplo unidimensional del funcionamiento de la umbralización adaptativa. Izquierda: señal a binarizar. Derecha: media local (rojo), umbral (verde). Los puntos donde la curva azul supera a la verde serán los que superen el umbral.

3.1.5. Umbralización en OpenCV

Existen dos funciones principales para realizar la umbralización en OpenCV:

- `threshold()`: realiza la umbralización usando un umbral fijo o mediante el método de Otsu. Es interesante consultar la documentación para estudiar todas las opciones que tiene.
- `adaptiveThreshold()`: realiza la umbralización adaptativa. Véase la documentación para conocer los argumentos de esta función.

Ejercicio: Binarización adaptativa del gradiente morfológico.

En este ejercicio partiremos del gradiente morfológico que se ha implementado y realizaremos una binarización adaptativa

1. Estudia en la documentación el significado de los argumentos de la función adaptiveThreshold().
2. Haz una copia del programa gradiente morfológico.
3. Sobre dicha copia modifica la función `procesa()` para que se realice la binarización pedida. Las líneas a añadir se incluyen en el siguiente bloque.

```
//Calculo del gradiente morfológico.
int contraste = 2;
int tamano = 7;
Imgproc.adaptiveThreshold(grad, binaria,255,
        Imgproc.ADAPTIVE_THRESH_MEAN_C,
        Imgproc.THRESH_BINARY,
        tamano, -contraste );
```

4. Prueba el programa realizado sobre el dispositivo. Deberás observar líneas blancas en los contornos de los objetos y nada en el resto. La línea del círculo interior de la señal deberá aparecer entera, no partida en fragmentos.
5. Experimenta con valores de contraste y tamaño de bloque diferentes.

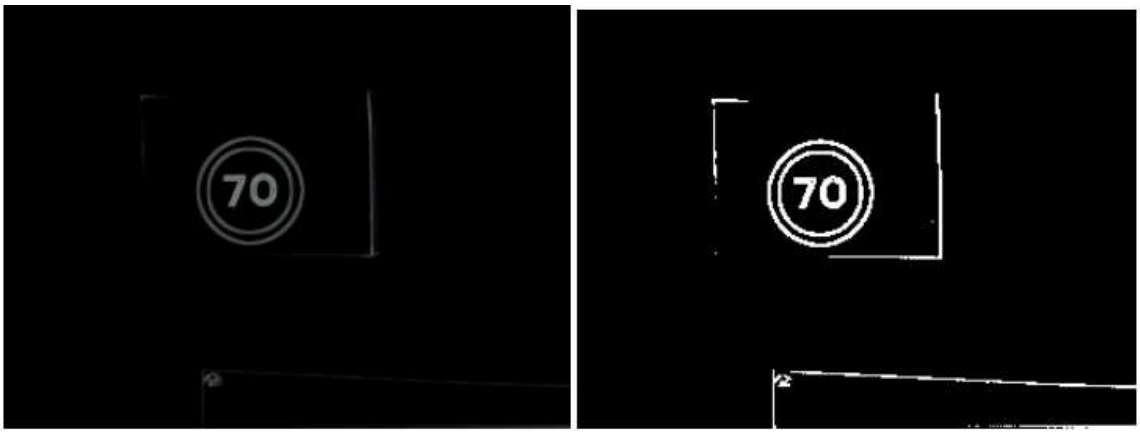


Figura 56. Gradiente morfológico y resultado de la binarización adaptativa.

Preguntas de repaso: Binarización

3.2. Segmentación

Como resultado de la binarización habremos obtenido una imagen binaria similar a la que se muestra en la Figura 57. En dicha imagen habremos puesto los píxeles que nos interesan a 1 y los que no a 0. No obstante, dicha imagen no es más que un conjunto de píxeles. Para seguir avanzando es necesario agrupar los píxeles en entidades de nivel superior que denominaremos **objetos**. El proceso por el que dividimos la imagen en objetos se denomina segmentación.

Definiremos los objetos usando la propiedad llamada conectividad. Los grupos de píxeles blancos que se toquen (formen una componente conexa) pertenecerán al mismo objeto, mientras que cuando pertenezcan a islas diferentes diremos que son objetos diferentes. Los píxeles negros de la imagen binaria formarán lo que denominaremos fondo, es decir, zonas que no nos interesan.

Por lo tanto, el proceso de segmentación tomará como entrada una imagen binaria y producirá como resultado una serie de objetos. Cada uno de los objetos estará formado por un conjunto de píxeles blancos que se tocan entre sí.

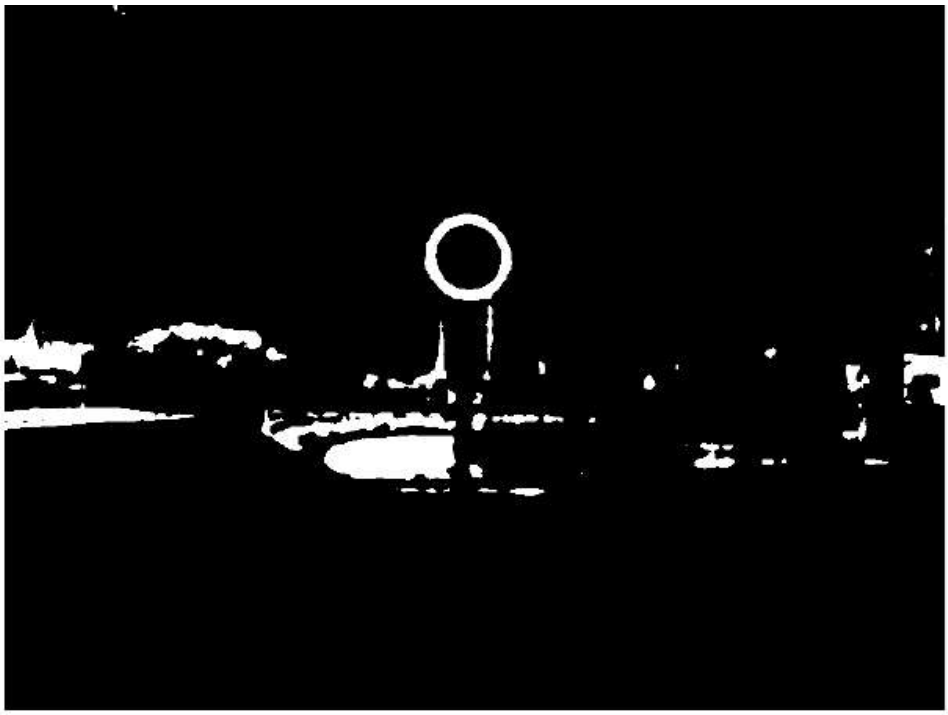


Figura 57. Resultado típico de la binarización.

Hay dos maneras clásicas de representar el resultado de la segmentación:

- Como una matriz del mismo tamaño que la imagen, pero donde los valores de los píxeles son enteros que representan el número de objeto (enteros positivos correlativos a partir de uno). A esta matriz se le suele llamar capa de segmentación o capa de etiquetas. Se suele visualizar como una imagen donde los píxeles con la misma etiqueta toman un mismo color (Figura 58).
- Para cada uno de los objetos, una lista de coordenadas (x, y) de los píxeles que pertenecen a dicho objeto.

Figura 58. Imagen binaria y el resultado de su segmentación visualizado con colores diferentes para cada etiqueta.

3.2.1. La segmentación en OpenCV

OpenCV utiliza una curiosa forma de tratar la segmentación. Dada una imagen binaria, en vez de etiquetar cada píxel con el número de objeto al que pertenece, lo que hace es etiquetar únicamente los puntos del contorno. La función que realiza esta operación se llama `findContours()` y en los párrafos siguientes vamos a explicar cómo funciona, ya que es muy potente (puede hacer muchas cosas) pero por esa misma razón es fácil no hacer un uso adecuado de la misma.

Figura 59. Segmentación de una imagen binaria por contornos del tipo realizado por OpenCV.

La función recibe como entrada una imagen binaria, como se muestra en la Figura 59 y devuelve una lista de contornos. Cada contorno es una lista de coordenadas (x, y) de los puntos que forman parte del contorno.

```
list<MatOfPoint> contornos = new Arraylist<MatOfPoint> ();
findContours(binaria, contornos, .
```

Hay dos maneras de devolver los contornos. La primera devuelve todos los puntos del contorno. La segunda elimina algunos puntos del mismo cuando los puntos intermedios pertenecen a un segmento recto. Así, por ejemplo, de este segundo modo, un contorno cuadrado se representaría con cuatro puntos. Para nuestra aplicación se recomienda usar el modo que devuelve todos los puntos especificando la opción `Imgproc.CHAIN_APPROX_NONE` como último argumento.

Los objetos que se muestran en la Figura 59 son objetos sencillos. Obsérvese que, a partir de los contornos, y dado que estos son necesariamente cerrados, es posible deducir todos los puntos de cada objeto. Sin embargo, el mundo real puede ser más complicado y los objetos pueden tener agujeros en su interior. Para poder representar adecuadamente los objetos con agujeros es necesario distinguir entre contornos interiores y exteriores:

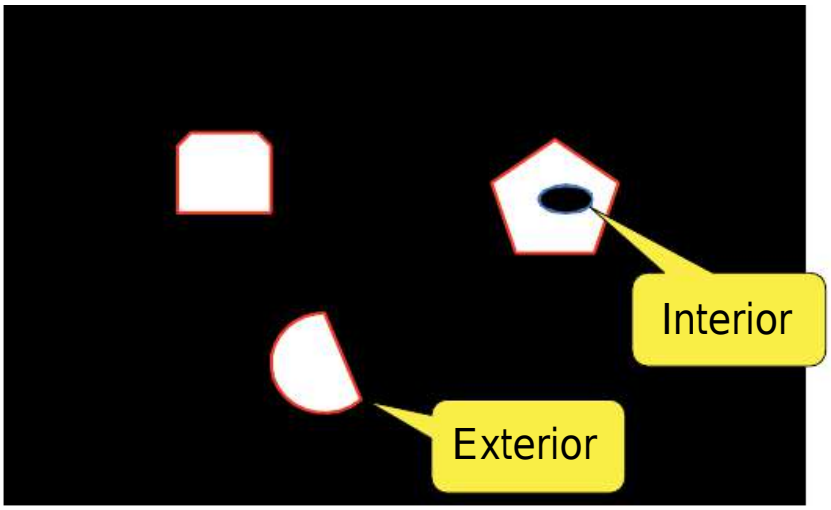


Figura 60. Contornos exteriores e interiores.

OpenCV nos puede devolver todos los contornos de la imagen binaria y además indicarnos si son interiores o exteriores. Si bien la Figura 60 es más complicada que la Figura 59, no es con mucho la situación más complicada que puede darse. En la Figura 61 se puede observar un ejemplo de situación compleja. En dicha figura vemos cómo el pentágono tiene dos agujeros, de los cuales uno de ellos contiene a su vez dos objetos blancos, uno de los cuales incluye también un agujero.

Dependiendo de las opciones que le demos a la función findContours, OpenCV puede devolver la información suficiente como para hacerse una idea de las relaciones entre los diferentes contornos en situaciones como la Figura 61.

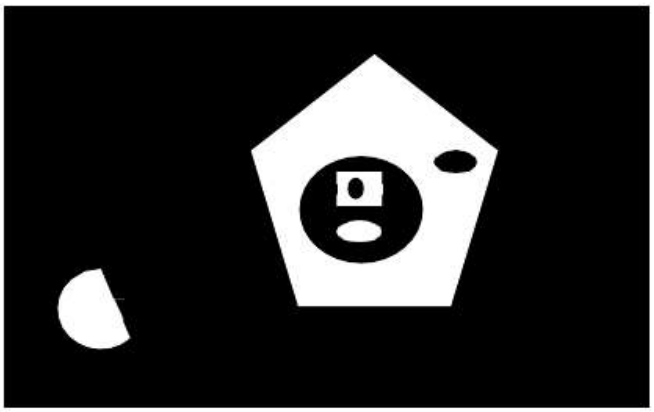


Figura 61. Imagen binaria con una jerarquía de anidamiento de contornos.

Una clasificación de los contornos de la Figura 61 podría ser en contornos exteriores e interiores. La Figura 62 muestra qué considera OpenCV contorno externo e interno. Nótese cómo los contornos exteriores de «islas blancas» en el interior de «lagos negros» se consideran externos.

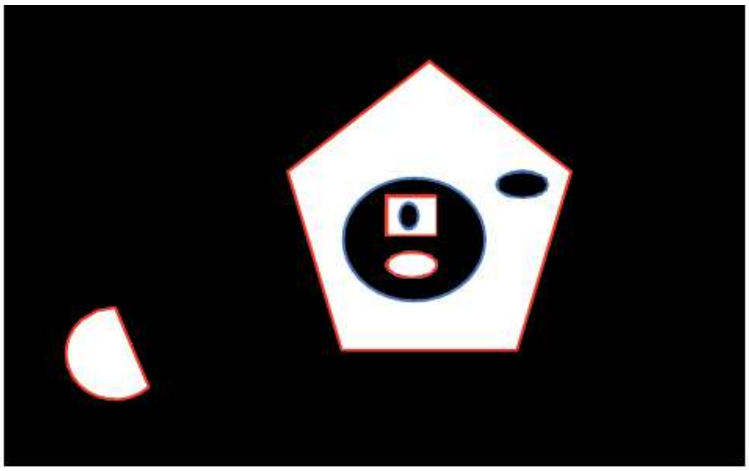


Figura 62. Contornos exteriores (rojo) e interiores (azul) en una situación compleja.

Para describir la situación de anidamiento de los contornos, OpenCV puede proporcionar para cada contorno la siguiente información:

- Número del contorno padre: los objetos más exteriores (pentágono y arco circular) no tienen padre. Los contornos interiores del pentágono tienen como padre el contorno del pentágono. Las islas elíptica y rectangular dentro del lago circular del pentágono tienen como padre el del lago.
- Número del primer contorno hijo.
- Número del contorno hermano anterior.
- Número del contorno hermano posterior.

Cuando no existe alguno de los contornos (por no tener hijos, no tener padre o ser el primer/último contorno hijo) dicho valor contiene -1.

3.2.2. Objetos delgados y objetos con agujeros

Es posible, sobre todo cuando se emplean gradientes, que tengamos objetos delgados (menos de 2 píxeles de espesor). Este tipo de objetos son típicos cuando calculamos gradientes. En la Figura 63 podemos observar dos objetos, un anillo y un rectángulo y en qué se convierten cuando los adelgazamos hasta un píxel. En la parte de la izquierda, cuando los objetos son gruesos, es fácil darse cuenta de que el anillo tiene un contorno exterior y otro interior, mientras que en el rectángulo solo tenemos el contorno exterior. Este hecho podría emplearse para distinguir fácilmente un objeto macizo de otro hueco. Cuando la figura se adelgaza, se sigue manteniendo que la circunferencia tiene dos contornos, uno exterior y otro interior, mientras que la línea solo tiene el exterior. En el caso de la circunferencia, puede ser que los puntos del contorno exterior e interior sean los mismos.

Esta distinción entre contornos exteriores e interiores puede ser útil, por ejemplo, para distinguir entre objetos que originariamente tienen un hueco y los que no lo tienen. Los contornos interiores siempre corresponderán a objetos con

hueco. Un ejemplo de objeto con hueco que resulta interesante en nuestra aplicación es la circunferencia de la señal.

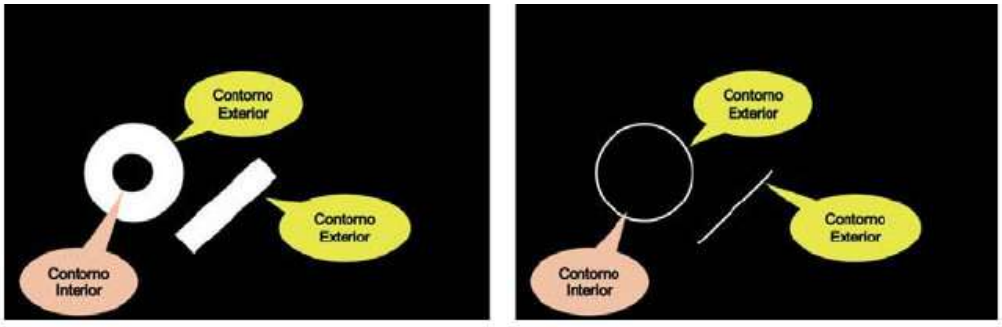


Figura 63. Objetos delgados

3.2.3. Opciones de findContours()

Dicha función, además de la imagen binaria a segmentar y el resultado, requiere un tercer parámetro que tiene que ver con el modo en que va a extraer los contornos:

`ImgProc.RETR_lIST:` devuelve todos los contornos sin distinguir entre exteriores e interiores ni ningún tipo de relación jerárquica.

`ImgProc.RETR_EXTERNAl:` solo devuelve los contornos exteriores.

`ImgProc.RETR_CCOMP:` devuelve los contornos con una jerarquía de dos niveles: exteriores/interiores.

`ImgProc.RETR_TREE:` devuelve la jerarquía completa de contornos del modo descrito más arriba.

Figura 64. Resultado de findContours() con la opción RETR LIST. No se distingue entre exterior e interior. Todos los contornos se devuelven como exteriores.

Figura 65. Resultado de findContours() con la opción RETR EXTERNAL. Solo se devuelven los contornos exteriores.

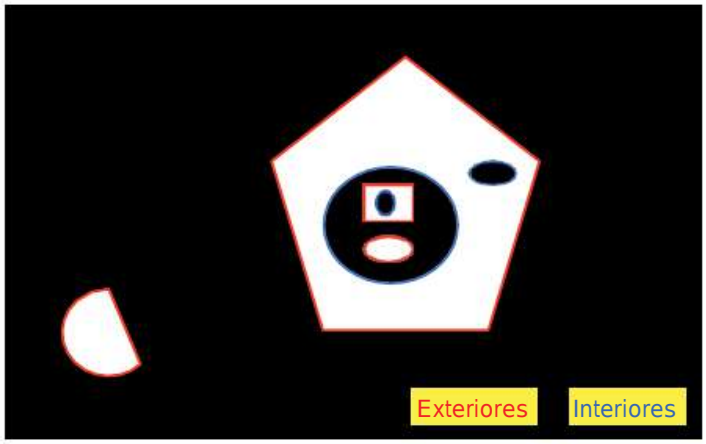


Figura 66. Resultado de findContours() con la opción RETR CCOMP. Los contornos se clasifican en exteriores e interiores en una jerarquía de dos niveles.

La función `findContours()` modifica la imagen binaria de entrada. Si se desea preservar, es necesario realizar previamente una copia.

Preguntas de repaso: Segmentación

3.3. Características

Tras la segmentación tendremos un contorno por cada objeto. Lo siguiente que tenemos que hacer es encontrar qué contornos pueden pertenecer a un círculo, para verificar a continuación si el círculo contiene en su interior dos o tres números.

Para determinar la forma de un contorno nos valdremos de una serie de características del contorno. Las características son valores numéricos que dan información sobre la posición, la forma, el tamaño, la orientación, etc. de los objetos. Una vez determinadas las características, realizaremos una selección de los objetos, ya que los objetos que buscamos tienen:

- Cierto tamaño: si fueran demasiado pequeños no podríamos leer los números.
- Cierta forma: son aproximadamente circulares.
- Cierta posición: queremos objetos que no estén cerca del borde de la imagen para estar seguros de que la señal esté entera.

3.3.1. El Bounding Box

El Bounding Box se define como el menor rectángulo que engloba a un conjunto de puntos. El Bounding Box se especifica con cuatro números:

- coordenadas (x, y) de la esquina superior izquierda,
- anchura y altura del rectángulo.

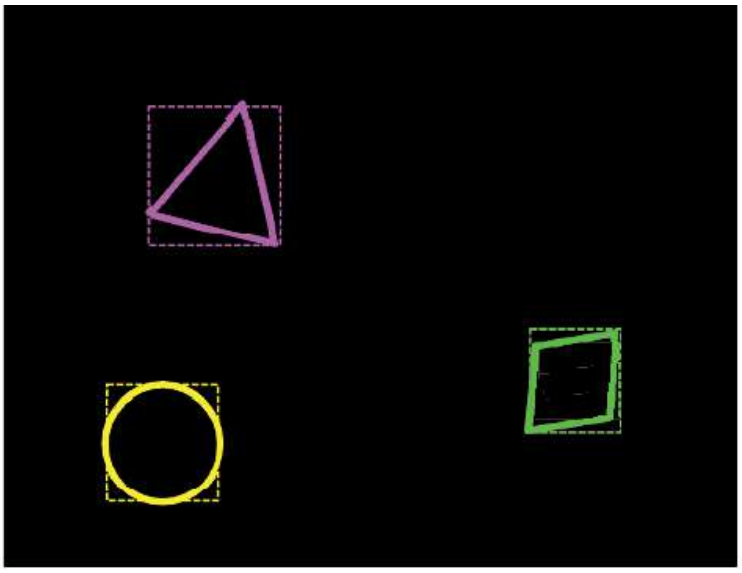


Figura 67: Concepto de Bounding Box.

En OpenCV se utiliza una clase especial llamada `Rect` para contener la información de un Bounding Box. Tiene cuatro campos:

```
Rect rect = new Rect();
rect.x ; // Coordenada x de la esquina superior izquierda
rect.y; // Coordenada x de la esquina superior izquierda
rect.width; //Anchura
rect.height; //Altura
```

Ejercicio: Selección de candidatos a círculos.

En este ejercicio partiremos de la umbralización adaptativa del gradiente y realizaremos una segmentación de la misma para seleccionar a continuación aquellos objetos que puedan ser círculos.

1. Haz una copia del programa binarización adaptativa.
2. Sobre dicha copia modifica la función `procesa()` para que se realice la segmentación y selección de candidatos a círculos. Las líneas a añadir se incluyen en el siguiente bloque. En dicho código se ha supuesto que el resultado de la binarización está en una `Mat` llamada `binaria`.

```
list<MatOfPoint> blobs = new Arraylist< MatOfPoint > () ;
Mat hierarchy = new Mat();
Mat salida = binaria.clone();//Copia porque finContours modifica entrada
Imgproc.cvtColor(salida, salida, Imgproc.COLOR_GRAY2RGBA);

Imgproc.findContours(binaria, blobs, hierarchy, Imgproc.RETR_CCOMP,
       Imgproc.CHAIN_APPROX_NONE );
int minimumHeight = 30;
float maxratio = (float) 0.75;
// Seleccionar candidatos a circulos
for (int c= 0; c< blobs.size(); c++ ) {
       double[] data = hierarchy.get(0,c);
       int parent = (int) data[3];
       if(parent < 0) //Contorno exterior: rechazar
              continue;
       Rect BB = Imgproc.boundingRect(blobs.get(c) );
       // Comprobar tamaño
       if ( BB.width < minimumHeight 11 BB.height < minimumHeight)
              continue;

       // Comprobar anchura similar a altura
       float wf = BB.width;
       float hf = BB.height;
       float ratio = wf / hf;
       if(ratio < maxratio 11 ratio > 1.0/maxratio)
              continue;

       // Comprobar no está cerca del borde
       if(BB.x < 2 11 BB.y < 2)
              continue;
       if(entrada.width() - (BB.x + BB.width) < 3 11 entrada.height() -
   (BB.y + BB.height) < 3)
              continue;

       // Aqui cumple todos los criterios. Dibujamos
   final Point P1 = new Point(BB.x, BB.y);
       final Point P2 = new Point(BB.x+BB.width, BB.y+BB.height);
```

```
        Imgproc.rectangle(salida, P1, P2, new Scalar(255,0,0) );
    } // for
return salida;
```

- Llamamos a la función `findContours()` con un argumento llamado `hierarchy` que es el que contiene la información sobre contornos externos e internos.
- Una vez realizada la segmentación, realizaremos un bucle que recorrerá todos los objetos. Nótese que este bucle tendrá muchas menos iteraciones que píxeles, ya que el número de objetos siempre debe ser mucho menor que el de píxeles:
- Dado que la imagen binaria que se ha segmentado es un gradiente, estaremos en el caso de objetos «delgados», y tendremos un contorno exterior y otro interior prácticamente superpuestos.
- Lo primero que haremos será procesar solo los contornos internos. La forma en que OpenCV devuelve la información en `hierarchy` es la siguiente:
    - Devuelve una imagen (matriz) de una fila con tantas columnas como contornos.
    - Cada píxel de `hierarchy` está formado por cuatro números (a diferencia de las imágenes RGB normales que tienen tres):
        - hermano anterior,
        - hermano posterior,
        - primer descendiente,
        - ascendente: contorno padre.
- Cuando no existe alguno de los elementos anteriores, el valor que toma es -1.
- Por tanto, los contornos exteriores serán aquellos en los que el contorno padre sea negativo.
- Los contornos interiores serán aquellos en los que el contorno padre sea mayor o igual que cero.
- Es interesante observar cómo se accede a un elemento de un `Mat` en java con el método `get(row, col)`.
- También es interesante ver cómo se accede a una componente de un píxel mediante el operador `[]`.
- El ejemplo muestra cómo usar la función `BoundingRect()` que, dado un contorno, determina el Bounding Box.
- El ejemplo también muestra cómo dibujar un rectángulo sobre una imagen.

3. Prueba el programa realizado sobre el dispositivo. Deberás observar líneas blancas en los contornos de los objetos y rectángulos rojos en los objetos que puedan ser círculos de las características adecuadas:

a. tamaño no demasiado pequeño,
b. anchura y altura similares,
c. no cerca del borde de la imagen.

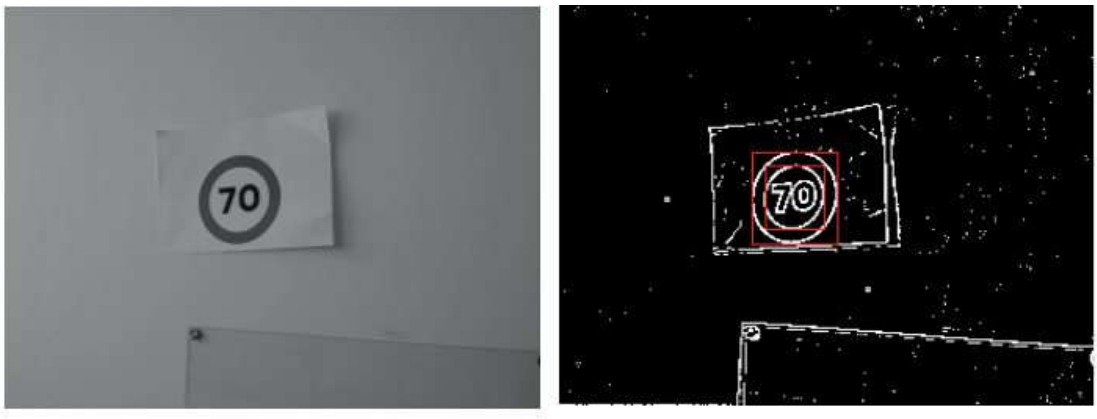


Figura 68. Resultado esperado de la selección de candidatos a círculos.

Práctica: Eliminar detecciones anidadas.

Intenta modificar el programa anterior con el fin de evitar que se detecten dos círculos concéntricos. Lo que se desearía es que en ese caso nos quedáramos con el más interior. Hay tres posibles soluciones:

1. Modificar el gradiente con el fin de que las dos circunferencias se unan en un único objeto separado de los números y detectar el contorno interior.
2. Detectar en una primera fase ambos contornos, como lo que ya está hecho, e implementar una función que determine si están anidados uno dentro de otro y en ese caso devuelva el más interior.
3. Al extraer los contornos, obtener la jerarquía completa y buscar la condición de anidamiento usando la jerarquía devuelta por `findContours()`.

Práctica: Detección señal basada en color.

Partiendo del programa que binarizaba las zonas rojas, realiza una segmentación y selecciona los objetos que puedan ser un círculo. Dibuja los rectángulos de las zonas candidatas con color azul.

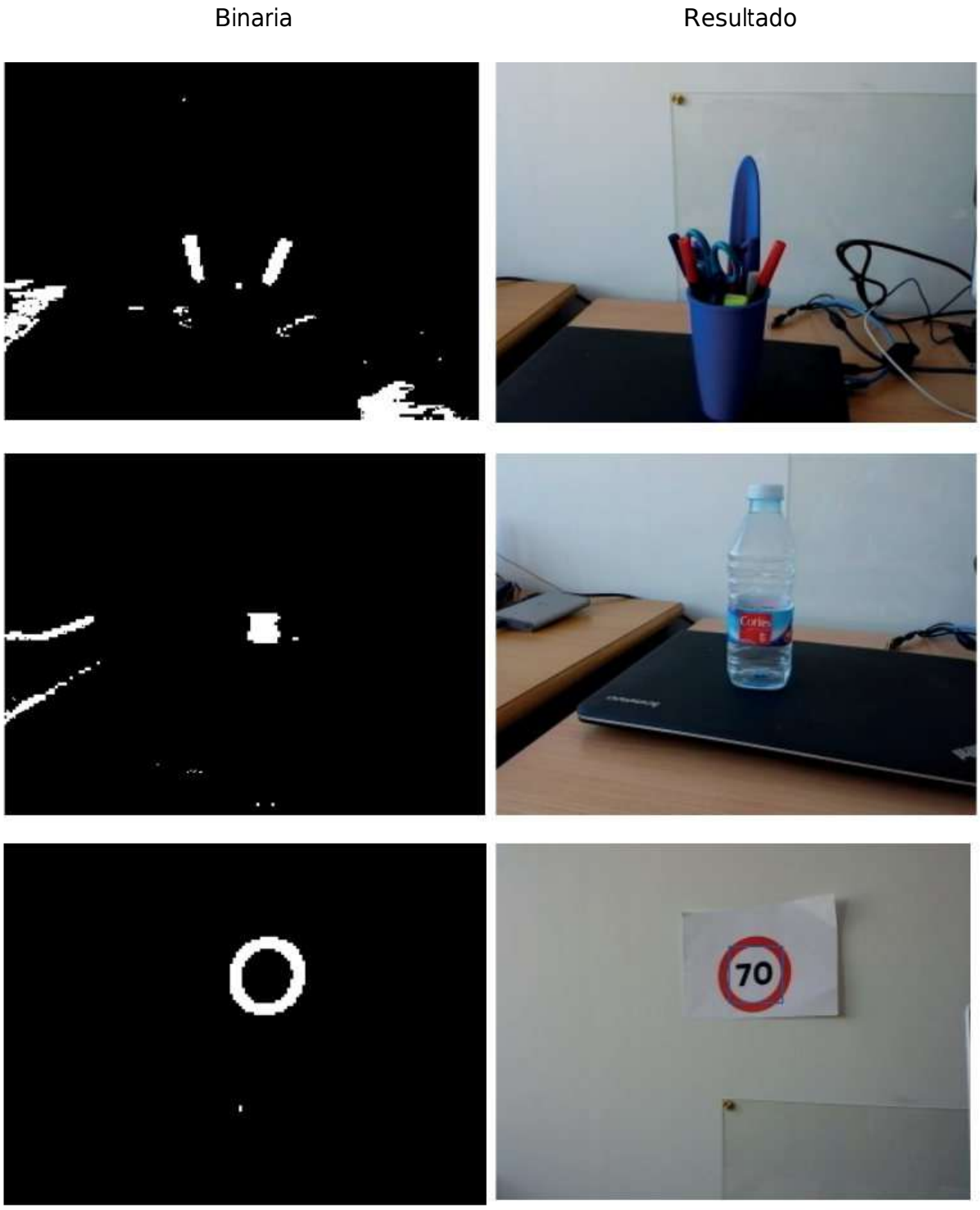


Figura 69. Resultado que se espera obtener. Observa que, aunque existen objetos rojos, solo se detectan aquellos que tienen un hueco interior y además cumplen los criterios de tamaño, proporción y posición.

3.3.2. Análisis del interior del rectángulo

En el anterior ejemplo hemos visto cómo seleccionar aquellos objetos que podrían ser círculos de una señal. El paso final consiste en analizar el interior de la señal para ver si tiene dos o tres números y proceder a leerlos. Existen distintas posibilidades de análisis. Aquí veremos una que suele proporcionar buenos resultados.

Una vez detectada una zona que puede ser un círculo, recortamos esa zona sobre la componente roja.

Figura 70. Componente roja con indicación de zona candidata a señal.

Si lo que estamos tratando realmente es el interior de una señal, los niveles de intensidad dentro del rectángulo seguirán una distribución bimodal:

- píxeles más oscuros en los números,
- píxeles más claros en el fondo blanco y el borde rojo (recordemos que estamos analizando la componente roja).

Mediante las opciones adecuadas de la función `threshold()` se puede binarizar el trozo de la señal usando el método de Otsu que determina el umbral automáticamente.

Figura 71. Zona recortada, binarización automática con Otsu, y lo mismo, pero invertido.

Una vez binarizada, el siguiente paso será volver a segmentar y seleccionar aquellos contornos que puedan ser números. Basado en el Bounding Box de los **contornos externos** podemos:

- Eliminar todos aquellos que toquen el borde del rectángulo (los números deben estar claramente dentro del mismo).
- Sean muy pequeños.
- Confirmar que la altura de los números debe estar en proporción al tamaño del círculo.
- Verificar que el centro de los números, en vertical, debe estar próximo al centro del rectángulo.

Si al final tenemos dos o tres objetos que cumplan los anteriores criterios, con gran probabilidad tendremos una señal de velocidad. Solo nos faltará leer los números y comprobar que el número tiene sentido.

Práctica: Segmentación de los dígitos dentro de una señal.

Partiendo de la práctica anterior, realiza una segmentación de los dígitos del interior de los círculos de color rojo. Selecciona los objetos que puedan ser un dígito y dibuja rectángulos de las zonas candidatas con color verde. La función procesa deberá tener el aspecto siguiente:

```
public Mat procesa(Mat entrada) { //entrada: imagen color
	Rect rectCirculo = new Rect();
	if (!localizarCirCuloRojo(rectCirculo))
		return entrada.clone();
	Mat salida = segmentarInteriorDisco(entrada, rectCirculo);
	return salida;
}
```

La función `localizarCirculoRojo()` deberá estar basada en lo que se hizo en la práctica anterior. Si hubiera más de un círculo, deberás detectar el mayor de ellos.

La función `segmentarInteriorDisco()` deberá:

1. Recortar de la imagen original la zona en la que se haya detectado un círculo rojo.
2. Extraer la componente roja y binarizarla usando Otsu. Ten en cuenta que los caracteres son más oscuros y por tanto deberás usar la opción:

```
Imgproc.THRESH_BINARY_INV+ Imgproc.THRESH_OTSU
```

3. A continuación realiza una segmentación y selecciona aquellos objetos que cumplan:
    a. Tener una altura mayor que la tercera parte del círculo.

b. Tener una altura mayor de 12 píxeles.

c. Tener una altura mayor que su anchura.

d. No tocar el borde del rectángulo del círculo.

4. Se devolverá una copia de la imagen de entrada con un rectángulo verde indicando la localización de los caracteres dentro del círculo.

3.4. Reconocimiento de dígitos

Tras la etapa anterior tendremos una serie de rectángulos que corresponden a las posiciones donde presumiblemente habrá un dígito. En esta etapa trataremos de identificar cada uno de estos dígitos, con lo que ya podremos saber la velocidad máxima indicada en la señal. Este tipo de problema suele conocerse como OCR (reconocimiento óptico de caracteres) y se suele abordar en dos sub-etapas.

1. Primero obtendremos, a partir del pequeño rectángulo binario, un conjunto de pocas características, en base a las cuales sea posible distinguir unos números de otros.
2. En segundo lugar trataremos de construir una función clasificadora, que tome como entrada un conjunto de características y nos de cómo salida el dígito correspondiente. Es decir, un valor entero entre 0 y 9.

La Figura 72 trata de ilustrar esta idea.

3.4.1. Extracción de características

La idea básica de las características para clasificar imágenes es que:

- Sean un conjunto pequeño de características.
- Que dos imágenes diferentes de un mismo tipo de dígito den como resultado características similares.
- Que dos imágenes de dígitos de clases diferentes den como resultado características bastante más diferentes que si fueran de la misma clase.

Lo siguiente que tenemos que hacer es elegir un conjunto de características para cada dígito que cumplan los requisitos antes comentados. Existen muchas posibilidades para esta elección. Aquí propondremos una que se ha verificado que funciona razonablemente bien.

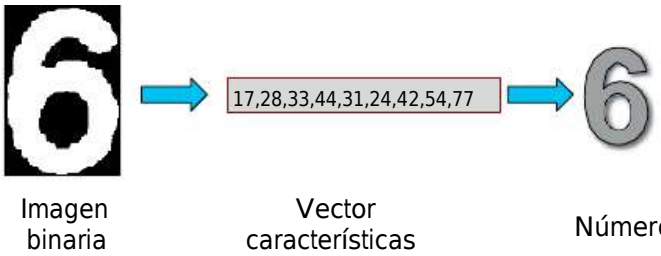


Figura 72. Reconocimiento a partir de un vector de características: a partir de la imagen binaria, obtenemos unas características en base a las cuales decidimos su clase.

3.4.1.1. Imagen de baja resolución

La Figura 73 muestra en qué consiste lo que se denomina imagen de baja resolución. El Bounding Box de la imagen binaria a reconocer se divide usando una rejilla de 3 x 3 elementos.

Si bien se podría elegir otro tamaño, este se ha comprobado que funciona correctamente. Un menor número de celdillas haría que las características obtenidas fueran poco discriminantes. Un número muy grande de celdillas haría que las pequeñas diferencias generadas por una binarización imperfecta tuvieran gran importancia.

A continuación, se calcula el valor medio de la intensidad en cada celdilla. Nótese que dicho valor medio corresponde al porcentaje de píxeles en blanco, pero puede representarse como una imagen de gris de 3 x 3. Si a continuación ponemos los 9 valores de dicha imagen de gris de 3 x 3 uno detrás de otro, obtendremos las características que necesitamos.

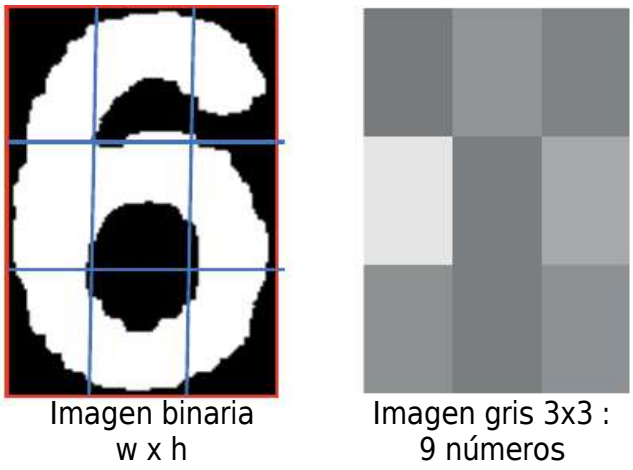


Figura 73. Imagen de baja resolución.

El siguiente método muestra una posible implementación para la extracción de estas características:

```
public Mat caracteristicas(Mat recorteDigito) {
                            //rectangulo: imagen binaria de digito
//Convertimos a flotante doble precisión
Mat chardouble = new Mat();
       recorteDigito.convertTo(chardouble, CvType.CV_64FC1);
//Calculamos vector de caracteristicas
       Mat digito_3x3 = new Mat();
       Imgproc.resize(chardouble, digito_3x3, new Size(3,3), 0,0,
                                                    Imgproc.INTER_AREA);
// convertimos de 3x3 a 1x9 en el orden adecuado
       digito_3x3 = digito_3x3.t();
       return digito_3x3.reshape(1, 1);
}
```

La imagen de entrada ha de coincidir con una imagen que contenga justo el dígito, como la de la izquierda de la figura anterior. Las operaciones que realiza la función son:

a. Convertir la imagen binaria de `unsigned char` a `double`. De esa manera, cuando calculemos la media dentro de cada celdilla, se tendrá mejor precisión.

b. Obtener la imagen de 3 x 3 de tipo double. Para ello es suficiente con redimensionar la imagen a un tamaño de 3 x 3. La opción Imgproc.INTER AREA hace referencia a la forma en que se calculan los píxeles de la salida. Esta opción significa que se calcula la media de los correspondientes píxeles de la entrada que se sitúen debajo.

c. Convertir la imagen de 3 x 3 a una imagen de 1 fila x 9 columnas (vector `caracteristicas`). El vector de características serán los 9 elementos de la imagen de baja resolución recorridos primero de arriba hacia abajo y luego de derecha a izquierda.

3.4.2. Clasificación

Un clasificador no es más que un bloque al que cuando le presentamos un vector de características a la entrada nos indica a qué clase pertenece.

Existen diferentes tipos de clasificadores, como redes neuronales, clasificadores SVM, de árbol … De hecho, OpenCV tiene un paquete que implementa muchos de ellos.

Nosotros aquí usaremos uno muy sencillo llamado clasificador de vecino más próximo.

Los clasificadores tienen múltiples aplicaciones tanto en el campo del análisis de imagen como en muchos otros ámbitos. Además del reconocimiento de caracteres, podemos citar otros como la detección y el reconocimiento facial, el reconocimiento de voz o la clasificación de correo como spam.

Independientemente del tipo de clasificador que se emplee, este tendrá unos parámetros que son los que al final determinan la función clasificadora. Estos parámetros se obtienen normalmente mediante una fase de entrenamiento. En esta fase se le proporciona al sistema una serie de ejemplos cuya clase es conocida, y mediante un algoritmo de aprendizaje se estiman los mejores parámetros del clasificador.

3.4.2.1. Entrenamiento

El entrenamiento consiste en proporcionar uno o más ejemplos de imágenes binarias de las cuales se sabe a qué número corresponden, y obtener sus características. El resultado del entrenamiento es una tabla que tiene tantas entradas como ejemplos de entrenamiento, y que para cada entrada contiene, las características obtenidas y de qué número se trata.

A partir de dichos datos es posible entrenar el clasificador.

3.4.2.2. Clasificador de vecino más próximo

En un clasificador de vecino más próximo se parte de una tabla de ejemplos anotados como la descrita más arriba y el objetivo es, dado un vector de características obtenido a partir de la imagen binaria de un número, predecir de qué número (clase) se trata.

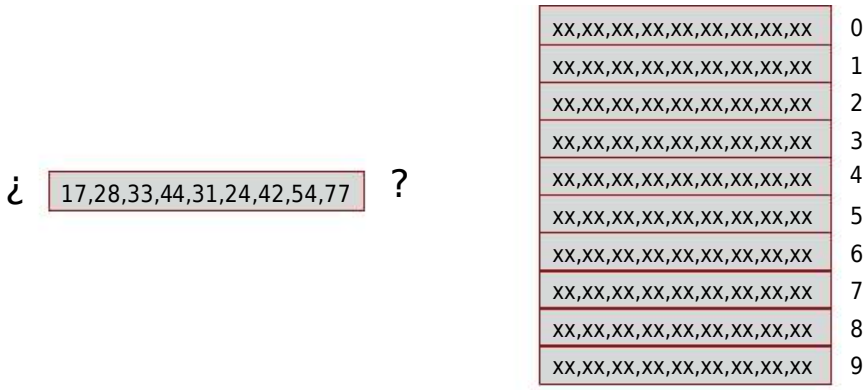


Figura 74. Concepto de clasificador de vecino más próximo.

La idea es sencilla. Se toma el vector desconocido y se calcula una distancia entre dicho vector y cada uno de los vectores de características de la tabla. Aquel vector de la tabla que resulte más similar al vector incógnita será el elegido y el número predicho será el asociado a dicho vector.

3.4.2.3. Medidas de distancia

En el punto anterior hemos visto que para utilizar un clasificador de vecino más próximo es necesario calcular la distancia entre el vector a clasificar y cada uno de los vectores de la tabla. Existen diversas maneras de calcular la distancia entre dos vectores. Aquí vamos a explicar 2:

- Distancia euclídea: basada en el teorema de Pitágoras.
- Distancia coseno: basada en el producto escalar y el ángulo que forman dos vectores.

La distancia euclídea corresponde a la clásica noción de distancia. Si tenemos dos vectores x, y, de N componentes cada uno, la distancia Euclídea viene dada por:

$$D_E \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}$$

El máximo parecido de dos vectores se logra cuando la distancia euclídea es cero, ya que en ese caso los vectores son idénticos.

La distancia coseno se calcula de acuerdo a la fórmula:

$$D_C = \frac{< x, y >}{\sqrt{< x, x >< y, y >}} = \frac{\sum_{i=1}^{N} x_i y_i}{\sqrt{(\sum_{i=1}^{N} x_i \, x_j) \; (\sum_{i=1}^{N} y_i \, y_i)}}$$

donde $< x, y >$ representa el producto escalar. La distancia coseno es una cantidad que siempre oscila entre -1 y 1. El máximo parecido se logra cuando la distancia coseno es próxima a 1. La distancia coseno es 1 cuando los dos vectores son proporcionales.

La diferencia entre la distancia coseno y la euclídea es que dos vectores proporcionales (no necesariamente iguales), tendrán una distancia coseno de valor 1 (máximo parecido), mientras que en el caso de la distancia euclídea solo dos vectores idénticos tendrán distancia cero.

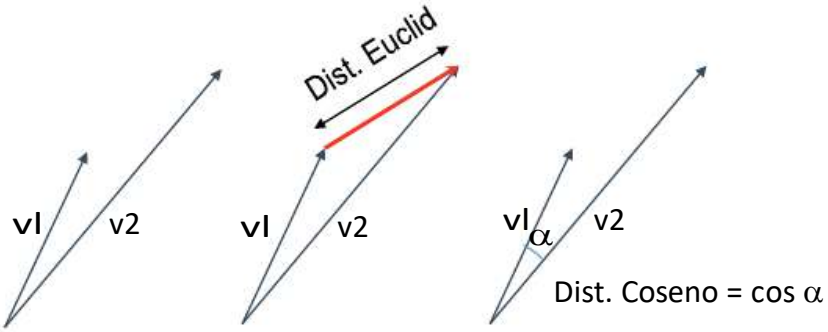


Figura 75. Medidas de parecido de dos vectores v1 y v2.
Centro: distancia Euclídea. Derecha: distancia coseno.

Ejercicio: Lectura de caracteres individuales. OCR-1

En este ejercicio, supondremos que la imagen que le vamos a mostrar a la cámara es un (único) número de los de una señal, y vamos a reconocerlo. El tipo de imagen a capturar es como el que se muestra en la Figura 76.

1. Haz una copia del programa Base5 y llámalo OCR1.
2. En la clase procesador, introduce las siguientes variables y el constructor:

```
public class Procesador {
	Mat tabla_caracteristicas;
	Mat binaria1;
	Mat binaria2;
	Mat entrada_gris;
	int NUMERO_CLASES = 10;
	int MUESTRAS_POR_CLASE = 2;
	int NUMERO_CARACTERISTICAS = 9;
	public Procesador() { //Constructor
		tabla_caracteristicas = new Mat(NUMERO_CLASES*MUESTRAS_POR_CLASE,
			NUMERO_CARACTERISTICAS, CvType.CV_64FC1);
		binaria1= new Mat();
		binaria2= new Mat();
```

```
                entrada_gris = new Mat();
                crearTabla();
        }
}
```

Además de las imágenes de trabajo, vamos a crear una variable para la tabla de las características de cada tipo de dígito. Dicha tabla la rellenaremos con la función `crearTabla()`.

3. En la clase procesador añade la función de crear tabla.

```
void crearTabla() {
        double datosEntrenamiento[][] = new double[][]{
        new double[]{0.5757916569709778, 0.8068438172340393,
0.6094995737075806, 0.6842694878578186, 0, 0.6750765442848206,
0.573646605014801, 0.814811110496521, 0.6094995737075806},
        new double[]{ 0.5408163070678711, 0.04897959157824516, 0,
0.8428571224212646, 0.79795902967453, 0.7795917987823486,
0.9938775897026062, 1, 0.995918333530426},
        new double[]{0.7524304986000061,  0.1732638627290726,
0.697916567325592,  0.6704860925674438,  0.3805555701255798,
0.9767361283302307, 0.6843749284744263, 0.7732638716697693,
0.6086806654930115},
        new double[]{0.6724254488945007, 0, 0.6819106936454773,
0.6561655402183533, 0.5406503081321716, 0.647357702255249,
0.6775066256523132, 0.8231707215309143, 0.732723593711853},
        new double[]{0.02636498026549816, 0.6402361392974854,
0.5215936899185181, 0.7385144829750061, 0.5210034847259521,
0.6062962412834167, 0.5685194730758667, 0.6251844167709351,
0.7910475134849548},
        new double[]{0.8133208155632019, 0.550218939781189,
0.6083046793937683, 0.7753458619117737, 0.4955636858940125,
0.6764461994171143, 0.4960368871688843, 0.8128473162651062,
0.6384715437889099},
        new double[]{0.6108391284942627, 0.985664427280426,
0.5884615778923035, 0.7125874161720276, 0.5996503829956055,
0.6629370450973511, 0.4828671216964722, 0.7608392238616943,
0.6695803999900818},
        new double[]{0.6381308436393738, 0, 0.1727102696895599,
0.7140188217163086, 0.5850467085838318, 0.8407476544380188,
0.943925142288208, 0.4654205441474915, 0.02728971838951111},
        new double[]{0.6880735158920288, 0.8049609065055847,
0.7363235950469971, 0.6299694776535034, 0.672782838344574,
0.6411824822425842, 0.6687054634094238, 0.7784574031829834,
0.7037037014961243},
        new double[]{0.6497123241424561,0.7168009877204895,
0.4542001485824585, 0.6476410031318665, 0.6150747537612915,
0.7033372521400452, 0.5941311717033386, 0.9686998724937439,
0.5930955410003662},
        new double[]{0.6764705777168274, 1, 0.7450980544090271,
0.7091502547264099, 0.05228758603334427,  0.6993464231491089,
0.6339869499206543, 0.9934640526771545, 0.7058823704719543},
        new double[]{0.3452012538909912, 0.3885449171066284, 0,
0.7770897746086121, 0.6501547694206238, 0.5789474248886108, 1, 1, 1},
```

```
        new double[]{0.6407563090324402, 0.06722689419984818,
0.7825630307197571, 0.7132352590560913, 0.6365545988082886,
0.9222689270973206, 0.7226890921592712, 0.5850840210914612,
0.7058823704719543},
        new double[]{0.5980392098426819, 0, 0.6666666865348816,
0.686274528503418, 0.5751633644104004, 0.6111111640930176,
0.6111112236976624, 0.7516340017318726, 0.7647058963775635},
        new double[]{0.03549695760011673, 0.717038631439209,
0.4705882370471954, 0.7474644780158997, 0.7109533548355103,
0.6531440615653992, 0.5862069725990295, 0.6744422316551208,
0.780933141708374},
        new double[]{0.6201297640800476, 0.5129870772361755,
0.5876624584197998, 0.7207792997360229, 0.5844155550003052,
0.6168831586837769, 0.5389610528945923, 0.8214285969734192,
0.7435064911842346},
        new double[]{0.6176470518112183, 1, 0.6764706373214722,
0.6699347496032715, 0.601307213306427, 0.6405228972434998,
0.5098039507865906, 0.7647058963775635, 0.8039215803146362},
        new double[]{0.7272727489471436, 0.0202020201832056,
0.2727272808551788, 0.8383838534355164, 0.8181818127632141,
0.7272727489471436, 0.8989898562431335, 0.1616161614656448, 0},
        new double[]{0.6928104758262634, 0.8071895837783813,
0.8333333134651184, 0.6764705777168274, 0.7026143074035645,
0.6209149956703186, 0.6601307392120361, 0.7712417840957642,
0.7941176891326904},
        new double[]{0.7320261597633362, 0.8202614784240723,
0.5653595328330994, 0.6503268480300903, 0.5882353186607361,
0.6732026338577271, 0.6045752167701721, 0.9869281649589539,
0.6339869499206543}};
        for (int i=0;i<20;i++)
                tabla_caracteristicas.put(i, 0, datosEntrenamiento [i]);
}
```

Para cada digito se han introducido dos muestras. Las características del dígito 0 se almacenan en las posiciones 0 y 10. Las del dígito 1 se almacenan en 1 y 11. Y así sucesivamente.

4. Añade la función `procesa()` en la clase `Procesador`. Esta será la que se llamará desde la actividad principal:

```
public Mat procesa(Mat entrada) {
        Imgproc.cvtColor( entrada, entrada_gris, Imgproc.COlOR_RGBA2GRAY);
        Rect rect_digito = new Rect();
        boolean localizado = localizarCaracter(rect_digito);
        if(localizado == false)
                return entrada.clone();
        //Recortar rectangulo en imagen original
        Mat recorte_digito = entrada_gris.submat(rect_digito);
        //Binarizacion Otsu
        Imgproc.threshold(recorte_digito, binaria2, 0, 255,
                        Imgproc.THRESH_BINARY_INV+ Imgproc.THRESH_OTSU);
        //leer numero
```

```
        int digito = leerRectangulo(binaria2);
        Mat salida = dibujarResultado(entrada, rect_digito, digito);
        return salida;
}
```

En esta función se pueden ver los pasos principales que vamos a seguir: convertir la imagen a grises, localizar el carácter a leer, en caso de encontrarlo recortar la imagen, y binarizar aplicando Otsu. Finalmente se lee el número y se muestra el resultado. Obsérvese cómo la imagen que le damos a Otsu como entrada es una imagen de gris bimodal, ya que es un pequeño rectángulo que contiene el dígito (ver Figura 76).

5. A continuación, añade el código de la función para localizar el carácter en la clase `Procesador`. En esta función, tras una binarización inicial, se realiza una segmentación y buscamos el objeto más grande. Si dicho objeto más grande es demasiado pequeño o toca el borde de la imagen, diremos que no hemos localizado nada.

```
boolean localizarCaracter(Rect digit_rect) {
int contraste = 5;
        int tamano = 7;
        Imgproc.adaptiveThreshold(entrada_gris, binaria1, 255,
                Imgproc.ADAPTIVE_THRESH_MEAN_C, Imgproc.THRESH_BINARY_INV,
                tamano, contraste );
        list<MatOfPoint> contornos = new Arraylist< MatOfPoint > () ;
        Mat jerarquia = new Mat();
        Imgproc.findContours(binaria1, contornos, jerarquia,
                        Imgproc.RETR_EXTERNAl, Imgproc.CHAIN_APPROX_NONE );
        int altura_minima = 30;
        int anchura_minima = 10;
        int max_area=-1;
                // Seleccionar objeto mas grande
        for (int c= 0; c < contornos.size(); c++ ) {
                Rect bb = Imgproc.boundingRect( contornos.get(c) );
                        // Comprobar tamaño
                if (bb.width < anchura_minima 11 bb.height < altura_minima)
                        continue;
                        // Descartar proximos al borde
                if(bb.x < 2 11 bb.y < 2)
                        continue;
                if(binaria1.width() - (bb.x + bb.width) < 3 11 binaria1.height() -
                                                        (bb.y + bb.width) < 3)
                        continue;
                        // Seleccionar el mayor
                int area = bb.width * bb.height;
                if( area > max_area ) {
                        max_area=  area;
                        digit_rect.x = bb.x;
                        digit_rect.y = bb.y;
                        digit_rect.width = bb.width;
                        digit_rect.height = bb.height;
                }
        }
```

```
        if( max_area < 0) return false; //No se ha detectado objeto valido
        else return true;
}
```

6. Añade el método `características()` que se ha visto antes. Recuerda que este método generaba un vector con las nueve características a partir de la imagen binaria del dígito.

7. Seguidamente, necesitaremos implementar la función que lee cada carácter:

```
public int leerRectangulo(Mat rectangulo) {
        Mat vectorCaracteristicas = caracteristicas(rectangulo);
// Buscamos la fila de la tabla que mas se parece
        double Sumvv = vectorCaracteristicas.dot(vectorCaracteristicas);
        int nmin = 0;
        double Sumvd = tabla_caracteristicas.row(nmin).dot(
                                                vectorCaracteristicas);
        double Sumdd = tabla_caracteristicas.row(nmin).dot(
                                        tabla_caracteristicas.row(nmin));
        double D = Sumvd / Math.sqrt(Sumvv * Sumdd);
        double dmin = D;
        for(int n= 1; n< tabla_caracteristicas.rows() ; n++) {
                Sumvd = tabla_caracteristicas.row(n).dot(vectorCaracteristicas);

                Sumdd = tabla_caracteristicas.row(n).dot(
                                            tabla_caracteristicas.row(n));
                D = Sumvd / Math.sqrt(Sumvv * Sumdd);
                if( D > dmin ){
                        dmin = D;
                        nmin = n;
                }
        }
        nmin = nmin % 10; // A partir de la fila determinamos el numero
        return nmin;
}
```

   a. Obtener el vector de características.

   b. Calcular la distancia con cada una de las filas de la tabla.

   c. Devolver el dígito correspondiente al máximo parecido.

8. Por último implementaremos una función que escriba el número leído sobre la propia imagen de entrada:

```
Mat dibujarResultado(Mat imagen, Rect digit_rect, int digit) {
        Mat salida = imagen.clone();
        Point P1 = new Point(digit_rect.x, digit_rect.y);
        Point P2 = new Point(digit_rect.x+digit_rect.width,
                                            digit_rect.y+digit_rect.height);
        Imgproc.rectangle(salida, P1, P2, new Scalar(255,0,0) );
        // Escribir numero
        int fontFace = 6;//FONT_HERSHEY_SCRIPT_SIMPlEX;
        double fontScale = 1;
        int thickness = 5;
```

```
        Imgproc.putText(salida, Integer.toString(digit ),
                        P1, fontFace, fontScale,
                        new Scalar(0,0,0), thickness, 8,false);

        Imgproc.putText(salida, Integer.toString(digit ),
                        P1, fontFace, fontScale,
                        new Scalar(255,255,255), thickness/2, 8,false);
        return salida;
}
```

9. Ejecuta el programa sobre una imagen como la que se muestra. Deberías observar el dígito correcto. En caso de que el número leído no coincida, **prueba a rotar el teléfono** por si la imagen estuviera boca abajo. La figura siguiente muestra los principales pasos que sigue el algoritmo anterior

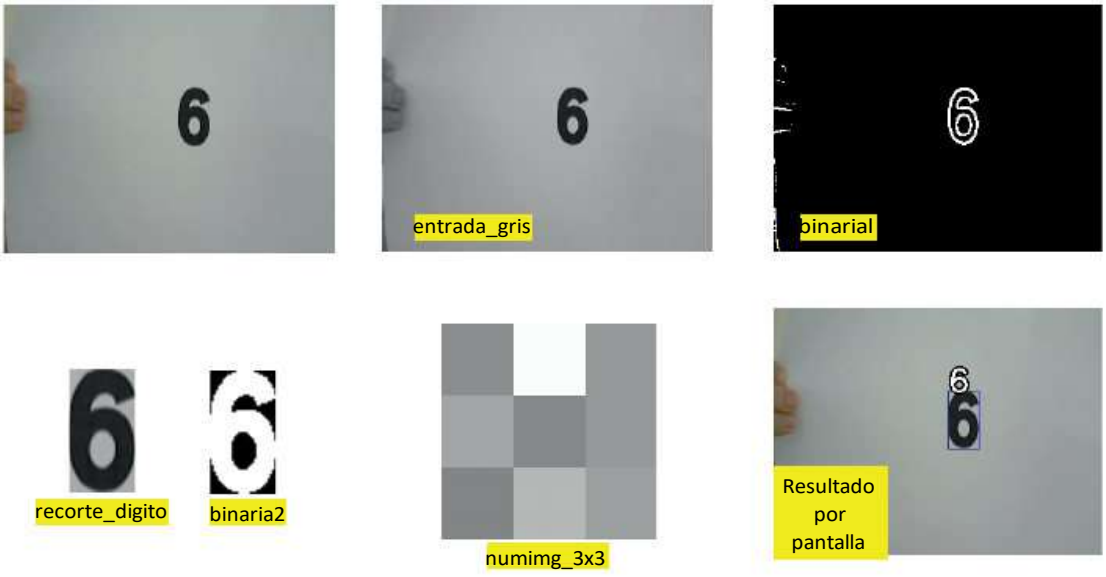


Figura 76. Principales pasos del programa para reconocer un número.

Práctica: Lectura de los caracteres dentro de un círculo rojo.

Partiendo del ejemplo OCR-1 ahora se pretende leer, no un número aislado, sino una secuencia de números dentro de un círculo rojo. La función procesa deberá tener el aspecto siguiente:

```
public Mat procesa(Mat entrada) {
//entrada: imagen color
        Rect rect_circulo = new Rect();
        boolean localizado = localizarCirCuloRojo(entrada, rect_circulo);
        if(localizado == false)
                return entrada.clone();
        Mat circulo = entrada.submat(rect_circulo); //Recorte zona de interes
        String cadenaDigitos = analizarInteriorDisco(circulo);
        if(cadenaDigitos.length() == 0)
```

```
            return entrada.clone();
        Mat salida = dibujarResultado(entrada, rect_circulo, cadenaDigitos);
        return salida;
}
```

La función `localizarCirculoRojo()` deberá estar basada en lo que hizo para detectar círculos rojos y que se muestra en la página 101. Si hubiera más de un círculo, deberás detectar el mayor de ellos.

La función `analizarInteriorDisco()` estará basada en la práctica Segmentación de los dígitos dentro de una señal, donde localizabas cada uno de los caracteres dentro del círculo. Deberás añadir además el reconocimiento de dígitos que se ha visto en el ejemplo OCR-1.

1. Recorta de la imagen original la zona en la que se haya detectado un círculo rojo.
2. Se leerán cada uno de los objetos y se generará una cadena de texto con todos los números leídos. En el caso de que el número de caracteres encontrados no sea 2 o 3, se devolverá la cadena "".

Por último deberás implementar un método `dibujarResultado()` que muestre los números leídos de modo similar a la imagen que se muestra en la Figura 77.

Desafío: Orden Correcto de los caracteres.

Cuando pruebes el programa, observarás que a veces los números se leen en orden correcto, y a veces no. Eso es porque el orden en que se procesan los objetos viene dado por el orden en que OpenCV ha realizado la segmentación.

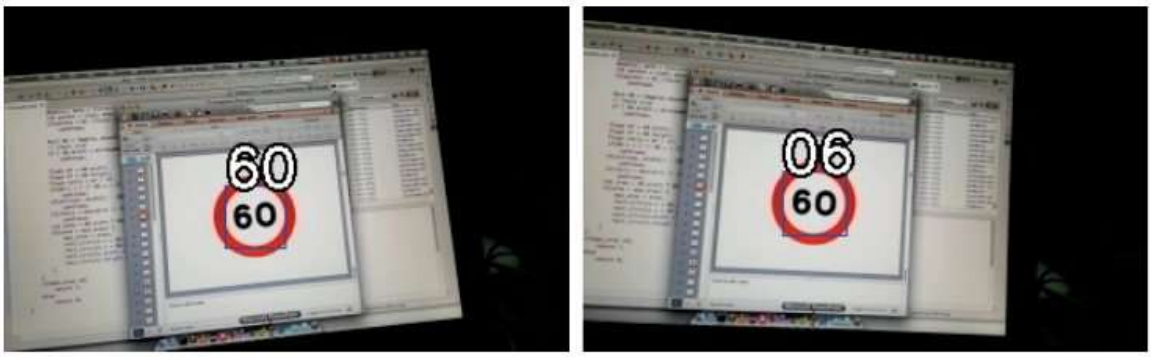


Figura 77. Números leídos en orden correcto e incorrecto.

1. Intenta modificar el programa para que los caracteres se lean en el orden correcto.

3.5. Conclusión

En este tema hemos visto cómo, basándonos en el resultado del preproceso, es posible localizar objetos en la escena con unas determinadas características: objetos circulares de borde rojo con dos o tres números negros en el interior.

También hemos visto cómo realizar un clasificador elemental que permita reconocer los dígitos. Las ideas expuestas en este tema se pueden aplicar al reconocimiento de otros tipos de objetos.

PARTE 2.

Internet de las Cosas

Por JESÚS TOMÁS

CAPÍTULO 4.

Android Things: Entradas / Salidas

Por JESÚS TOMÁS

El término “Internet de las Cosas” es sin duda una de las palabras de moda dentro y fuera de los círculos tecnológicos. Como suele pasar en estos casos, se ha extendido tanto y en tan poco tiempo que ha dado pie a multitud de interpretaciones y definiciones sobre cuáles son exactamente sus capacidades, o sobre donde empieza y donde termina exactamente un proyecto de Internet de las Cosas. Sin embargo, su propio nombre es tan autodescriptivo que no deja lugar a dudas de cuáles son sus pilares fundamentales: **las Cosas**, esa interacción con el mundo físico, una digitalización mediante soluciones electrónicas y de sistemas embebidos que permite transformarlas al mundo digital; e **Internet**, esa conectividad que nos permite integrar el mundo físico en una red de comunicaciones, y convertirlo en servicios para poder interactuar con él desde cualquier lugar.

Android Things es la plataforma de Google para el desarrollo en dispositivos en entornos de Internet de las Cosas. Nos permite aplicar todo el potencial de Android a este nuevo mundo.

En este primer capítulo haremos una introducción general del sistema y realizaremos la instalación. Luego pasaremos a mostrar el hardware utilizado en el capítulo (Raspberry Pi). La parte más importante del capítulo es la descripción de los diferentes tipos de entrada/salida y cómo se programan. El capítulo acaba con la implementación de controladores de usuario y la integración de Google Assistant SDK en Android Things.

Objetivos:

- Mostrar en que consiste Android Things y cómo puede utilizarse para el desarrollo de aplicaciones de Internet de las cosas.
- Describir los mecanismos de seguridad y actualizaciones propuestos por Google.
- Enumerar las diferentes plataformas hardware soportadas, haciendo hincapié en Raspberry Pi.
- Describir el proceso de instalación y de las herramientas de desarrollo.
- Repasar algunos conceptos de electrónica usados en la unidad.
- Conocer las principales características y aprender a utilizar entradas / salidas. En concreto GPIO, PWM, I^2C y UART.
- Comparar el uso de microcontroladores y microprocesadores a la hora de controlar sensores.
- Utilizar un microcontrolador programado con Arduino como esclavo de la Raspberry Pi.
- Aprender a implementar controladores de usuario.
- Integrar Google Assistant SDK en Android Things.

4.1. Internet de las cosas

Internet de las cosas (IoT), es una de las tendencias más prometedoras en tecnología. De acuerdo a muchos analistas, IoT puede ser la tecnología más activa en la próxima década. Tendrá un gran impacto en nuestras vidas y promete modificar nuestros hábitos.

La primera vez que se utilizó el término Internet de las Cosas fue en 1999 por Kevin Ashton, profesor del MIT. Utilizó la siguiente definición: "El IoT es el mundo en el que cada objeto tiene una identidad virtual propia y capacidad potencial para integrarse e interactuar de manera independiente en la red con cualquier otro individuo, ya sea una máquina o un humano."

Internet de las cosas incluye todos los objetos que pueden conectarse a Internet, para intercambiar información. Estos objetos pueden estar conectados en cualquier momento y en cualquier lugar. En resumen, IoT es un ecosistema donde los objetos están interconectados y, al mismo tiempo, se conectan a Internet.

El concepto de objetos conectados no es nuevo y con el paso de los años se ha desarrollado. El nivel de la miniaturización de circuitos y la potencia creciente de la CPU con un consumo menor hace posible que se imagine un futuro donde hay millones de "cosas" que se hablan entre sí.

Algunos ejemplos interesantes pueden ser el Amazon Dash Button, con solo pulsar un botón realizamos un pedido de un determinado producto (ver izq.). O el Amazon Key, que nos permite controlar la apertura de la puerta de casa a distancia (ver derecha).

Hay varios elementos que contribuyen a crear el ecosistema de IoT y es importante entender el papel que juegan. El componente básico de IoT es un objeto inteligente. Es un dispositivo que se conecta a internet y es capaz de intercambiar datos. Puede ser un sensor simple que simplemente toma un valor y lo transmite, o un sistema más complejo, con una mayor capacidad de procesamiento.

Todos los electrodomésticos de una casa son claros candidatos a convertirse en objetos inteligentes. Pero existen un abanico mucho más amplio wearables, automoción, vigilancia, ciudades inteligentes, …

Existe una gran variedad de alternativas para que los objetos inteligentes obtengan conectividad: Ethernet, WiFi, Bluetooth, LoWPAN, NB-IoT, Zigbee, redes celulares, LoRA, … También hay varios protocolos de aplicación ampliamente utilizados: HTTP, MQTT, CoAP, AMQP, XMPP, Stomp, …

4.2. Introducción a Android Things

Android Things es la nueva plataforma de Google para el desarrollo en dispositivos incrustados, que trabajan en entornos de Internet de las Cosas. Nos permite aplicar todo el potencial de Android a este nuevo mundo.

Android Things fue anunciado en Google I/O 2015 con nombre inicial de Brillo. Aunque en los primeros desarrollos tenía bastantes diferencias con Android (por ejemplo, la no posibilidad de usar Java), la versión finalmente lanzada en el Google I/O 2018, es prácticamente un sistema Android adaptado a plataformas basadas en microprocesador.

La característica más destacable sería la rapidez en poner en marcha un producto comercial, sin necesidad de tener conocimientos en el diseño de sistemas integrados. Esto es posible gracias a que nos ofrece una solución completamente administrada.

4.2.1. Solución completamente administrada

Desarrollar y comercializar un producto de IoT es un proceso muy complejo. Tenemos que dominar conceptos de electrónica, sistemas de microprocesadores, comunicaciones y software. Podemos destacar dos problemáticas particulares donde es especialmente complicado dar con una solución satisfactoria:

Actualizaciones de software: Va a ser inevitable que periódicamente tengamos que actualizar el software de nuestro producto para resolver errores en el desarrollo, agujeros de seguridad, mejoras de prestaciones, cambios de política de servicios o la realización de test A/B de usabilidad. Si dejamos las actualizaciones en manos de los usuarios, muchos nunca lo realizarán. Por lo que será más interesante realizar estas actualizaciones de forma transparente al usuario.

Seguridad: Un dispositivo de IoT puede recopilar información muy sensible para nuestros usuarios (cámaras, hábitos de vida,). Toda comunicación entre dispositivos o con la nube no ha de poder ser interceptada. Otra situación, si cabe más peligrosa es la posibilidad de que se introduzca software malicioso en el dispositivo. Tendríamos que garantizar que el software no ha sido reemplazado. Incluso en caso de que se tuviera acceso completo al dispositivo.

El ciclo de desarrollo de un producto que cumpla estas características suele ser elevado. Seleccionar el hardware, microprocesadores, protocolos, etc. podría demorarse meses o incluso años. Google nos ofrece una solución con la que podríamos disponer de un prototipo en función de semanas y llevarlo a fase de producción en cuestión de meses. Además, no resulta imprescindible tener conocimientos sobre sistemas integrados ni seguridad.

La solución está basada en una arquitectura de cuatro niveles:

En el nivel inferior tendríamos el hardware. Como veremos más adelante Google ha certificado cuatro plataformas (de NXP, Qualcomm y MediaTek) que podrán ser usadas en productos comerciales y dos solo para la fase de prototipo (de NXP y Raspberry Pi).

En el siguiente nivel encontramos el sistema operativo, por supuesto se trata de Android Things. Como se ha comentado no difiere demasiado de una distribución convencional. Simplemente se han añadido nuevas librerías para acceder a sensores y se han eliminado otras que no tendrían sentido en estos dispositivos.

En el siguiente nivel nos permite incorporar todos los servicios de Google o de terceros con los que estamos familiarizados en el desarrollo de aplicaciones Android. Entre estos servicios podemos destacar: Google Play Services, Google Cloud Platform, Cloud IoT Core, Firebase, TensorFlow, ...).

En el último nivel tenemos el nivel de aplicación, donde entra en juego nuestro desarrollo. Aunque existen algunas peculiaridades, como la gestión de permisos o el arranque de la aplicación, una aplicación para Android Things es básicamente igual que una para Android. Como se muestra en el gráfico disponemos de los mismos lenguajes de programación: Java, Kotlin o C++.

4.2.1.1. Modelo de actualizaciones

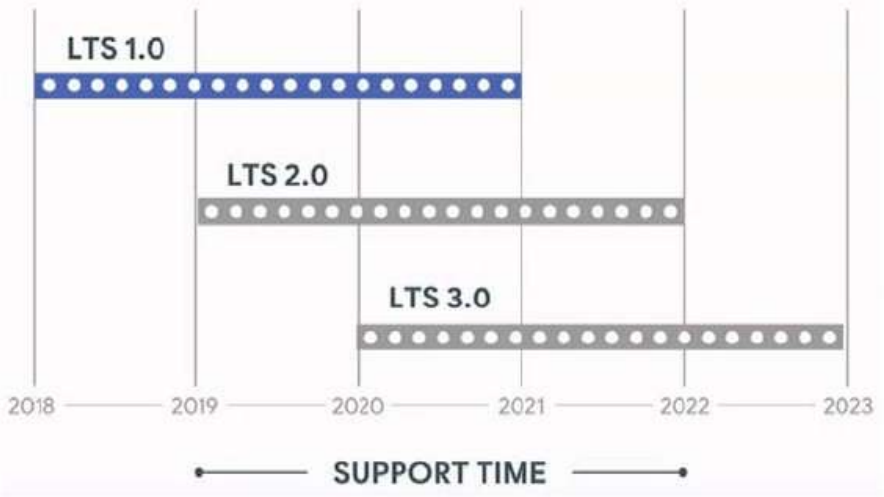
Para permitir que la solución sea completamente administrada juega un papel fundamental la consola. Una de las principales funciones de esta consola es subir las actualizaciones en línea (OTA) a los dispositivos. Estas actualizaciones pueden incluir la aplicación del fabricante o la imagen del sistema.

Google se compromete a hacerse cargo de las actualizaciones de forma gratuita durante los primeros tres años de soporte, siempre que ejecute la "versión de soporte a largo plazo" de Android Things. Las actualizaciones automáticas están habilitadas de manera predeterminada y llegarán como actualizaciones de seguridad mensuales y ocasionales actualizaciones importantes del sistema operativo. Google también menciona que después de tres años habrá "opciones adicionales para soporte extendido".

Si Google ha de hacer las actualizaciones le da un mayor control sobre el SO. En el modelo de actualizaciones tradicional de Android, las actualizaciones eran responsabilidad del fabricante. Esto era así porque este podía, y solía, modificar a su gusto el SO. A ningún fabricante le gusta un modelo de hardware interoperable (como el que se utiliza en los PC), donde el hardware es poco relevante. Todos quieren diferenciarse de la competencia y personalizar el software es un factor clave. El problema de este modelo es que muchos fabricantes no realizaban actualizaciones o dejaban de hacerlo en modelos más antiguos.

Google quiere cambiar esto, aprovechando su preponderancia en el mercado quiere forzar a los fabricantes a cambiar la política de actualizaciones. La idea es separar el sistema operativo Android de los controladores y firmware específicos del hardware en cada teléfono Android, esta iniciativa se conoce como "Proyecto Treble". Comenzando desde Android O, Google lanza actualizaciones de seguridad mensuales de Android directamente a los teléfonos, saltándose a los fabricantes.

Las actualizaciones de seguridad están garantizadas por Google durante un periodo de tres años. De forma prestablecida, estas se producirán de forma automática, cada vez que Google lanza una actualización del sistema o nosotros actualizamos la aplicación. Las únicas actualizaciones que se realizarán de forma automática son las de la versión principal. Es decir, cuando salga la versión 2.0 de Android Things, no estaremos obligados a utilizarla, pero seguiremos recibiendo actualizaciones para la versión 1.X.



4.2.1.2. Ejemplo de productos basados en Android Things

El primer producto comercial basado en Android Things es el altavoz inteligente LG WK7, que además de escuchar música permite interactuar con el asistente de Google. Otro tipo de dispositivo donde se está utilizando Android Things es en las pantallas inteligentes. Este concepto fue utilizado por primera vez en Amazon Echo Show, básicamente añade una pantalla a un altavoz inteligente como el Amazon Echo o Google Home. Varias marcas, como LG, Lenovo y J BL, lanzarán en verano del 2018 sus modelos basados en Android Things. Como característica destacable de todos estos productos se puede destacar el breve periodo de tiempo desde que se concibe la idea hasta que el producto está en el mercado.

Vídeo[Tutorial]: What's new in Android Things (Google I/O '18)

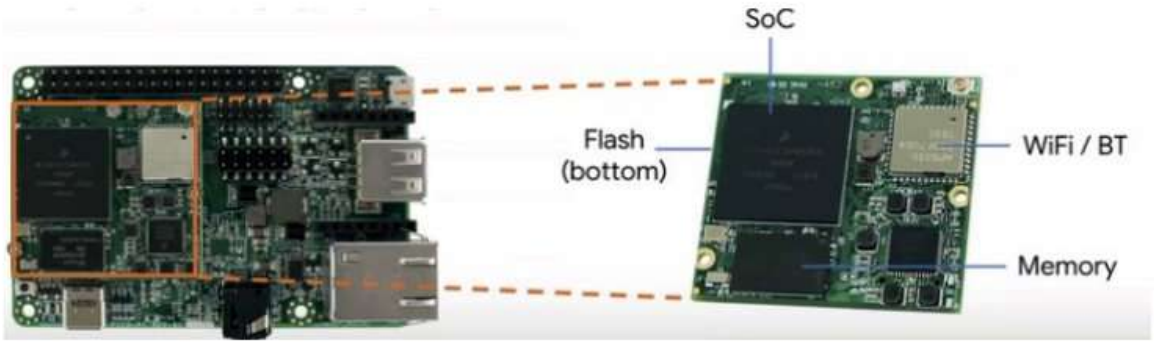
Preguntas de repaso: Introducción a Android Things

4.2.2. Plataformas hardware soportadas

Como hemos comentado Google trabaja con varios fabricantes para certificar que plataformas podrán recibir actualizaciones directamente desde Google.

Para definir estas plataformas se utiliza el concepto de SoM (System on Module) que es muy parecido al concepto de SoC (System on Chip). En ambos casos tenemos una CPU, RAM, almacenamiento, WiFi, E/S y todo lo demás que necesita para tener un ordenador, en un SoM se integra en una pequeña placa y en un SoC se integra en un chip de silicio. Los SoM son simplemente más grandes, y más baratos, que los SoC, ya que los componentes están esparcidos por una pequeña placa de circuito sin necesitar tanto nivel de integración.

Tanto si utilizamos un SoM como un SoC, es frecuente montarlos en una placa formando un SBC (Single Board Computer) donde se añaden E/S adicionales, conectores, alimentación, etc.



Ejemplo de un SoC, integrado en un SoM, integrado en un SBC.

En la siguiente tabla se muestra los cuatro SoM que actualmente soportan Android Things:

Platform	NXP i.MX8M  Learn More	Qualcomm SDA212  Learn More	Qualcomm SDA624  Learn More	MediaTek MT8516 
CPU & Memory	• NXP i.MX8M • 1.5Ghz quad-core ARM Cortex A53 • 1GB or 2GB RAM	• Qualcomm Snapdragon™ 212 • Quadcore 1.267Ghz ARM Cortex A7 • 1GB RAM	• Qualcomm Snapdragon™ 624 • Octacore 1.8Ghz ARM Cortex A53 • 2GB RAM	• MT 8516 • 1.3Ghz quad-core ARM Cortex A35 • 512MB RAM
GPU	QC7000Lite	QC Adreno 304	QC Adreno 506	N/A
Storage	4GB eMMC	4GB eMMC	4GB eMMC	4GB eMMC
Display	MX8-DSI-OLED1	N/A	8-inch WXGA Innolux Display with Touch	N/A
Camera	OV5640 MIPI CSI	N/A	Omnivision OV5693 5MP sensor	N/A
Networking	10/100/1000 Ethernet Wi-Fi 802.11ac Bluetooth® 4.2	Wi-Fi 802.11ac (2.4/5.0GHz) Bluetooth® 4.2	Wi-Fi 802.11ac (2.4/5.0GHz) Bluetooth® 4.2	10/100/1000 Ethernet Wi-Fi 802.11ac (2.4/5.0GHz) Bluetooth® 5.0
USB	2x USB 3.0 Type C	2x USB 2.0 Host 1x USB 2.0 OTG	1x USB 3.0 Type C	1x USB 2.0 Host 1x USB 2.0 OTG
Size (width x length)	50.3mm x 50.3mm	50mm x 46.5mm	50mm x 46.5mm	N/A
Type	Physical	Physical	Physical	Virtual

Adicionalmente se proporcionan plataformas de desarrollo para permitir la creación de prototipos y realizar las pruebas. No cumplen con los requisitos de seguridad de Google para la certificación de clave de identificación y arranque verificado, y no pueden recibir actualizaciones de seguridad:

	NXP Pico i.MX7D	Raspberry Pi 3 Model B
Platform		
CPU & Memory	• NXP i.MX7D • 1GHz dual-core ARM Cortex A7 • 512MB RAM	• Broadcom BCM2837 • 1.2GHz quad-core ARM Cortex A53 • 1GB RAM
GPU	N/A	VideoCore IV
Storage	4GB eMMC	MicroSD card slot
Display	LCD8000-43T VL050-80128NM-C01	HDMI Raspberry Pi Touch Display
Camera	OV5640 MIPI CSI	RPi Camera module v2
Networking	10/100/1000 Ethernet Wi-Fi 802.11ac (2.4/5.0GHz) Bluetooth® 4.1	10/100 Ethernet Wi-Fi 802.11n (2.4GHz) Bluetooth® 4.1
USB	1x USB 2.0 Host 1x USB 2.0 OTG	4x USB 2.0 Host
Size (width x length)	37mm x 40mm	85mm x 56mm (complete board)
Type	Physical	Physical

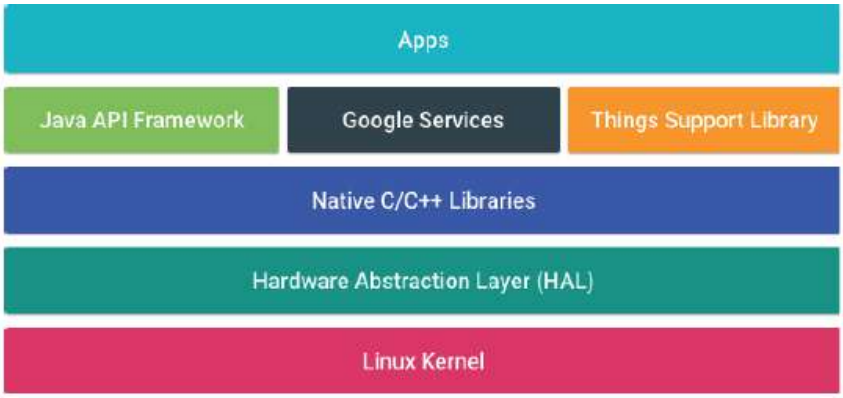
Los SoM pueden ser físicos o virtuales. En el primer caso un SoM es una placa que puede extraerse físicamente. En el segundo es un diseño de referencia proporcionado por el fabricante y certificado por Google. Luego podemos integrar los componentes individuales de ese diseño directamente en un producto. Los SoM virtuales son buenos para productos de gran volumen en los que se necesita flexibilidad y control para diseñar los componentes que se ajusten al diseño de la placa y las dimensiones del producto.

Para casa SoM Google nos proporciona un Board Suport Package (BSP). Por lo que no es necesario interactuar con los fabricantes del hardware. Además, nos garantiza una sencilla migración en caso de cambiar de SoC.

Preguntas de repaso: Plataformas hardware compatibles

4.2.3. SDK de Android Things

El SDK de Android Things es muy similar al de Android. Básicamente es extendido con el módulo Things Support Library, que nos facilita el acceso al hardware.



Podemos destacar las siguientes diferencias con respecto a la programación para dispositivos móviles.

- Acceso más flexible a periféricos y controladores.
- Las aplicaciones del sistema no están presentes para optimizar los requisitos de inicio y almacenamiento.
- Los dispositivos solo muestran una aplicación a los usuarios, en lugar de múltiples como los dispositivos móviles. Esta aplicación es arrancada automáticamente al inicio. No se da al usuario la posibilidad de arrancar aplicaciones.

Dentro de Things Support Library se han añadido los siguientes APIs:

- **Bluetooth**: emparejamiento y conexión con dispositivos.
- **Device Updates**: actualizaciones de software basadas en OTA (Over The Air).
- **LoWPAN**: Acceso a redes de área personal inalámbricas de baja potencia (LoWPAN)
- **NDK**: El Kit de Desarrollo Nativo se integra de forma predeterminada. Nos permite desarrollar las aplicaciones en C / C++.
- **Peripherical I/O**: Permite la comunicación con sensores y actuadores utilizando protocolos e interfaces estándar de la industria. Es compatible con GPIO, PWM, I2C, SPI y UART.
- **User Driver**: Los controladores de usuario nos permiten registrar nuevos drivers de dispositivo desde la aplicación. Podemos inyectar eventos de hardware al sistema, que otras aplicaciones podrán capturar.
- **Settings**: Nos permite configurar la pantalla, hora del sistema y configuraciones regionales.

Las siguientes características no serían soportadas en las APIs indicadas:

Característica	API
Interfaz de usuario (barra estado, navegación, quick settings)	NotificationManager KeyguardManager WallpaperManager
`VoiceInteractionService`	SpeechRecognizer
`android.hardware.fingerprint`	FingerprintManager
`android.hardware.nfc`	NfcManager
`android.hardware.telephony`	SmsManager TelephonyManager
`android.hardware.usb.accessory`	UsbAccessory
`android.hardware.wifi.aware`	WifiAwareManager
`android.software.app_widgets`	AppWidgetManager
`android.software.autofill`	AutofillManager
`android.software.backup`	BackupManager
`android.software.companion_device_setup`	CompanionDeviceManager
`android.software.picture_in_picture`	Activity Picture-in-picture
`android.software.print`	PrintManager
`android.software.sip`	SipManager

4.2.3.1. Cambios de comportamiento

Aplicaciones comunes no disponibles: No se incluyen muchas de las aplicaciones comunes en los teléfonos móviles, como el teléfono, contactos, Settings, calendario, . Por lo tanto, no podremos usar las intenciones comunes que utilizaban estas aplicaciones o los proveedores de contenido estándar.

Salida por pantalla opcional: Algunos dispositivos disponen de salida HDMI donde podremos conectar una pantalla. Podremos diseñar la salida por pantalla de igual forma como lo hacíamos en Android. La ventana de la aplicación ocupa todo el espacio de la pantalla. Android Things no incluye la barra de estado del sistema ni los botones de navegación. Sin embargo, muchos de los sistemas a desarrollar no van a necesitar salida por pantalla. A pesar de este hecho, las actividades siguen siendo el componente principal de la aplicación. El sistema seguirá enviando los eventos de entrada a la actividad en primer plano, que tenga el foco.

Actividad de arranque (home): En Android Things no hay una aplicación de arranque (home), desde donde el usuario pueda lanzar otras aplicaciones. Se espera que una de las aplicaciones instaladas asuma este rol y se ejecute tras el arranque del sistema. Para ello, la aplicación ha de tener una actividad con un filtro de intención que incluya `CATEGORY_DEFAULT` y`IOT_LAUNCHER`.

Declaración de permisos: Los permisos necesarios han de ser declarados en el manifiesto. Los permisos peligrosos se otorgan al arrancar el dispositivo sin la verificación del usuario. Por lo tanto, tras instalar una aplicación con permisos peligrosos tendrás que reiniciar el sistema.

Notificaciones: No son soportadas. Al no existir la barra de estado, no podrían mostrarse.

Soporte para servicios de Google: Muchas de las Apis de Google tienen soporte para Android Things. Sin embargo, aquellas que requieren de una entrada directa del usuario o de su autentificación no van a ser soportadas. Cada versión de Android Things incluye la última versión estable de Google Play Services. Esta versión no podrá ser actualizada a través de Google Play Store, al no estar incluido en Android Things. En la siguiente tabla se muestran las Apis soportadas y no soportadas:

Servicios soportados	Servicios no soportados
Awareness, Cast, Google Analytics for Firebase, Firebase Authentication, Firebase Cloud Messaging (FCM), Firebase Crash Reporting, Firebase Realtime Database, Firebase Remote Config, Firebase Storage, Fit, Instance ID, Location, Maps, Nearby Connections, Nearby Messages, Places, Mobile Vision, SafetyNet	AdMob, Android Pay, Drive, Firebase App Indexing, Firebase Dynamic Links, Firebase Invites, Firebase Notifications, Play Games, Sign-In

4.2.4. Consola Android Things

Desde la consola de Android Things podremos instalar y actualizar la imagen del sistema en los dispositivos hardware. Nos permite:

- Descargar e instalar la última imagen del sistema Android Things.
- Crear imágenes de fábrica que contengan aplicaciones OEM junto con la imagen del sistema
- Instalar actualizaciones de la aplicación de forma online (OTA - On The Air).
- Monitorear las aplicaciones para comprobar qué están funcionando correctamente.

Para acceder a la consola: https://partner.android.com/things/console/

Más recursos: https://androidthings.withgoogle.com

Preguntas de repaso: SDK de Android Things

4.3. Raspberry Pi 3

En este capítulo vamos a trabajar con un dispositivo concreto, por lo que va a ser interesante conocer sus características y disponer de ciertos detalles que nos permitirán realizar las conexiones.

A este dispositivo se le conoce como Low Cost Single Board Computers o sistema embebido. Y suele emplearse para realizar una o algunas funciones dedica-

das durante la fase de prototipo. Es decir, se trata de un ordenador de reducido precio, integrado en una pequeña placa que puede ser utilizado para resolver problemas concretos.

4.3.1. Comparativa con otros modelos

El modelo Raspberry Pi 3 es en la actualidad el más popular, pero existen gran variedad de modelos. Puedes consultar una tabla comparativa en[1]. A continuación, destacamos algunos modelos:

Raspberry Pi 3 B (35€ PCcomponenntes, 30€ Aliexpress)
4 x ARM Cortex-A53 a 1,2 GHz, RAM 1G, Ethernet, Wifi, Bluetooth, HDMI, 4xUSB 2.0, Jack audio, ranura SD

Raspberry Pi Zero W (13,9€ Aliexpress)
Broadcom BCM2835 a 1 GHz, RAM 512M, HDMI, 1x microUSB, ranura SD

Orange Pi Plus 2 (12,3€ Aliexpres)
H3 Quad-core Cortex-A7 a 1,6GHz, H.265/HEVC 4K, RAM 2G, Ethernet, Wifi, ~~Bluethooth~~, HDMI, 4xUSB 2.0, ranura SD, 16 G Flash

Intel Edison (91€)

2 x Atom Silvermont a 0,5GHz + 1 x Quark a 0,1GHz, RAM 4G, Wifi, Bluetooth, USB 2.0, ranura SD
https://software.intel.com/es-es/iot/hardware/edison

Nota: Solo Raspberry PI 3 B e Intel Edison son compatibles con Android Things.

[1] http://socialcompare.com/en/comparison/raspberry-pi-alternatives

4.3.2. Características

La Raspberry Pi 3 está basada en el System on Chip (SoC) BCM2837 de Broadcom. Un SoC es un ordenador completo integrado en un chip, es decir, un procesador, RAM, GPU y demás funciones. Este modelo es un ARM de 64bits a 1.2GHz, quad-core y con 1GB de RAM. Se parece mucho a los SoC utilizados en los teléfonos inteligentes.

Este chip se monta en una placa de desarrollo junto a otros elementos: Un chip LAN9514 para controlar los puertos USB y Ethernet. Otra parte importante para este capítulo es que cuentan con una conexión GPIO (General Purpose Input Output), similares a las de los microcontroladores que solemos utilizar con Arduino. Eso sí, funcionan a 3.3v y no a 5v. El sistema operativo va en una tarjeta Micro SD. Si queremos que este sea Android Things tendrá que ser de al menos 8GB.

1.- cuatro puertos USB 2,0 hub

2.- Ethernet 10/100 Mbps (modelo+ 1000Mbps)

3.- Alimentación con conector micro USB (usar cargador de 2,5A)

4.- salida HDMI (full HD)

5.- conector de cámara CSI

6.- conector tipo Jack 3,5mm de salida de audio y vídeo compuesto

7.- ranura para tarjeta de memoria micro SD

8.- conector de pantalla DSI

9.- WiFi 802.11n y Bluetooth 4.1 (modelo+ 802.11ac dual band y Bluetooth 4.2 LS BLE)

10.- CPU Broadcom BCM2837 64bit quard-core a 1,2GHz, con 1 GB RAM (modelo+ BCM2837B0 a 1,4GHz)

11.- Controlador USB/Ethernet

12.- Conector de 40 pines con salidas GPIO

Una de las ventajas de este tipo de dispositivos es su gran versatilidad. Podemos adquirirlos para aprender a realizar nuestros proyectos y luego utilizarlos para otros propósitos, como un PC de escritorio, una consola de videojuegos, un centro multimedia para la TV o un rúter Wifi.

4.3.3. Alternativas para el Sistemas Operativos

Realmente Raspberry Pi es un ordenador de bajo coste y por lo tanto podremos instalar diferentes Sistemas Operativos (SO).

El SO más utilizado es **Raspbian**. Podríamos decir que es el oficial, al haberse creado expresamente para este dispositivo. Se trata de una distribución de Linux basada en Debian. Hay otras distribuciones de Linux que también puedes instalar como **Fedora** o **Arch Linux**, que disponen de versiones especiales para Raspberry Pi.

Aunque no hay una imagen oficial de Android para Raspberry Pi, existen alternativas de terceros para conseguirlo. Un ejemplo lo tenemos es **RaspAnd**, con el que podremos ejecutar aplicaciones y juegos de Android. Incorpora el gestor de contenido Kodi y el navegador Firefox.

Una opción muy popular es instalar en la Raspberry Pi un gestor de contenido multimedia. Por ejemplo, **OSMC** o **OpenElec** (Open Embedded Linux Entertainment Center), ambos basado en Kodi. Si solo te interesa escuchar música instala **Pi MusicBox**.

También se puede usar como NAS (Almacenamiento conectado en red (NAS) para disponer de nuestro sistema de almacenamiento en la nube. El chip que controla los puertos USB y la conexión Ethernet es el mismo (LAN9514), lo que produce un cuello de botella.

Otra opción muy popular es usarla como video consola. Con este fin podemos instalar **RetroPie**, que nos permite ejecutar videojuegos antiguos de más de 50 sistemas. Además, permite instalar Kodi como reproductor multimedia.

También puedes instalar **Kano**, un sencillo sistema operativo para que los niños tengan un primer contacto con la informática.

Para el desarrollo de dispositivos de Internet de las cosas, además de Linux, puedes instalar **Windows 1O IoT Core**, la propuesta de Microsoft para este mundo. Necesitarás un PC con Windows y Visual Studio. Y por supuesto la alternativa que estudiaremos en este capítulo, **Android Things**.

Preguntas de repaso: Raspberry Pi 3

4.4. Instalación de Android Things

Para realizar este apartado vas a necesitar el siguiente material:

- Raspberry Pi 3 B
- Tarjeta microSD con al menos 8GB
- Adaptador para leer tarjeta microSD en el PC
- Cable de alimentación microUSB
- Cable HDMI, un monitor y ratón (opcional)
- Cable Ethernet (opcional)

4.4.1. Descarga de la Imagen del Sistema de Android Things

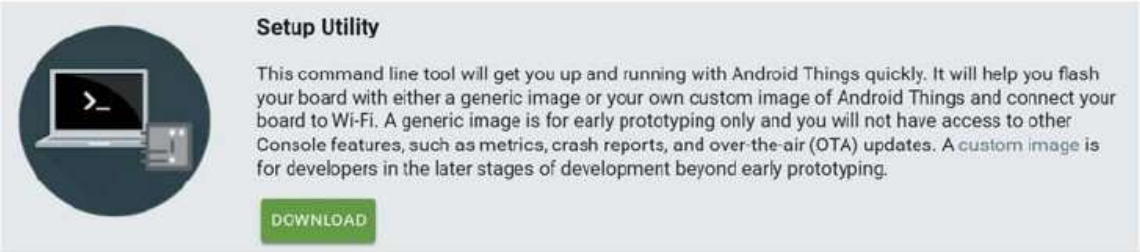
Ejercicio: Instalar Android Things en la memoria SD

Para poder disfrutar de Android Things en la Raspberry Pi vamos a tener que instalar el sistema operativo en la memoria SD.

Nota: La realización de este ejercicio puede durar alrededor de una hora.

1. Descarga la herramienta de configuración en línea desde la consola: https://developer.android.com/things/preview/download.html

   En el menú selecciona la opción Tools y luego el botón DOWNLOAD.



2. Descomprime el fichero y ejecuta la herramienta. En Windows, haz clic en el botón derecho en el archivo ejecutable y seleccione Ejecutar como administrador. En Mac o Linux, utiliza un comando similar al siguiente desde un terminal:

```
$ sudo ~ / Downloads / android - things - setup - utility / android -
things - setup - utility - linux
```

3. Selecciona la opción `1-Install Android Things and optionally set up Wi-Fi`

```
Android Things Setup Utility (version 1.0.19)
-----------------------------
This tool will help you install Android Things on your board and set up
Wi-Fi.

What do you want to do?
1 - Install Android Things and optionally set up Wi-Fi
2 - Set up Wi-Fi on an existing Android Things device
```

4. Selecciona la opción 1 (`Raspberry Pi 3`)

```
What hardware are you using?
1 - Raspberry Pi 3
2 - NXP Pico i.MX7D
3 - NXP Pico i.MX6UL
```

5. Selecciona de nuevo la opción 1:

```
Do you want to use the default image or a custom image?
1 - Default image: Used for development purposes. No access to the Android
Things Console features such as metrics, crash reports, and OTA updates.
2 - Custom image: Upload your custom image for full device development and
management with all Android Things Console features.
```

6. Tras descargar la imagen nos pedirá que introduzcamos la memoria SD en el ordenador y seleccionemos el adaptador:

```
Plug the SD card into your computer. Press [Enter] when ready

Running Etcher-cli...
? Select drive (Use arrow keys)
> \\.\PHYSICALDRIVE2 (8.0 GB) - SD/MMC Card Reader USB Device
```

7. Confirma que quieres borrar todos los datos de la SD. El proceso de instalación puede tardar hasta 20 minutos y el de verificación otro tanto.

```
? This will erase the selected drive. Are you sure? Yes
Flashing [------------------------] 100º eta 0s
Validating [------                  ] 24º eta 19m42s
```

Nota: Si no usar esta herramienta de configuración, puedes descargar una imagen de Android Things (por ejemplo, desde la consola) y escríbala en la tarjeta microSD. Para ello puedes usar uno de estos links:

- Windows
- Mac
- Linux

8. Inserta la tarjeta microSD en la ranura en la parte inferior de la Raspberry Pi.

4.4.2. Configuración de la conexión a Internet

Hasta que no lo conectemos a Internet no podremos decir que realmente es un dispositivo de IoT. La forma habitual es utilizar Ethernet o WiFi, pero también podríamos conectarnos por USB, Bluetooth, etc. Un dispositivo Android Things puede trabajar en condiciones muy diferentes. La forma más sencilla de conectarnos a Internet va a ser utilizando un monitor o un cable Ethernet. En ocasiones no ten-

dremos esta posibilidad, a lo largo de este punto vamos ver diferentes alternativas para configurar la conexión.

Ejercicio: Configurar WiFi con monitor

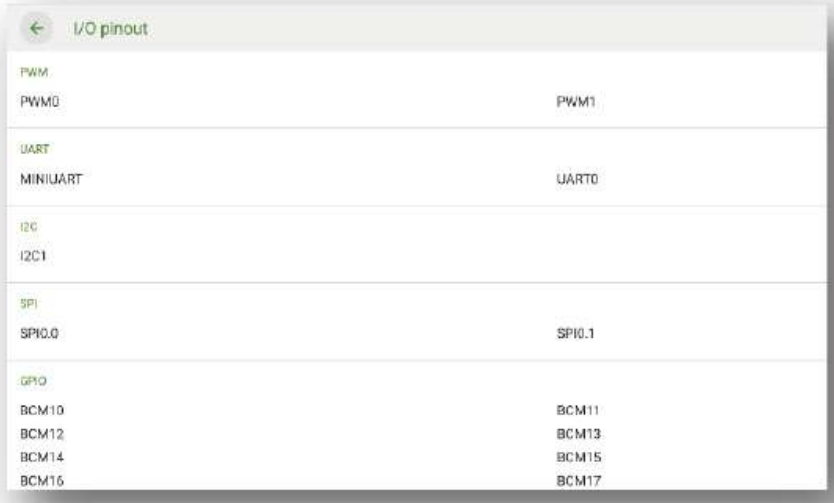
1. Conecta cables de alimentación al conector mini USB y el monitor al conector HDMI, tal como se muestra en la figura. El arranque puede durar unos minutos.

2. Si tienes un cable Ethernet conéctalo a tu red de área local. El servidor de DHCP asignará una IP al dispositivo. Podrás averiguar la IP que ha sido asignado a la Raspberry Pi, al ser mostrada en la pantalla:

3. Si no la has conectado por Ethernet puedes conectarla a una red WiFi. Conecta un ratón a un conector USB para poder interaccionar con la pantalla y selecciona la opción CONNECT TO NETWORK.

Nota: Si no tienes acceso la red local por Ethernet, ni WiFi, puedes hacer una de las siguientes cosas: Conecta el cable Ethernet a tu ordenador y asigna a la Raspberry Pi una dirección IP usando DHCP. O conecta un cable serie de la Raspberry Pi a tu ordenador y usa una consola serie para conectarse a WiFi .

4. Desde la pantalla inicial dispones de tres opciones:
   - **General**: muestra la versión del sistema operativo, actualizar parches de seguridad, rearrancar el dispositivo o resetear con valores de fábrica.
   - **Networks**: permite configurar la conexión por Ethernet y WiFi.
   - **Peripherals**: muestra los periféricos, entre los que se incluyen los pines de entrada / salida (I/O Pinout):



Ejercicio: Configurar WiFi usando Ethernet

En muchas ocasiones un dispositivo Android Things no está conectado a un monitor. En este caso vamos a disponer varias opciones para realizar la conexión a Internet. Veamos como conectarlo con un cable Ethernet.

1. Conecta la Raspberry Pi con un cable Ethernet conéctalo a tu red de área local. El servidor de DHCP asignará una IP al dispositivo. El problema es cómo averiguar la IP. Para ello puedes conectarte al switch y ver los dispositivos conectados con sus respectivas IP. Si no tienes acceso, puedes utilizar una herramienta de barrido de IPs, para ver las IPs asignadas en tu red (por ejemplo: http://angryip.org/).
2. Para asegurarte que es la IP correcta prueba el comando (cambiando la IP por la obtenida):

```
adb connect <dirección-ip>
```

Ejercicio: Configurar WiFi usando Setup Utility

A diferencia de Ethernet en WiFi vamos a necesitar configurar la red y contraseña desde la Raspberry Pi.

1. Ejecuta la herramienta Android Things Setup Utility usada en el ejercicio donde instalamos la imagen de las SD.
2. Selecciona la opción 2:

```
Android Things Setup Utility (version 1.0.19)
-----------------------------
This tool will help you install Android Things on your board and set up
Wi-Fi.

What do you want to do?
1 - Install Android Things and optionally set up Wi-Fi
2 - Set up Wi-Fi on an existing Android Things device
```

3. Selecciona la opción 1:

```
What hardware are you using?
1 - Raspberry Pi 3
2 - NXP Pico i.MX7D
3 - NXP Pico i.MX6UL
```

4. Te pedirá que conectes el dispositivo mediante un cable Ethernet. Pulsa Enter y tratará de encontrarla de forma automática.
5. Si no la encuentra te pedirá que introduzcas su IP que ha obtenido a través de Ethernet.
6. Te pedirá el nombre SSID de la red y la contraseña para establecer la conexión.

Ejercicio: Configurar WiFi usando adb

Veamos un método alternativo para configurar WiFi en el dispositivo.

1. Abre una consola Shell en el dispositivo. Para ello tienes dos alternativas:
   - Asigna una IP usando un método alternativo. Ejecuta el comando `adb connect <dirección-ip>`
   - Conecta un cable serie como se describe en el ejercicio: Conexión por cable USB.
2. Ejecuta el comando `adb shell`. Este comando te permite abrir un shell Linux en el dispositivo.
3. Envía una intención para arrancar el servicio de Wi-Fi, indicando el SSID de tu red:

```
$ am startservice \
    -n com.google.wifisetup/.WifiSetupService \
    -a WifiSetupService.Connect \
    -e ssid <SSID_de_tu_red> \
```

Además, puedes añadir alguno de los siguientes parámetros:

`-e passphrase <contraseña>` Añádelo si tu red requiere contraseña.

`-e passphrase64 <contraseña_en_base64>` Añádelo si la contraseña contiene alguno de los siguientes caracteres `espacio`, `!`, `"`, `$`, `&`, `'`, `(`, `)`, `;`, `<`, `>`, `,` o `1`. La contraseña ha de estar codificada en base64 (www.base64encode.org).

`--ez hidden true` Añádelo si el SSID de tured es oculto.

Nota: Introduce el comando en una línea, sin los caracteres`\`

4. Verifica que se ha conectado utilizando el `logcat`:

```
$ logcat -d 1 grep Wifi
...
V WifiWatcher: Network state changed to CONNECTED
V WifiWatcher: SSID changed: ...
I WifiConfigurator: Successfully connected to ...
```

5. Testea que tienes conectividad:

```
$ ping 8.8.8.8
```

6. Testea que la fecha y hora han sido configuradas a través de la red:

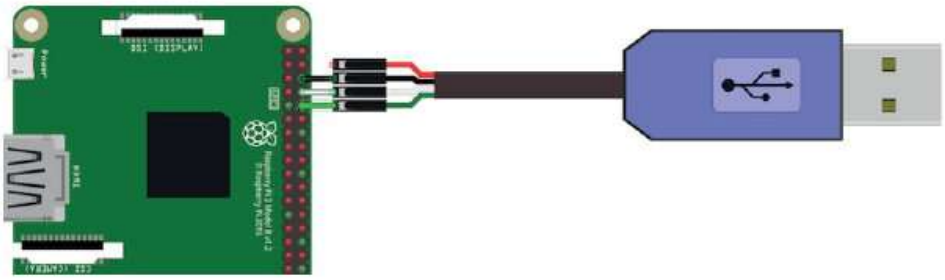
```
$ date
```

Una configuración incorrecta de fecha puede causar errores con SSL. Si es necesario reinicia el dispositivo.

Ejercicio: Conexión por cable USB

Si no dispones de conexión por WiFi o Ethernet existe una tercera posibilidad que consiste en usar un cable USB para conectarte a una consola serie. Podrás depurar el dispositivo y revisar la información de registro del sistema. Desde esta consola podrás ver los mensajes de registro del kernel (dmesg) y tendrás acceso a un shell completo que puede usar para acceder a comandos como `logcat`.

1. Consigue a un cable USB a TTL (https://www.adafruit.com/products/954):
2. Conéctalo como se muestra en la figura:



3. Conecta el USB a tu ordenador.
4. Abre un programa de terminal como PuTTY (Windows), Serial (Mac OS) o Minicom (Linux). Configura los parámetros del puerto serie siguientes: velocidad: 115200 baudios, bits de datos: 8, paridad: ninguna, bits de parada: 1.

4.4.3. Acceder al dispositivo desde Android Studio

Una vez conectado a Internet y conocida su IP podemos conectarlo con Android Studio para instalar y depurar aplicaciones.

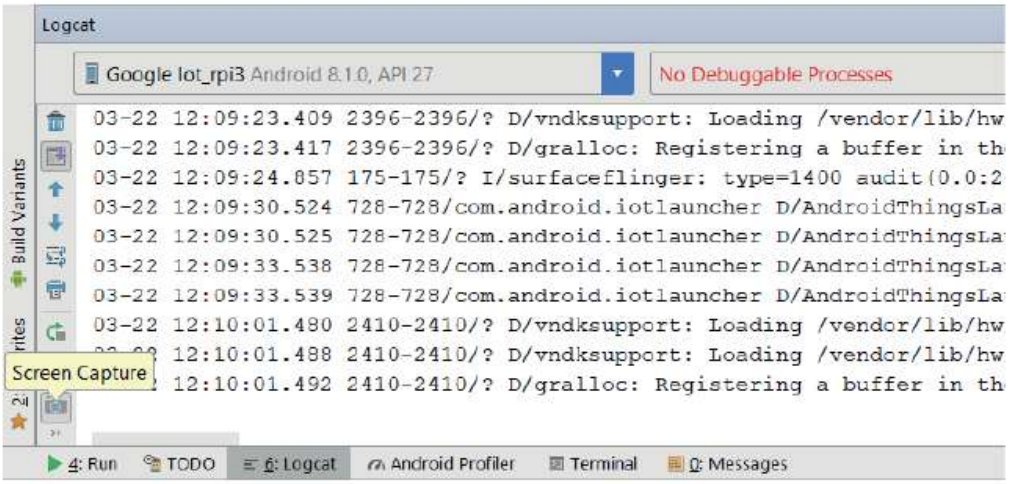
Ejercicio: Conectar el dispositivo con Android Studio

1. Desde una consola ejecuta el siguiente comando:

```
adb connect <dirección-ip>
```

   Ha de aparecer `connect to <dirección-ip>`

2. Abre la pestaña Logcat y selecciona el dispositivo. Aparecerán todos los mensajes de Log:



3. Selecciona el icono Screen Capture que se muestra en la imagen anterior. Se mostrará la pantalla del dispositivo. Esta opción es interesante cuando no dispongas de monitor.
4. Abre la pestaña Android Profiler e investiga el estado de CPU, memoria y red.

4.4.4. Un primer proyecto

Ejercicio: Primer proyecto Android Things

En este ejercicio aprenderás a crear tu primera aplicación para Android Things.

1. Crea un nuevo proyecto con los siguientes datos:

   `Application name: Android Things`

   `Package name: org.example.androidthings`

   `Phone and Tablet`

   `O Android Things`

```
      Minimum SDK: API 24 Android 7.0 (Nougat)
   Add an activity: Android Things Empty Activity
   Activity Name: MainActivity
      O Generate a UI layout File
      Layout Name: activity_main
```

Deja el resto de opciones por defecto. Para tener acceso a las nuevas APIs es interesante que el target SDK sea API 27 Android 8.1 (Oreo) o superior.

2. Navega por los ficheros generados y observa las diferencias.
   - No se han añadido recursos para los iconos.
   - En AndroidManifest dentro de la sección `<application>` se ha añadido la siguiente librería y un filtro especial para la actividad:

```
<uses-library android:name="com.google.android.things" />
<activity android:name-".HomeActivity">
   <intent-filter>
      <action android:name-"android.intent.action.MAIN" />
      <category android:name-"android.intent.category.LAUNCHER" />
   </intent-filter>
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.IOT_LAUNCHER" />
      <category android:name="android.intent.category.DEFAULT" />
   </intent-filter>
</activity>
```

   - En build.gradle (Module: app) se ha añadido la siguiente dependencia:

```
dependencies {

   compileOnly 'com.google.android.things:androidthings:+'
}
```

3. Reemplaza el código de la actividad principal por:

```
public class HomeActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      PeripheralManager perifericos - PeripheralManager.getinstance();
      Log.d("HomeActivity", "GPIO: " + perifericos.getGpioList());
   }
}
```

Este código obtiene una referencia del manejador de periféricos y muestra un log con la lista de las entradas/salidas GPIO disponibles.

4. Ejecuta el proyecto. El resultado ha de ser similar al siguiente:

```
com.example.androidthings D/HomeActivity: GPIO: [BCM10, BCM11, BCM12,
BCM13, BCM14, BCM15, BCM16, BCM17, BCM18, BCM19, BCM2, BCM20, BCM21,
BCM22, BCM23, BCM24, BCM25, BCM26, BCM27, BCM3, BCM4, BCM5, BCM6, BCM7,
BCM8, BCM9]
```

Práctica: Mostrar información en el Layout de la actividad.

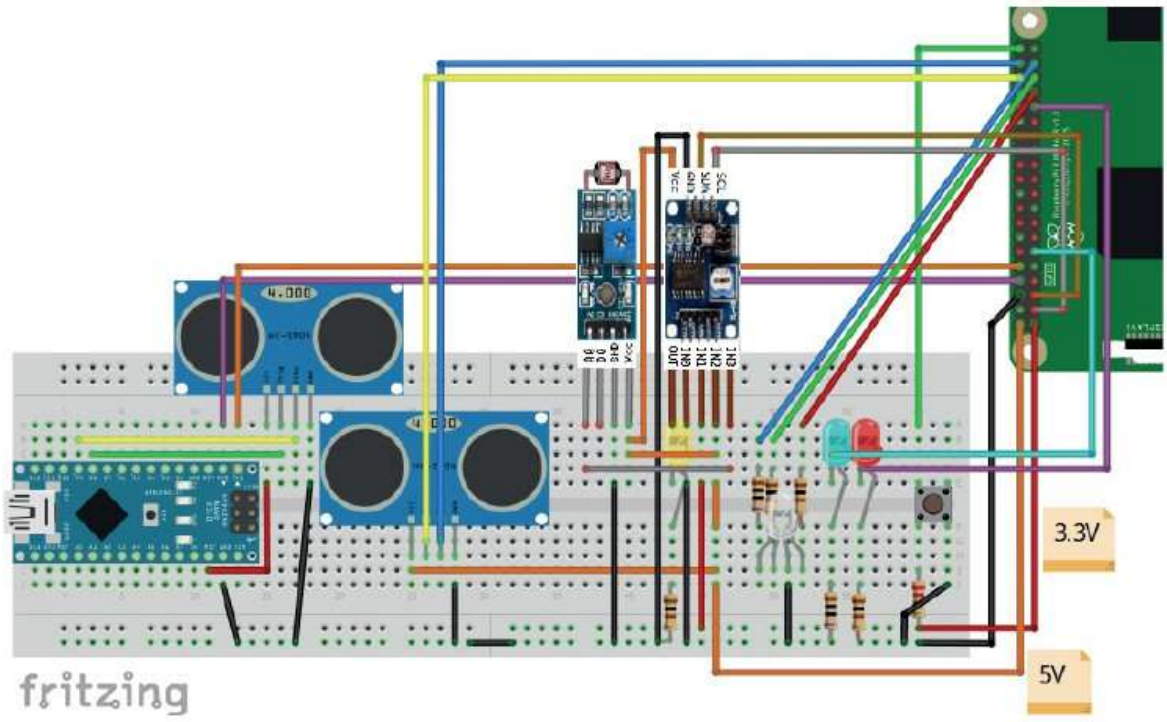
Utiliza el layout creado en la actividad para mostrar el resultado que hemos mostrado en el ejercicio anterior en el Logcat. Si conectaras un ratón, podrías incluso utilizar este Layout para realizar entradas en la aplicación.

4.4.5. Uso del laboratorio remoto

Lo ideal para la realización de estas unidades es que dispongas de una Raspberry Pi y los diferentes componentes para realizar los circuitos. No obstante, es posible que no tengas el material, o que te falte alguno de los componentes necesarios para realizar algún ejercicio concreto. En estos casos podrás utilizar el laboratorio remoto. Solo necesitas Android Studio, un navegador y una conexión a internet.

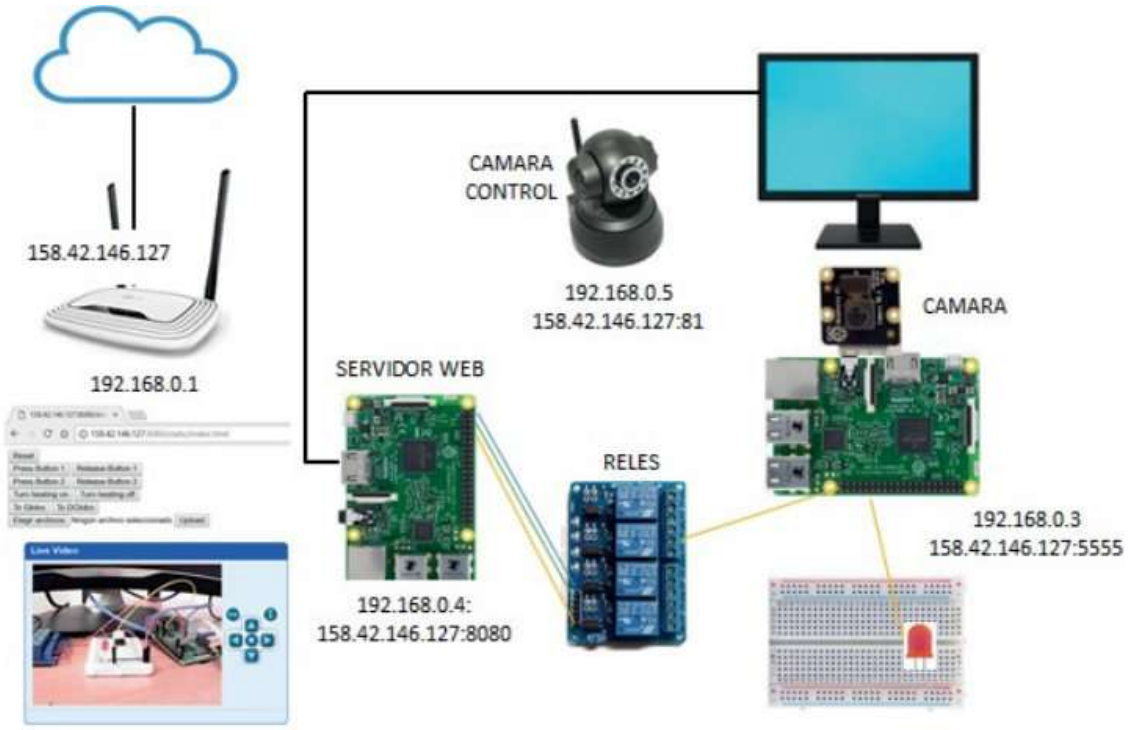
El primer paso ha de ser verificar si el laboratorio ya está siendo usado por otro alumno. Entra en Poliformat y selecciona la asignatura. En el menú de la izquierda selecciona la herramienta de Chat. Si nadie lo está usando escribe el mensaje "COMIENZO A USAR LAB. REMOTO". Es muy importante que cuando termines el trabajo escribas "DEJ O DE USAR LAB. REMORO". El periodo de uso es de una hora. Pasado este tiempo verifica en el Chat si algún compañero ha escrito un mensaje indicando que quiere usar el laboratorio. En caso de no ser así, escribe "CONTINÚO USANDO LAB. REMOTO" y podrás utilizarlo una hora más.

El siguiente esquema muestra los circuitos montados en el laboratorio. Corresponde a los puntos 1.6 y 1.7.



Nota: si no vas a usar el laboratorio remoto porque dispones de tu placa, te recomendamos que utilices una distribución parecida a la anterior. De esta forma, en lugar de montar y desmontar cada circuito, podrás montarlos todos en la misma placa.

El esquema de los diferentes elementos del laboratorio se muestra a continuación:



Los elementos son:

Servidor Web: Permite el acceso remoto para mostrar la imagen de la cámara y dispone de varios botones para realizar distintas acciones. Controla 4 relés que te permitirán resetear la Raspberry Pi y accionar botones de entrada.

Raspberry Pi con Android Things: Es el dispositivo que vas a programar.

Cámara de control: Nos ofrece una vista en tiempo real del laboratorio y permite verificar que la activación de los LEDs es correcta.

Los diferentes elementos se encuentran en una red que utiliza direcciones IP privadas en la red 192.168.0.0/24. Para acceder desde el exterior se utiliza la dirección IP pública 158.42.146.127. Cada elemento se mapea en un puerto distinto.

Para utilizar el laboratorio sigue los siguientes pasos:

- Con un navegador Web accede a http://158.42.146.127:8080.
- Verifica que tienes visión de los diferentes dispositivos y que están conectados. Puedes utilizar las flechas para cambiar el ángulo de la cámara.
- Si pulsas el botón de RESET la Raspberry Pi se reiniciará (no operativo).
- El botón UNINSTALL ALL desinstala todas las apps. Utilízalo si tienes problemas para instalar tus apps.

- Dispones de dos botones conectados a dos entradas GPIO de la Raspberry Pi.
- En un futuro se podrá poner en marcha un ventilador o una luz.
- Para realizar aplicaciones de visión artificial la Raspberry Pi dispone de una cámara que apunta a un monitor. Puedes cambiar la imagen mostrada en el monitor por medio de los botones <120>, <100>, <80> _
- Para interaccionar con el dispositivo, desde la línea de comando escribe:

```
adb connect 158.42.146.127:13
```

- Abre Android Studio y verifica que en la pestaña Logcat aparece el dispositivo Google Iot_rpi3.
- En esta ventana selecciona el icono Screen Capture para verificar la salida gráfica del dispositivo.

  Nota: en la mayoría de prácticas esta salida no es relevante.

- Si seleccionas la pestaña Android Profiler podrás realizar un análisis de CPU, memoria y red usadas por el dispositivo.
- También es posible utilizar la pestaña Debug para realizar una depuración de tus aplicaciones.
- Es el momento de probar las diferentes prácticas. Para verificar su correcto funcionamiento utiliza la cámara que se muestra en la página Web u observa las salidas mostradas en el Logcat.

4.5. Algunos conceptos de electrónica

A lo largo de este capítulo vamos a tener que montar pequeños circuitos electrónicos para conectar sensores y actuadores. Es posible que tus conocimientos de electrónica sean limitados, si es así, esta sección puede ayudarte a comprender algunos aspectos claves.

Puedes leer esta sección ahora, pero también puedes esperar a que los diferentes conceptos sean necesarios en algún ejercicio. En estos casos se te indicará el apartado a leer.

4.5.1. Voltaje y fuente de alimentación

Los circuitos electrónicos manipulan la información representándola por medio de señales eléctricas. Normalmente es el voltaje o tensión eléctrica la magnitud utilizada para este propósito.

Para conseguir este voltaje vamos a necesitar una fuente de alimentación, por ejemplo, la Raspberry Pi utiliza un cargador USB. Veamos algunos voltajes utilizados en casi todos los circuitos:

V_{IN}: Voltaje de la fuente de alimentación conectado a la placa. En la Raspberry Pi 9V.

V_{CC} (o **V_{DD}**): Voltaje de corriente continua regulado interno que alimenta los componentes de la placa. Los voltajes comunes son + 5V, + 3.3V y + 1.8V.

GND: (Ground): Punto de referencia utilizado para indicar los 0V de la placa. Todos los voltajes se miden respecto a este. Se conoce también como masa.

4.5.2. Señales analógicas y digitales

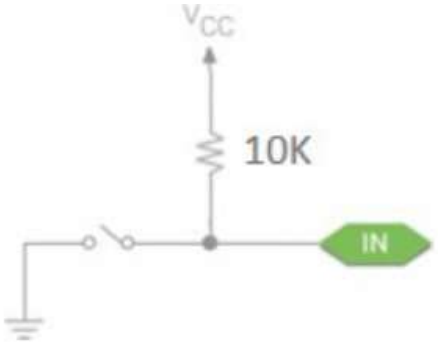
Según como puede variar el voltaje de la señal podemos diferenciar dos tipos de señales.

Analógicas: El voltaje de la señal es proporcional a la información que representa. Por ejemplo, si un sensor de temperatura nos da un voltaje entre 0V y 5V, que representa una temperatura entre 0º y 100º.

Digitales: El voltaje que puede tomar la señal se limita a un número finito de valores. Si estos valores son 2, se conocen como señales binarias. Cuando el valor está cercano a V_{CC}, se conoce como "1". Cuando el valor está cercano a GND, se conoce como "0". Para representar una magnitud como la temperatura se necesitan varias señales binarias, por ejemplo 8 bits, 00101100.

4.5.3. Resistencia pull-up / pull-down

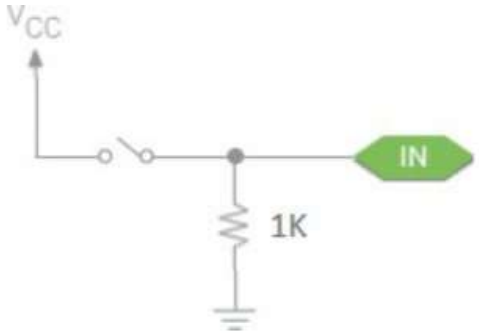
En el ejercicio anterior hemos conectado una resistencia entre la entrada GPIO y Vcc, tal y como se muestra en el esquema siguiente. Veamos por qué esta resistencia es necesaria. Si eliminaras esta resistencia y el pulsador estuviera abierto, tendríamos un cable desde la entrada, sin conectar a ningún sitio. Realmente no estaría conectado a masa, ni a Vcc, por lo que no podríamos decir que es ni 0 ni 1. A este tercer estado se le conoce como alta impedancia.



Puede ser peligroso dejar una entrada en alta impedancia. Un cable sin conectar, puede actuar como una antena. Si hay señales radio eléctricas cerca (producidas por un motor, antena ...) se podría inducir una corriente eléctrica en este cable, que podría engañar al GPIO provocando su activación. Para evitar dejar la entrada en alta impedancia, la conectamos a Vcc. Pero, cuando se cierre el pulsador la corriente pasará de masa a Vcc directamente, provocando un cortocircuito. Al

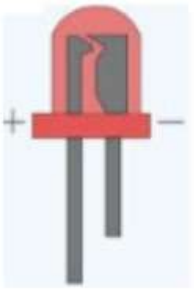
colocar esta resistencia, la corriente entra en la entrada, siguiendo el camino más fácil.

Si queremos que cuando el pulsador esté a abierto el valor de la entrada sea 0, y al pulsarlo pase a 1, utilizaríamos una resistencia de pull-down.



El valor habitual para una resistencia de pull-up es de 10K y de pull-down es de 1K.

4.5.4. LED y cálculo de resistencia de ajuste



Se trata de un diodo emisor de luz (Light Emitting Diode). Como todo diodo la corriente solo puede pasar del ánodo (pata más larga) al cátodo (pata más corta). Por lo tanto, para que emita luz es imprescindible que conectes el ánodo a tensión alta y el cátodo a masa. IMPORTANTE: nunca conectes un LED directamente con un pin a 3,3/5V y el otro a GND. La corriente sería demasiado alta y el LED se fundirá. Has de limitar el paso de corriente utilizando una resistencia.

Para calcular el valor de la resistencia podemos utilizar la ley de Ohm: $V = I \cdot R$. El valor máximo de corriente que puede pasar por un LED es de 20mA (un valor de 17mA es suficiente, si es menor brillará algo menos). También has de tener en cuenta que un led provoca una caída de voltaje entre 1,8V - 2,1V si es rojo, amarillo o verde y entre 3V - 3,8V si es azul, violeta o blanco.

Un ejemplo: tenemos un LED rojo, con una caída de 1,8V y queremos que pase una corriente de 17mA. Si la fuente de alimentación es de 9V la resistencia necesaria sería:

$$V = I \cdot R \Rightarrow R = V / I = (9V - 1,8V) / 17mA = 423,5\Omega \approx 470\ \Omega$$

Para una alimentación de 3,3V con LED rojo, con caída de 1,8V:

$$R = (3,3V - 1,8V) / 17mA = 88,2\Omega \approx 100\Omega$$

Para una alimentación de 3,3V con LED azul, con caída de 3,1V::

$$R = (3,3V - 3,1V) / 17mA = 11,8\Omega \approx 10\Omega$$

Más información sobre conceptos básicos de electrónica en: https://developer.android.com/things/hardware/hardware-101.html

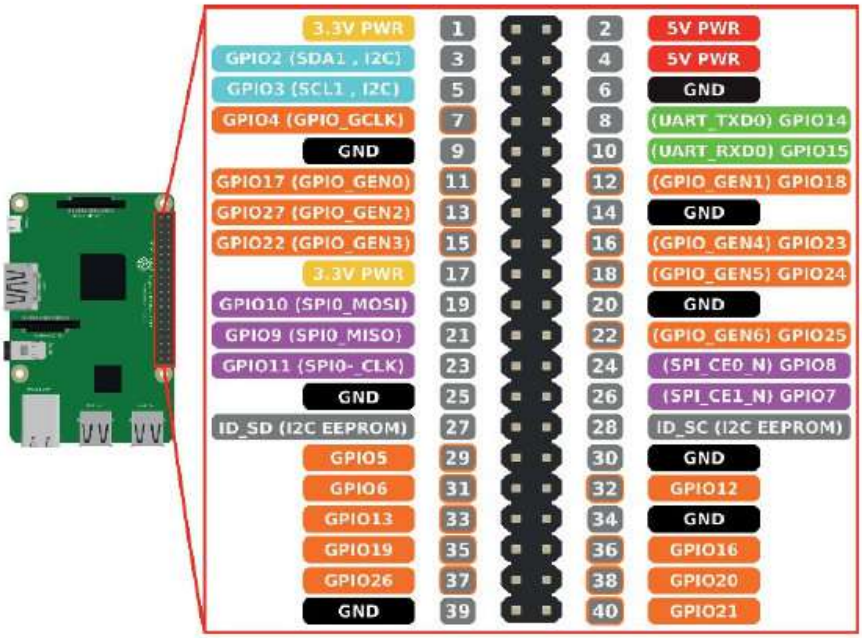
Preguntas de repaso: Conceptos de electrónica

4.6. Entradas / Salidas en Android Things

El potencial de Android Things está en la posibilidad de conectar hardware de entrada salida y poderlo controlar directamente desde nuestra la aplicación.

Las entradas salidas de propósito general se controlan por una serie de registros del chip BCM2837. Estos registros son mapeados en direcciones de memoria para su acceso. Cuando trabajas con un sistema operativo el acceso a estas posiciones está prohibido (excepto root) y tienes que utilizar los servicios ofrecidos por el sistema operativo.

Las conexiones a las entradas/salidas se realizan a través de 40 pines, tal y como se muestra a continuación:



GPIO Signal	Alternate Functions	
BCM2	I2C1 (SDA)	
BCM3	I2C1 (SCL)	
BCM7	SPI0 (SS1)	
BCM8	SPI0 (SS0)	
BCM9	SPI0 (MISO)	
BCM10	SPI0 (MOSI)	
BCM11	SPI0 (SCLK)	
BCM13	PWM1	
BCM14	UART0 (TXD)	MINIUART (TXD)
BCM15	UART0 (RXD)	MINIUART (RXD)
BCM18	I2S1 (BCLK)	PWM0
BCM19	I2S1 (LRCLK)	
BCM20	I2S1 (SDIN)	
BCM21	I2S1 (SDOUT)	

Una opción interesante puede ser utilizar un cable que permite llevar estos 40 pines a la placa de prototipos. De esta forma protegemos el desgaste de la placa original.

GND Ground (8 terminales) – Toma de tierra. SE utiliza como referencia para el resto de voltajes.

5V (2 terminales) – Conectados a la alimentación. Se utiliza un fusible para evitar problemas en caso de cortocircuito. Podemos consumir hasta 1A desde estos pines.

3,3V (2 terminales) – Podemos consumir hasta 1A.

GPIO (24 terminales) – Cada pin puede configurarse para trabajar como entrada o como salida. Como entrada hay que usar una tensión entre 0 y 3,3V. Las salidas no tienen que tener un consumo superior a 50mA.

PWM (2 terminales)

UART (2 terminales)

I²C (4 terminales)

SPI (5 terminales)

4.6.1. Conexiones GPIO

GPIO es el acrónimo de General Purpose Input/Output, en castellano Entrada/Salida de Propósito General. Corresponde a un pin de un chip que puede utilizarse para realizar una entrada o salida binaria, normalmente para interactuar con algún dispositivo exterior.

Si lo programamos como salida e indicamos nivel bajo en este pin tendremos 0V, y si indicamos nivel alto tendremos un voltaje de 3,3V. Hay que tener la pre-

caución de no conectar a esta salida un dispositivo con consumo superior a 16mA. El máximo que puede proporcionar todas las salidas simultáneamente es 50mA.

Si por el contrario lo programamos como entrada, leeremos un 0 si la tensión es menos de 0,8V y leeremos un 1 si la tensión está entre 1,3 y 3,3V. Nunca hay que superar este voltaje, dado que podríamos destruir el bloque GPIO.

Podemos tener hasta 24 entradas/salidas GPIO, pero algunos de los pines pueden compartir otras funciones como I2C, UART o PWM.

Hay que destacar que en Raspberry Pi no hay entradas analógicas como en otros controladores (como muchos compatibles Arduino). Si queremos leer un valor analógico tendremos que utilizar un conversor Analógico/Digital (A/D), como veremos más adelante.

Ejercicio: Entrada GPIO en Android Things

En este ejercicio aprenderás a comunicarte con el hardware del dispositivo. En concreto, leer la entrada de un simple botón.

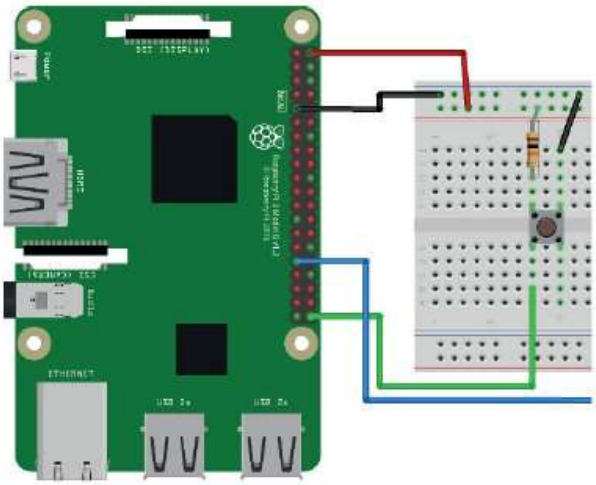
Material necesario:

- un pulsador:
  Si utilizas un pulsador de 4 pines, ten en cuenta que los pines marcados están interconectados. Al pulsar, los 4 pines estarían unidos.

- una resistencia de pull-up de 10 KΩ (color: café, negro, naranja). Se explica en el siguiente apartado.

- tablero de prototipos y cables.

1. Monta los sensores y actuadores sobre la placa de prototipos y conectarlos a la Raspberry Pi. El siguiente esquema te muestra cómo hacerlo:



- Conecta una conexión del botón al pin de entrada GPIO elegido (en este ejemplo BCM21).
- Conecta el otro extremo del botón a tierra y el mismo pin de entrada GPIO a + 3.3V a través de una resistencia pull-up de 10 KΩ.

2. Añade el siguiente código a `HomeActivity` para controlar la pulsación del botón:

```
private static final String BOTON_PIN - "BCM21"; // Puerto GPIO del botón
private Gpio botonGpio;

@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    PeripheralManager manager - PeripheralManager.getInstance();
    try {
       botonGpio - manager.openGpio(BOTON_PIN); // 1. Crea conecxión GPIO
       botonGpio.setDirection(Gpio.DIRECTION_IN);// 2. Es entrada
       botonGpio.setEdgeTriggerType(Gpio.EDGE_BOTH);
                    // 3. Habilita eventos de disparo por flanco de bajada
       botonGpio.registerGpioCallback(callback); // 4. Registra callback
      } catch (IOException e) {
         Log.e(TAG, "Error en PeripheralIO API", e);
      }
   }

   private GpioCallback callback - new GpioCallback() {
      @Override public boolean onGpioEdge(Gpio gpio) {
         try {
            Log.e(TAG,"cambio botón "+Boolean.toString(gpio.getValue()));
         } catch (IOException e) {
            e.printStackTrace();
         }
         return true; // 5. devolvemos true para mantener callback activo
      }
   };

   @Override protected void onDestroy() {
      super.onDestroy();
      if (botonGpio !- null) {   // 6. Cerramos recursos
         botonGpio.unregisterGpioCallback(callback);
         try {
            botonGpio.close();
         } catch (IOException e) {
            Log.e(TAG, "Error al cerrar botonGpio.", e);
         }
      }
   }
```

Para conseguir reaccionar ante el cambio de una entrada GPIO hemos seguido los siguientes pasos:

1. Usamos `PeripheralManager` para abrir una conexión con el puerto GPIO conectado al botón.
2. Configuramos el puerto como de entrada.

3. Configuramos qué transiciones de estado generarán eventos de llamada. Puede ser `EDGE_NONE` (no se dispara), `EDGE_RISING` (por flanco de subida), `EDGE_FALLING` (por flanco de bajada), `EDGE_BOTH` (por flanco de subida y bajada).
4. Registra un objeto `GpioCallback` para recibir los eventos de disparo.
5. Implementamos el objeto callback. El método `onGpioEdge(Gpio)` será llamado cuando ocurra el evento programado. El parámetro que se pasa es el puerto GPIO que causa el evento. Resulta sencillo averiguar su nombre de puerto ((`getName()`)) o su valor (`getValue()`). Importante, este método ha de devolver verdadero, si queremos continuar recibiendo futuros eventos de disparo.
6. Cuando la aplicación ya no necesita la conexión GPIO, cierra el recurso. En el ejemplo lo hacemos en `onDestroy()`.

3. Solicita el siguiente permiso en AndroidManifest:

```
<uses-permission
   android:name-"com.google.android.things.permission.USE_PERIPHERAL_IO"/>
```

4. Ejecuta el proyecto. Se producirás un error de compilación. Si abres el Logcat verás que se indica que el permiso no ha sido otorgado. Reinicia la Raspberry Pi para que el permiso sea otorgado.
5. Ejecuta de nuevo el proyecto. Cada vez que sueltes el botón ha de aparecer una línea en Logcat:

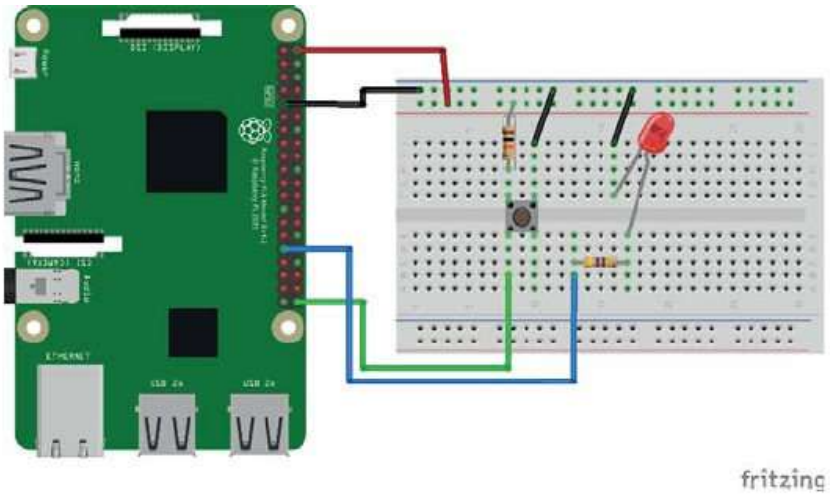
Ejercicio: Salida GPIO en Android Things

En este ejercicio aprenderás a comunicarte con una salida GPIO. En concreto un simple LED. Programaremos un `Handler` que se ejecute cada segundo para conseguir un efecto de parpadeo en el LED.

Material adicional:

- **un LED**: de color rojo.
- Resistencia de 100 Ω (color: marrón, negro, marrón) 

1. Monta los sensores y actuadores sobre la placa de prototipos y conectarlos a la Raspberry Pi. El siguiente esquema te muestra cómo hacerlo:



- Conecta el pin de salida GPIO elegido (BCM6) a un extremo de una resistencia en serie.
- Conecta el otro extremo de la resistencia al ánodo del LED (pata más larga).
- Conecta el cátodo del LED (pata más corta) a tierra.

2. Añade el siguiente código a `HomeActivity` para controlar el LED:

```
private static final int INTERVALO_LED - 1000; // intervalo parpadeo (ms)
private static final String LED_PIN - "BCM6"; // Puerto GPiO del LEO
private Handler handler - new Handler();       // Handler para el parpadeo
private Gpio ledGpio;

@Override protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   PeripheralManager manager - PeripheralManager.getInstance();

   try {
      ledGpio - manager.openGpio(LED_PIN); // 1. Crea conexión GPIO
      ledGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
                              // 2. Se indica que es de salida
      handler.post(runnable); // 3. Llamamos al handler
   } catch (IOException e) {
      Log.e(TAG, "Error en PeripheralIO API", e);
   }
}

private Runnable runnable - new Runnable() {
   @Override public void run() {
      try {
         ledGpio.setValue(!ledGpio.getValue()); // 4. Cambiamos valor LED
         handler.postDelayed(runnable, INTERVALO_LED);
            // 5. Programamos siguiente llamada dentro de INTERVALO_LED ms
      } catch (IOException e) {
         Log.e(TAG, "Error en PeripheralIO API", e);
      }
   }
};
```

Para conseguir reaccionar ante el cambio de una entrada GPIO hemos seguido los siguientes pasos:

1. Usamos `PeripheralManager` para abrir una conexión con el puerto GPIO conectado al LED.
2. Configuramos el puerto como de salida, indicando que inicialmente esté a nivel bajo.
3. Usando el `Handler`, hacemos una primera llamada al objeto `Runable`. El código de este objeto se ejecutará en un nuevo hilo.
4. El valor de salida es cambiado. Si estaba activo lo desactivamos y a la inversa.
5. Programamos una nueva llamada al `Runable` para dentro de un segundo. Con lo que conseguiremos un efecto de parpadeo.

3. Ejecuta el proyecto. Verifica que el LED parpadea.

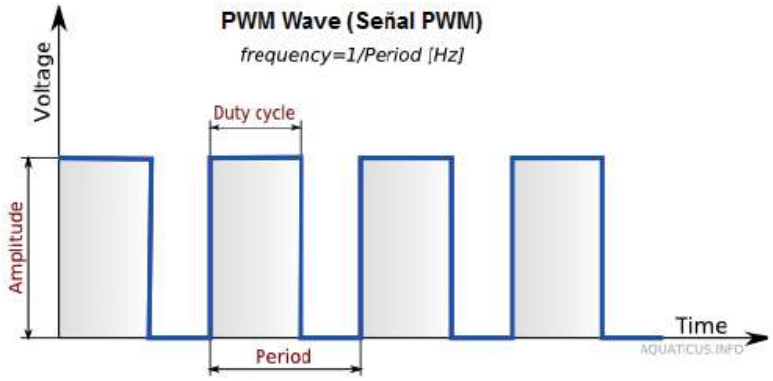
Práctica: Controlar parpadeo mediante el botón.

Tras realizar los dos ejercicios anteriores, modifica el proyecto para que el LED deje de parpadear cuando el botón está pulsado.

4.6.2. Salidas PWM

Como acabamos de ver, una salida GPIO solo puede tomar dos valores, 1 o 0. Imaginemos que queremos iluminar un LED, pero con una salida algo menor que 1 para que brille menos. Una solución podría ser utilizar una salida analógica para controlar el voltaje exacto que queremos aplicar. Dado que Raspberry Pi no dispone de salidas analógicas, tendríamos que utilizar un conversor D/A (digital analógico).

Existe otra solución que consiste en intercalar periodos de 1 con periodos de 0 en la salida. Si lo hacemos con una frecuencia lo suficientemente alta el efecto no será perceptible y obtendremos el resultado deseado.



La modulación de ancho de pulso o PWM[2] (Pulse Width Modulation) es un método común usado para aplicar una señal de control proporcional a un dispositivo usando una salida binaria. Por ejemplo, los servomotores usan PWM para determinar su velocidad de rotación. O las pantallas LCD ajustan su brillo en función del valor promedio de una señal PWM.

Una señal PWM es controlado por medio de dos parámetros. El periodo de la señal y el porcentaje del tiempo en el que la señal está a nivel alto.

La Raspberry Pi solo dispone de dos salidas PWM: Sus nombres son "PWM0" (pin 13, segunda función de "BCM18") y "PWM1" (pin 33, segunda función de "BCM13").

Ejercicio: Salida PWM en Android Things

En este ejercicio aprenderás a configurar una salida PWM. En concreto vamos a configurar el nivel de brillo de un LED.

Material específico necesario:

- un LED azul
- resistencia de 10 Ω (color: marrón, negro, negro)

1. Conecta el ánodo de un LED azul a la salida P18 (BCM18), conecta el cátodo a masa por medio de una resistencia de 10 Ω.
2. Añade el siguiente código a `HomeActivity` para controlar el LED por PWM:

```
private static final int PORCENTAGE_LED_PWM - 25; // % encendido
private static final String LED_PWM_PIN - "PWM0"; // Puerto del LEO
private Pwm ledPwm;

@Override protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   PeripheralManager manager - PeripheralManager.getInstance();

   try {
      ledPwm - manager.openPwm(LED_PWM_PIN); // 1. Crea conexión GPIO
      ledPwm.setPwmFrequencyHz(120);          // 2. Configuramos PWM
      ledPwm.setPwmDutyCycle(PORCENTAGE_LED_PWM);
      ledPwm.setEnabled(true);
   } catch (IOException e) {
      Log.e(TAG, "Error en al acceder a salida PWM", e);
   }
}

@Override protected void onDestroy() {
   super.onDestroy();
```

[2] https://developer.android.com/things/sdk/pio/pwm.html

```
    if (ledPwm !- null) {                        // 3. Cerramos recursos
        try {
            ledPwm.close();
            ledPwm - null;
        } catch (IOException e) {
            Log.e(TAG, "Error al cerrar PWM", e);
        }
    }
}
```

Para conseguir reaccionar ante el cambio de una entrada GPIO hemos seguido los siguientes pasos:

1. Usamos `PeripheralManager` para abrir una conexión con el puerto PWM conectado al LED.
2. Configuramos la frecuencia de la señal generada y el porcentaje en que estará activa la señal.
3. Cuando la aplicación ya no necesita la salida PWM, cierra el recurso. En el ejemplo lo hacemos en `onDestroy()`.

3. Ejecuta el proyecto. Verifica la intensidad LED. Aumenta el valor de `PORCENTAGE_LED_PWM` para aumentar su brillo.
4. Introduce un valor de `DutyCycle` de 25 y una frecuencia de 1. El LED se encenderá cada segundo, durante 250 mseg.

Práctica: Cambiar el brillo de un LED de forma periódica

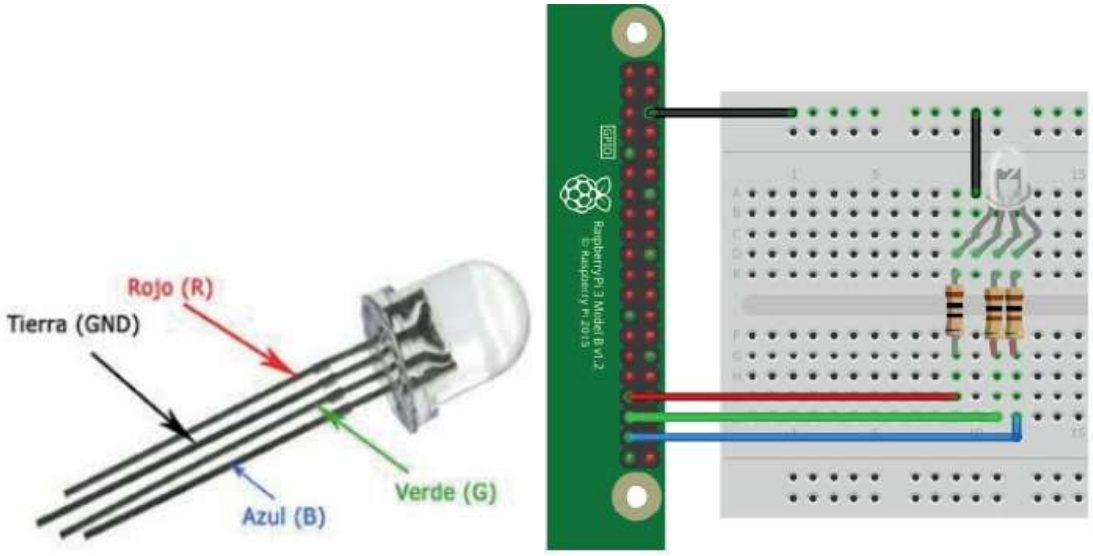
Aprovechando el temporizador de un segundo utilizado para el LED rojo, haz que el led azul modifique su brillo siguiendo la siguiente secuencia: 0%. 20%, 40%, 60%, 80%, 100%, 0%. 20%, 40%, ..

Práctica: Control de un LED RGB

En esta práctica vamos a realiza el control de un LED tricolor. Realmente se trata de tres LEDs dentro de la misma cápsula. Uno es rojo, otro verde y otro azul, de manera que, combinando la activación de cada uno, podemos conseguir distintas tonalidades. Lo ideal sería conectarlos a salidas PWM para controlar la intensidad de cada componente. Por desgracia, la Raspberry Pi solo tiene 2 salidas PWM. Por esta razón y para simplificar la programación vamos a utilizar salidas GPIO binarias. Al disponer de 3 bits para su control vamos a poder generar 7 colores diferentes, más el estado apagado.

Realiza un programa que haga pasar el LED por cada uno de estos 8 estados a intervalos de un segundo y lo repita de forma cíclica. Utiliza las salidas BCM13, BCM19 y BCM26. Puedes utilizar el esquema que se muestra a continuación. En principio las tres resistencias pueden ser de 100Ω. No obstante, dependiendo del

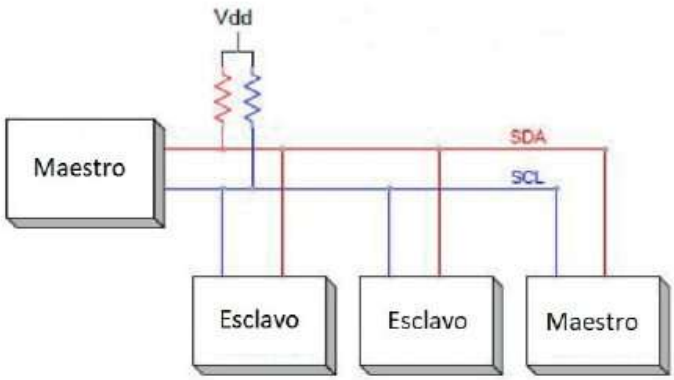
LED, es posible que tengas que ajustar algún valor. Por ejemplo, en el esquema se ha utilizado una resistencia de 10Ω para el componente rojo.



Nota: Si no dispones de un LER RGB, puedes realizar la práctica usando tres LEDs diferentes. Preferiblemente de colores rojo, verde y azul.

4.6.3. Bus series I²C

El bus I²C[3] permite interconectar varios dispositivos utilizando una modalidad de transmisión serie y síncrona. Su nombre viene del inglés, Inter-Integrated Circuit, lo que nos indica que ha sido diseñado para la interconexión de circuitos integrados. Fue desarrollado por Philips a principios de los 80, para reducir el número de pines en los chips. Como vamos a ver I²C permite controlar múltiples dispositivos utilizando solo 2 pines. A diferencia de otros interfaces de comunicación, como UART o USB que solo permite conectar dos dispositivos entre sí, I²C permite conectar varios dispositivos utilizando una topología en bus:



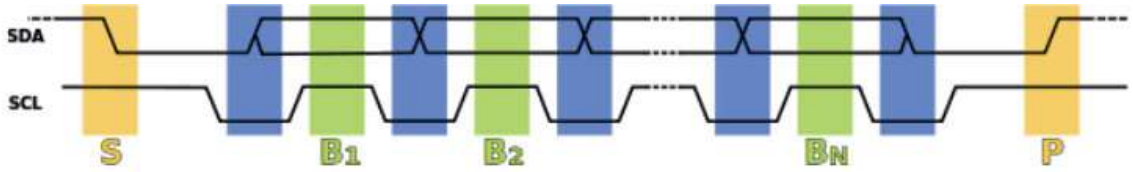
[3] https://en.wikipedia.org/wiki/I%C2%B2C

Cada dispositivo puede tomar el rol de maestro o el de esclavo. Una transferencia de datos siempre la inicia un maestro, y es un esclavo el que contesta. Lo más habitual es disponer de un solo maestro, pero existe un modo multímetro.

Se utiliza una topología en bus con dos líneas: SDA (Serial DAta) para transmitir los datos en serie y SCL (Serial CLock) con una señal que marca el comienzo de cada bit. Al utilizar esta señal de reloj se dice que es una transmisión síncrona. En el bus se añaden dos resistencias de pull-up de forma que, si ningún dispositivo está utilizando el bus, la línea estará a nivel alto, es decir a 1.

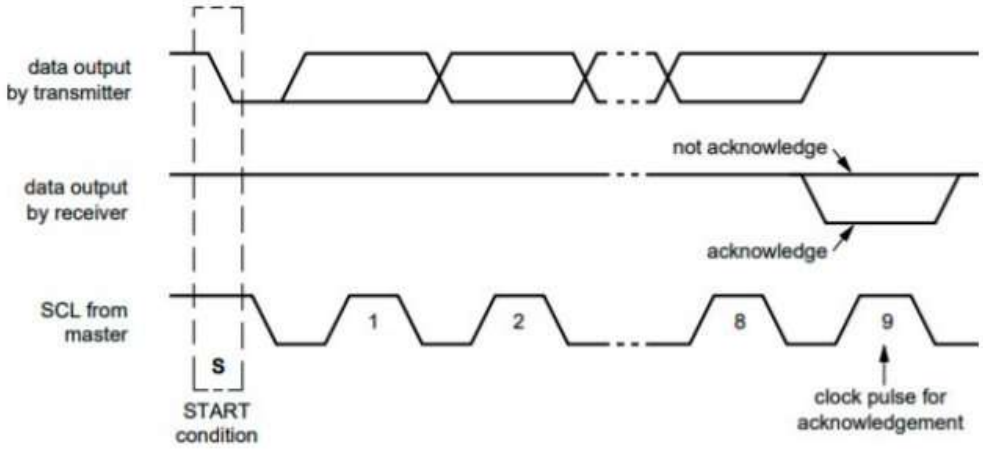
La señal de reloj siempre es introducida por el maestro, por lo que la velocidad de transmisión puede variar según marque el maestro. Existen varios valores preestablecidos, 0,1 0,4 1,0 3,4 y 5,0 Mbits/s. La unidad de datos transferida siempre es un byte.

El primer byte enviado por el maestro siempre es la dirección del esclavo en 7 bits, seguido de un 1 si quiere leer del esclavo o un 0 si quiere escribir. A este bit se le llama R/W. De los 7 bits de la dirección, los 4 más significativos son predeterminados por el fabricante y los tres últimos pueden ser fijados por tres (jumpers) que podemos configurar en el circuito. Esto nos permite introducir hasta 8 (2^3) circuitos iguales en el bus.



Si se realiza un flanco de bajada en la línea de datos, estando la línea de reloj a 1, significa que se inicia la transmisión (S: bit de arranque). Cuando el reloj baje, se pondrá el primer bit en la línea, cuando el reloj suba el bit podrá leerse. hasta que el reloj no vuelva a bajar el valor del bit ha de permanecer estable (B: bit de datos). Este proceso se repite hasta transmitir los N bits. Un blanco de subida, mientras el reloj está a 1, significa que termina la transmisión (P: bit de parada).

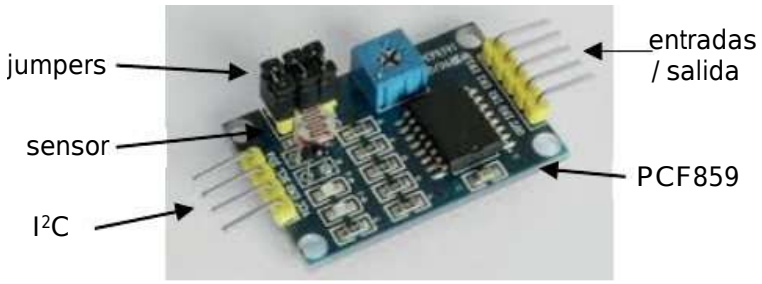
Tras la transmisión de cada byte se espera un bit de reconocimiento por parte del receptor (es decir, tras recibir 8 bits el receptor transmitirá un bit). Si el bit es 0, el reconocimiento es positivo y 1 en caso contrario.



Para indicar el final de una transmisión, el último byte leído es reconocido por el maestro como un no reconocimiento (Not ACKnowledge). Una transmisión es finalizada por la señal de parada.

4.6.3.1. El conversor A/D y D/A PCF8591

El PCF8591 es un chip que implementa un conversor de entradas analógicas en digitales de cuatro canales y un conversor digital/analógico de una salida. La resolución es de 8 bits. Este chip se conecta a un dispositivo maestro mediante el bus I^2C. La velocidad máxima de conversión viene dada por la velocidad máxima del bus I^2C.



El direccionamiento es de 7 bits, siendo los cuatro más significativos 1001 y los tres últimos configurables mediante jumpers. Si dejas los tres puestos estos tres bits serán 000. El primer byte enviado por el maestro enviando la dirección del dispositivo, seguido del bit de lectura/escritura.

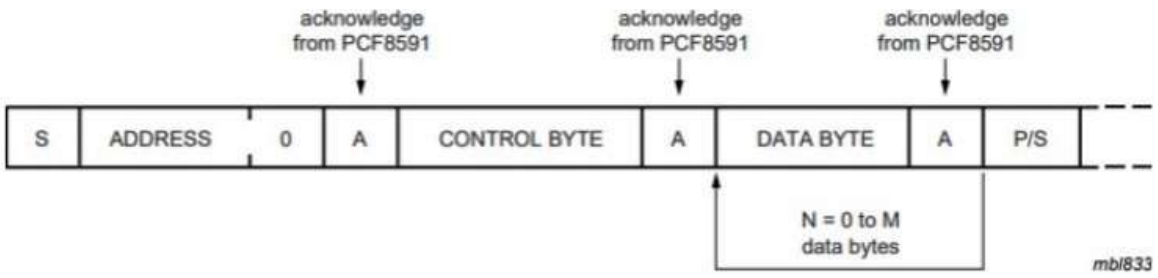
7	6	5	4	3	2	1	0
MSB							LSB
1	0	0	1	A2	A1	A0	R/$\overline{W}$

Conversión D/A

El maestro envía la dirección del PCF8591, seguida de un 0 en el bit R/W. El segundo byte que se envía es el byte de control, con los siguientes bits:

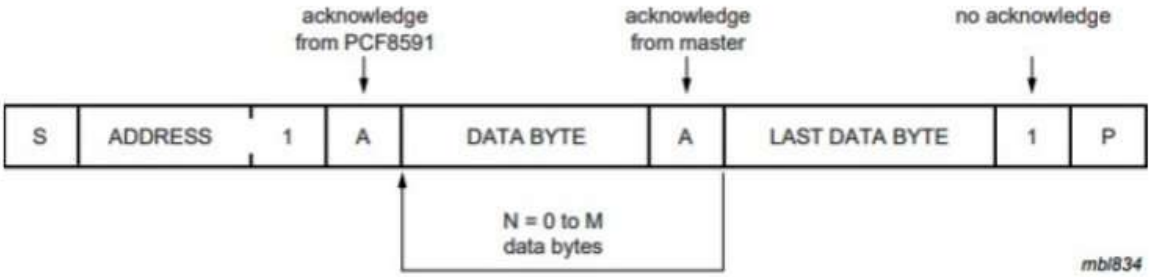
bit	valor	significado
7	0	valor fijo
6	0/1	1 activamos la salida analógica
5,4	00	4 entradas analógicas AIN0-AIN3 respecto a GND
	01	3 entradas analógicas respecto a AIN3
	10	AIN0 y AIN1 respecto a GND, AIN2 respecto a AIN3
	11	AIN0 respecto a AIN1, AIN2 respecto a AIN3
3	0	valor fijo
2	0/1	1 activamos auto incremento
1,0	00	canal 0
	01	canal 1
	10	canal 2
	11	canal 3

El tercer byte enviado es el valor digital a convertir. Podemos continuar la transmisión de bytes, de forma que el valor de salida será actualizado, hasta que el maestro introduzca un bit de parada.



Conversión A/D

El maestro envía la dirección del PCF8591, seguida de un 1 en el bit R/W. El PCF8591 envía la conversión digital de la entrada analógica. Si queremos utilizar un voltaje diferencial en las entradas tendremos que configurar el byte de control. El valor de la conversión será vuelto a mandar hasta que el maestro mande un reconocimiento negativo y un bit de parada. Si el modo activamos auto incremento está activo, se enviarán los diferentes canales uno tras otro, de forma cíclica.



Para una descripción más detallada consulta su data sheet[4]:

4.6.3.2. Utilización del bus I²C desde Android Things

Vemos como configurar el bus I²C desde el API de Android Things:

El primer paso va a ser conocer el nombre del puerto al que te quieres conectar. Para conocer los nombres de los dispositivos conectados utiliza el siguiente código:

```
PeripheralManager manager - PeripheralManager.getInstance();
List<String> listaDispositivos - manager.getI2cBusList();
```

En una Raspberry Pi 3, posiblemente obtendrás el nombre: `I2C1`. El siguiente código muestra un ejemplo de escritura y lectura, aplicado al chip PCF8591:

```
private static final byte ACTiVA SALiOA - 0x40; // e1ee ee ee
private static final byte AUTOiNCREMENTO - 0x04; // eeee e1 ee
private static final byte ENTRAOA e      - 0x00; // eeee ee ee
private static final byte ENTRAOA 1      - 0x01; // eeee ee e1
```

[4] https://www.nxp.com/docs/en/data-sheet/PCF8591.pdf

```
private static final byte ENTRAOA 2       - 0x02; // eeee ee 1e
private static final byte ENTRAOA 3       - 0x03; // eeee ee 11
private static final String IN_I2C_NOMBRE - "I2C1"; // Puerto de entrada
private static final int IN_I2C_DIRECCION - 0x48; // Dirección de entrada
private I2cDevice i2c;

try {
   i2c - manager.openI2cDevice(IN_I2C_NOM8RE, IN_I2C_DIRECCION);

   byte[] config - new byte[2];
   config[0] - (byte) ACTiVA SALiOA + ENTRAOA e; // byte de control
   config[1] - (byte) 0x80;                      // valor de salida (128/255)
   i2c.write(config, config.length);             // escribimos 2 bytes

   byte[] buffer - new byte[5];
   i2c.read(buffer, buffer.length);              // leemos 5 bytes
   String s -"";
   for (int i-0; i<buffer.length; i++) {
      s +- " byte "+i+": " + (buffer[i]&0xFF);
   }
   Log.d(TAG, s);                        // mostramos salida

   i2c.close();                          // cerramos i2c
   i2c - null;                           // liberamos memria
} catch (IOException e) {
   Log.e(TAG, "Error en al acceder a dispositivo I2C", e);
}
```

El código anterior resulta bastante sencillo de entender.

Este código hace una escritura/lectura en crudo (raw), adaptada al protocolo de comunicación definido para PCF8591 (ver sección anterior). Sin embargo otros dispositivos I2C, siguen un protocolo estándar conocido como System Management Bus (SMBus). En este protocolo se definen una serie de registros que pueden ser de lectura o escritura y se les asigna una dirección. Para más información sobre cómo utilizarlo consultar la documentación oficial[5].

Ejercicio: Una entrada/salida I2C en Android Things

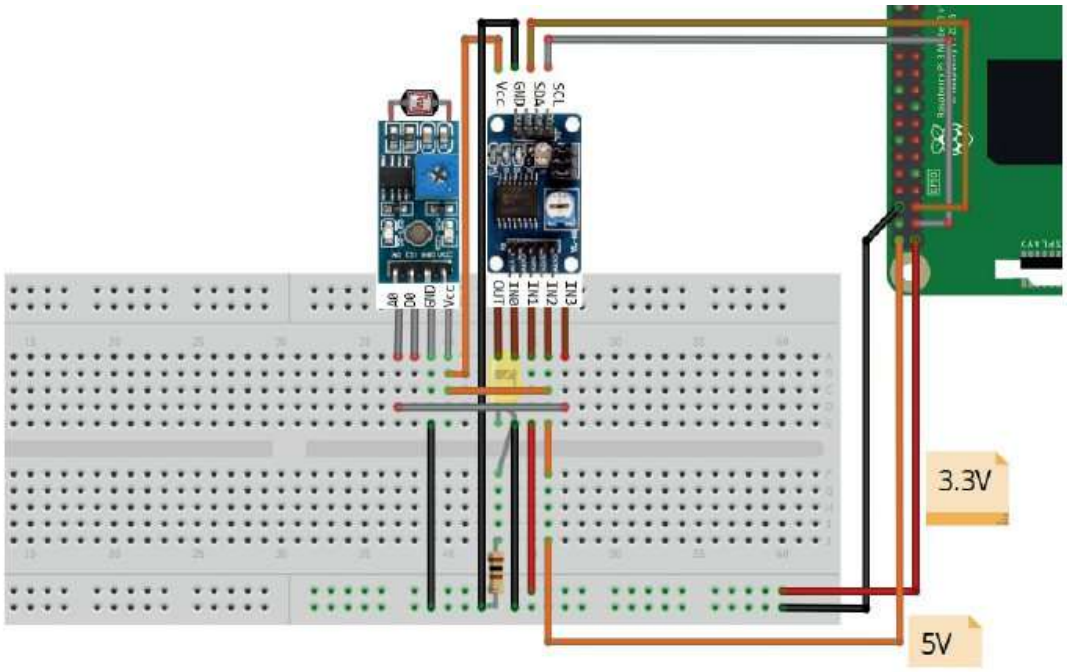
Material necesario:

- una placa con el chip PCF8591.
- un LED para mostrar la salida y una resistencia para ajustar su intensidad.

[5] https://developer.android.com/things/sdk/pio/i2c.html

- (opcional) potenciómetro o fotoresistencia para obtener una entrada analógica. Si has comprado el Kit de 16 sensores para Arduino puedes utilizar el módulo de sensor fotoresistor. Este sensor dispone de dos salidas: La analógica que nos ofrece un valor entre 0V y el nivel de alimentación inversamente proporcional al nivel d luz detectado. También tiene una salida digital que se activa al superar un nivel determinado. Este nivel puede ser configurado por medio de un pequeño potenciómetro.

1. Realiza las conexiones del bus I²C tal y como se muestra en la figura. En este caso lo alimentamos a 5V, aunque también se podría alimentar a 3,3V.



2. Crea un nuevo proyecto con los siguientes datos:

```
Application name: I2C
 Phone and Tablet
O Android Things
    Minimum SDK: API 27 Android 8.1 (Oreo)
Add an activity: Android Things Empty Activity
Activity Name: MainActivity
    Generate a UI layout File
```

Deja el resto de opciones por defecto.

3. Añade el siguiente código mostrado anteriormente en el ejemplo anterior, dentro de `onCreate()`.

4. Solicita el siguiente permiso en AndroidManifest:

```
<uses-permission
   android:name-"com.google.android.things.permission.USE_PERIPHERAL_IO"/>
```

5. Verifica que los tres jumpers de la placa PCF8591 coinciden con la dirección del bus I^2C.
6. Para verificar la salida analógica, puedes utilizar un voltímetro midiendo entre el pin OUT y GND. Como hemos indicado 255, el valor ha de ser de 5V. También puedes utilizar un LED (recuerda utilizar la resistencia adecuada). El LED ha de brillar a máxima potencia.
7. Modifica el valor de salida a 128, el valor leído ha de ser de 2,5V y el LED ha de perder brillo.
8. Modifica el valor de salida a 0, el valor leído ha de ser de 0V y el LED ha de apagarse.
9. Para verificar las entradas conecta IN0 a GND. Los cuatro valores leídos han de ser 0. Conecta IN0 a 3,3V. Los cuatro valores leídos han de ser 255*3,3/5. Conecta IN0 a 5V. Los cuatro valores leídos han de ser 255.
10. Si dispones de un potenciómetro o una fotorresistencia, utilízala como entrada.
11. Trata de configurar el modo autoincremental para que los valores leídos correspondan a entradas IN0 a IN3.

```
Lista de dispositivos I2C: [I2C1]
byte 0: 0 byte 1: 0 byte 2: 168 byte 3: 255 byte 4: 44
```

4.6.3.3. Utilización del chip PCF8591 por medio de un driver

Como acabamos de ver utilizar un chip puede ser bastante complejo. Este puede tener varios modos de funcionamiento o complicados protocolos de comunicación. Comprender las especificaciones que nos ofrecen en el data sheet, puede llevarnos mucho tiempo.

Afortunadamente este trabajo suele estar ya resuelto, de forma que no tenemos más que instalarnos el driver del dispositivo. Este se encargará de realizar las tareas complejas, ofreciéndonos un interfaz mucho más sencillo.

Por ejemplo, para el chip PCF8591 podemos encontrar el siguiente driver:

https://github.com/davemckelvie/things-drivers/tree/master/pcf8591

Ejercicio: Usar un driver para PCF8591

Nota: El material necesario y las conexiones coinciden con el ejercicio anterior.

1. Puedes crear un nuevo proyecto o partir del ejercicio anterior comentando el código.
2. Añade la siguiente dependencia en build.gradle (app):

```
dependencies {

    implementation 'nz.geek.android.things:things-drivers:1.8.0'
}
```

3. Añade los siguientes permisos en AndroidManifest:

```
<uses-permission
  android:name-"com.google.android.things.permission.MANAGE_INPUT_DRIVERS"/>
<uses-permission
   android:name-"com.google.android.things.permission.USE_PERIPHERAL_IO"/>
```

4. Declara las siguientes variables:

```
private I2cAdc adc;
private Handler handler - new Handler();
private Runnable runnable - new UpdateRunner();
```

5. En el método `onCreate()` añade:

```
I2cAdc.I2cAdcBuilder builder - I2cAdc.builder();
adc - builder.address(0).fourSingleEnded().withConversionRate(100).build();
adc.startConversions();
handler.post(runnable);
```

Se crea un builder que nos permitirá configurar el conversor AD. Se indica la dirección, pero solo la indicada en los jumpers (para 0 los tres ha de estar insertados). El modo de funcionamiento (lectura de los cuatro canales) y cada cuanto queremos una conversión.

Al llamar al método `startConversions()` se inicia la lectura de datos de forma continua. Para evitar fluctuaciones en los datos se utiliza la siguiente ecuación:

valor de salida = (salida anterior + nueva lectura) / 2

6. Añade el siguiente código:

```
private class UpdateRunner implements Runnable {
   @Override public void run() {
      String s -"";
      for (int i-0; i<-3; i++) {
         s +- " canal "+i+": "+adc.readChannel(i);
      }
      Log.d(TAG, s);
      handler.postDelayed(this, 1000);
   }
}
```

Se muestra los valores obtenidos en cada canal, en periodo de 1 segundo.

El driver que estamos utilizando tiene el inconveniente de no permitir la salida de datos utilizando el conversor D/A.

4.6.4. Entradas / salidas series SPI

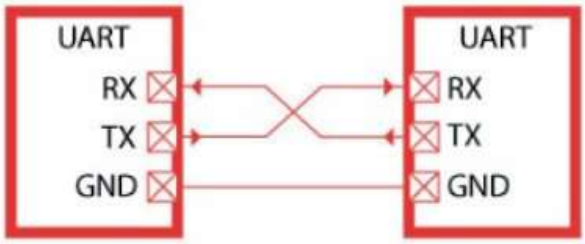
https://developer.android.com/things/sdk/pio/spi.html

http://android.geek.nz/measuring-analog-values-with-android-things/

4.6.5. Entradas / salidas series UART

UART (Universal Asynchronous Receiver-Transmitter) es una interfaz de comunicación serie con la que podemos conectar gran variedad de dispositivos como

GPS o pantallas LCD. Se conoce como un interfaz asíncrono dado que no utiliza una señal de reloj.



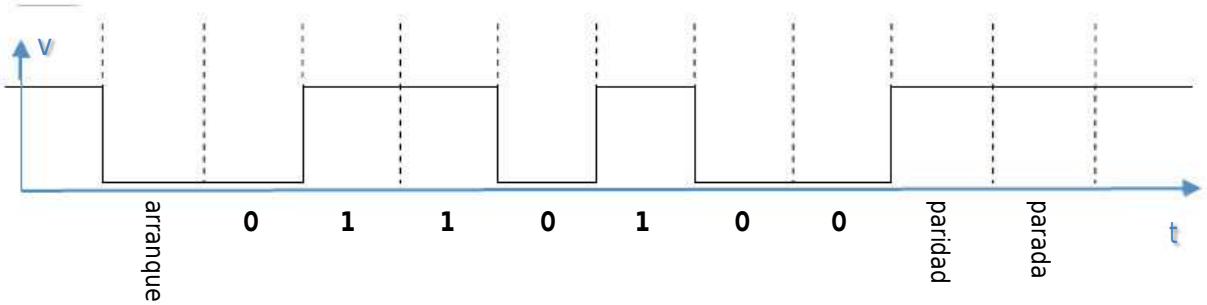
Como se muestra en el esquema anterior utiliza tres cables para la conexión. Los cables de transmisión (TX) y recepción han de estar cruzados entre los dos dispositivos. De este esquema podemos extraer que la transmisión es full dúplex (podemos transmitir y recibir a la vez) y que solo podemos interconectar dos dispositivos.

Uno de los mayores inconvenientes de la comunicación por UART es que emisor y receptor han de ponerse de acuerdo en varios parámetros de configuración. Para entender, estos parámetros veamos cómo se transmite la información.

La línea ha de permanecer a nivel alto mientras no se transmite información. Cada vez que se quiere transmitir una palabra, se transmite el **bit de arranque**, consiste en poner la línea a nivel bajo durante un periodo de tiempo, de la duración de un bit. Luego se transmitirían los **bits de datos**, pudiéndose configurar que estos sean entre 5 y 8. Cada bit se representa como un nivel alto para el "1" y nivel bajo para el "0", durante un periodo de bit. Los bits son transmitidos de derecha a izquierda, es decir primero el menos significativo. A continuación, se puede añadir un **bit de paridad**, que puede ser: par (la suma de "1" transmitidos es par), impar (la suma de "1" transmitidos es impar) o ninguna (no se añade este bit). Finalmente, se transmite el **bit de parada**. Consiste en dejar la línea a nivel alto un mínimo de uno o dos periodos de bit, según hayamos configurado.

Tras el bit de parada se podría transmitir una nueva palabra, comenzando por su bit de arranque. Pero si no disponemos de más datos la línea quedaría a nivel alto un tiempo indefinido.

En la siguiente gráfica se muestra la transmisión de la palabra 0010110, utilizando 7 bits de datos, 1 bit de paridad par y 1 bit de parada.



Si el receptor no tiene estos parámetros correctamente configurados, la transmisión no podrá realizarse. Otro aspecto de vital importancia es que emisor y receptor han de ponerse de acuerdo en el tiempo de bit. Cualquier pequeña diferencia entre los relojes de cada extremo, provocará un error en la transmisión. Este tiempo se indica en baudios (bits por segundo) siendo los valores más habituales 300, 1200, 4800, 9600, _ , 115200, _ ,1M, 2M baudios.

Vemos como configurar los puertos UART desde el API de Android Things:

El primer paso va a ser conocer el nombre del puerto al que te quieres conectar. Para conocer los nombres de los dispositivos conectados utiliza el siguiente código:

```
PeripheralManager manager - PeripheralManager.getInstance();
List<String> listaDispositivos - manager.getUartDeviceList();
```

En una Raspberry Pi 3, obtendremos los siguientes nombres: `MINIUART` y `UART0`. Nuevos dispositivos UART pueden ser conectados a través de un puerto USB. Por lo que la lista anterior puede aumentar si el usuario ha conectado nuevos dispositivos.

Para abrir el UART y configurarlo utilizaremos:

```
UartDevice uart;

try {
   uart - manager.openUartDevice("UART0");
   uart.setBaudrate(115200);
   uart.setDataSize(8);
   uart.setParity(UartDevice.PARITY_NONE);
   uart.setStopBits(1);
} catch (IOException e) {
   Log.w(TAG, "Error iniciando UART", e);
}
```

Para cerrarlo:

```
uart.close();
```

Para escribir bytes utilizaríamos:

```
byte[] buffer - {...};
int bytesEscritos - uart.write(buffer, buffer.length());
```

Si queremos escribir los caracteres de un String:

```
int bytesEscritos - uart.write(s.getBytes(), s.length());
```

Los datos recibidos son almacenados internamente en una memoria FIFO. Podremos extraer estos datos utilizando:

```
int bytesLeidos - uart.read(buffer, buffer.length);
```

Los datos son eliminados de la memoria de lectura y copiados a `buffer`. Pero si los datos superan al valor indicado en el segundo parámetro, solo se extraerán

estos bytes. Para asegurarnos que todos los datos son extraídos independientemente del tamaño de buffer, podemos utilizar este código.

```
byte[] buffer - new byte[16]; // Máximo de datos leidos cada vez 16
do {
   bytesLeidos - uart.read(buffer, buffer.length);
   // Procesamos un máximo de 16 bytes
} while(bytesLeidos >0);
```

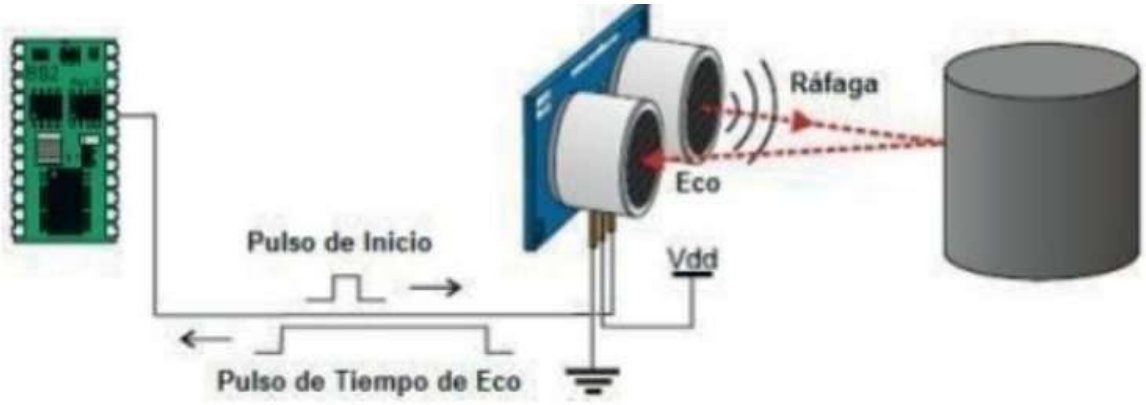
También podemos programar un escuchador de evento que se active cada vez que lleguen nuevos datos:

```
uart.registerUartDeviceCallback(
   new UartDeviceCallback() {
      @Override public boolean onUartDeviceDataAvailable(UartDevice uart){
         try {
             // Usar código anterior para leer bytes
         } catch (IOException e) {
             Log.w(TAG, "Error al leer de UART", e);
         }
         return true; // Continue listening for more interrupts
      }

      @Override public void onUartDeviceError(UartDevice uart, int error){
         Log.w(TAG, uart + ": Error " + error);
      }
   }
);
```

En el ejercicio Usar Arduino como esclavo a través de UART se muestra un ejemplo del uso de este API.

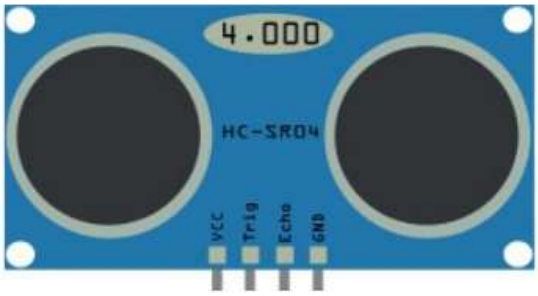
4.6.6. Medidor ultrasónico de distancia

En los siguientes apartados vamos a hacer uso del sensor ultrasónico HC-SR04 pasa medir distancias, por lo que comenzamos describiendo su funcionamiento. Se basa en el principio del sonar para determinar la distancia de un objeto, de forma similar a como lo hacen los murciélagos o un submarino. Es decir, se emite un pulso de ultrasonidos y se mide el tiempo del rebote. Conociendo la velocidad de propagación del sonido por el aire y sabiendo que el sonido tiene que ir y volver, calcular la distancia a partir de este tiempo es sencillo:



El sensor HC-SR04 tiene un rango de funcionamiento de 2 a 400 cm. Puede hacer medidas en un ángulo de 30º. Su funcionamiento no se ve afectado por la luz solar o el color del material, aunque los materiales como la tela, pueden ser difíciles de detectar.

Dispone de cuatro conectores: **VCC** que ha de conectarse a 5V y **GND** a masa. **Trig** es una entrada que ha de activarse un mínimo de 10µs para solicitar una medida. Entonces el sensor emitirá una ráfaga de 8 pulsos a 40KHz. Activará la salida **Echo** todo el tiempo que tarda en llegar el rebote de esta señal. Para obtener la distancia en cm, has de dividir el ancho del pulso obtenido en Echo entre 58.



Puedes encontrar el datasheet del sensor en el siguiente link[6].

Ejercicio: Medidor ultrasónico de distancia con Android Things

Nota: Este ejercicio está basado en el siguiente tutorial de Daniel Dallos.

Material necesario:

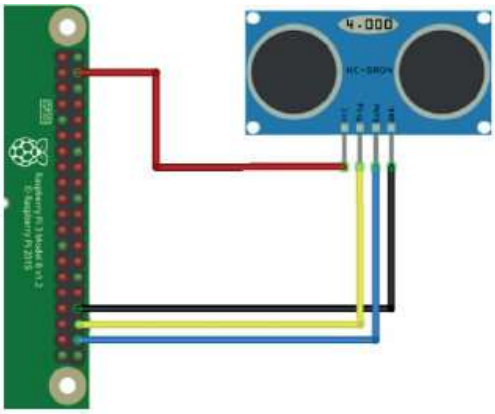
- Sensor de distancia HC-SR04.

1. Crea un nuevo proyecto para Android Things, sin layout, de forma similar a como se ha hecho en los ejercicios anteriores.

[6] https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE

Nota: Si lo prefieres puedes realizar este ejercicio dentro del mismo proyecto que el anterior, para facilitar la entrega de la tarea.

2. Conecta el sensor de distancia a la Raspberry Pi. El siguiente esquema te muestra cómo hacerlo:



3. Añade las siguientes variables:

```
private static final String ECHO_PIN_NAME - "BCM20";
private static final String TRIGGER_PIN_NAME - "BCM16"; //antes 21
private static final int INTERVALO_ENTRE_LECTURAS - 3000;
private Gpio mEcho;
private Gpio mTrigger;
```

4. Inicializa los puertos los puertos GPIO en `onCreate()`:
5. Añade el siguiente método:

```
int hazAlgo;

protected double leerDistancia() throws IOException, InterruptedException{
   mTrigger.setValue(false);
   Thread.sleep(0, 2000); // 2 mseg
   mTrigger.setValue(true);
   Thread.sleep(0, 10000); //1e msec
   mTrigger.setValue(false);
   while (mEcho.getValue() -- false) {
      hazAlgo - 0;
   }
   long tiempoIni - System.nanoTime();
   while (mEcho.getValue() -- true) {
      hazAlgo - 1;
   }
   long tiempoFin - System.nanoTime();
   long anchoPulso - tiempoFin - tiempoIni;
   double distancia - (anchoPulso / 1000.0) / 58.23; //cm
   Log.i(TAG, "distancia {Android Things): " + distancia + " cm");
   return distancia;
}
```

6. Añade el código necesario para que el método sea llamado cada 3 segundos.
7. Ejecuta el programa y haz una estimación de la precisión conseguida.
8. ¿En algún momento el programa deja de funcionar? ¿Cuál puede ser la causa? ¿Cómo podrías resolver el problema?

Preguntas de repaso: Entradas / Salidas en Android Things

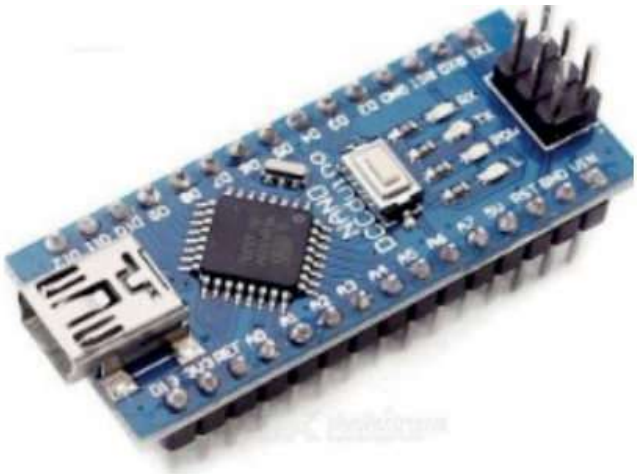
4.7. Usar un microcontrolador Arduino como esclavo

Si has realizado el ejercicio anterior, habrás comprobado que la Raspberry Pi no es el dispositivo más adecuado para interaccionar con los sensores. Esto es debido a varias causas:

- Al estar basado en un sistema operativo multiproceso, como Linux, no podemos asegurar que tenemos el procesador 100% disponible para nosotros.
- Las entradas GPIO en Android Things son especialmente lentas. Si quisiéramos generar una señal binaria activando y desactivando una salida GPIO, la frecuencia máxima que podríamos alcanzar es de 3KHz[7]. El problema es que muchos sensores necesitan una señal de más frecuencia, por lo que no vamos a poder controlarlos directamente desde Android Things. Podrías preguntarte, porque ocurre esto si los procesadores con los que estamos trabajando son muy rápidos. La razón estaría en que los desarrolladores de Google a la hora de desarrollar este API, han utilizado el método SYSFS, primado la seguridad frente a la velocidad.
- Otro inconveniente lo encontramos en la falta de entradas/salidas GPIO analógicas. Este problema se puede paliar utilizando un conversor A/D.
- La mayoría de sensores han sido diseñados para ser usados desde un microcontrolador. En sus especificaciones suelen incluirse fragmentos de código para descubrir su utilización en entorno Arduino. En algunos casos incluso se incluye una librería para controlarlos.

Todos estos inconvenientes no aparecen cuando estamos usando un microcontrolador. Al tener un único hilo de ejecución, no tenemos el problema de ser interrumpidos en mitad de un proceso. Pueden trabajar generando una señal binaria en una salida GPIO de hasta 8MHz. Disponen de entradas analógicas. Al tener un propósito muy concreto, normalmente controlar un sensor, suelen ser sencillos de programar, y por lo tanto más fiables. El consumo es muy bajo. Algunos incorporan mecanismos para entrar en suspensión cuando no son requeridos y volver a estar operativos en milisegundos. Esto permite que puedan tener una autonomía de varios años, alimentados con una pequeña batería. Finalmente, su precio es muy reducido. A continuación, se muestran dos microcontroladores, junto con su precio, que utilizaremos en los siguientes ejercicios:

[7] https://stackoverflow.com/questions/41727580



Arduino Nano (3,22 €)

ATTINY85 con USB (1,34 €)

A la izquierda tenemos un Arduino Nano que incorpora un microcontrolador ATmega328 (mismo que utiliza Arduino UNO). Este chip tiene 14 entradas/salidas (6 son PWM y 6 son conversores A/D con 10 bits de resolución), 1 KB de EEPROM, 2 KB de SRAM, 32 KB de memoria flash, frecuencia 16Mhz. El microcontrolador se monta sobre una placa que puede ser fácilmente insertada sobre una placa de prototipos. Además, incorpora un USB que facilita su programación, o como veremos más adelante la conexión con la Raspberry Pi.

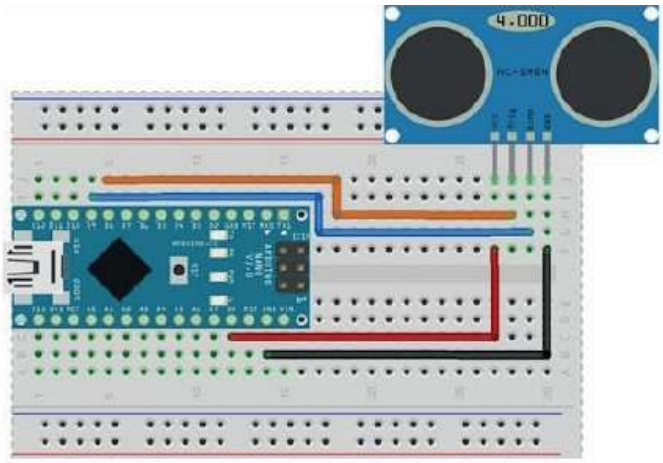
A la derecha tenemos el microcontrolador ATTINY85 montado en una placa que nos facilita la conexión directa a un procesador con entrada USB. Por ejemplo, en la Raspberry Pi podríamos utilizar una de las 4 disponibles. También podemos conectarlo a un USB de nuestro ordenador para programarlo utilizando la plataforma Arduino. En tamaño del microcontrolador es muy pequeño, 0,9 x 0,7 mm y tiene solo 8 pines (chip de la izquierda). Sus prestaciones también son reducidas. 5 entradas/salidas (2 son PWM), 512 bytes de EEPROM, 512 bytes de SRAM, frecuencia 20Mhz.

Ejercicio: Medidor ultrasónico de distancia con Arduino

Material necesario:

- Microcontrolador compatible Arduino (hemos utilizado Arduino Nano).
- Cable con conector de USB a Mini USB, conector macho tipo "A" a macho mini USB tipo “B” 5 pines.
- Sensor de distancia HC-SR04.

1. Monta el sensor de distancia y el Arduino Nano sobre la placa de prototipos, tal y como se muestra a continuación:



Ha de conectar GND, 5V, D8 y D9 en Arduino con GND, VCC, Trig y Echo en el sensor.

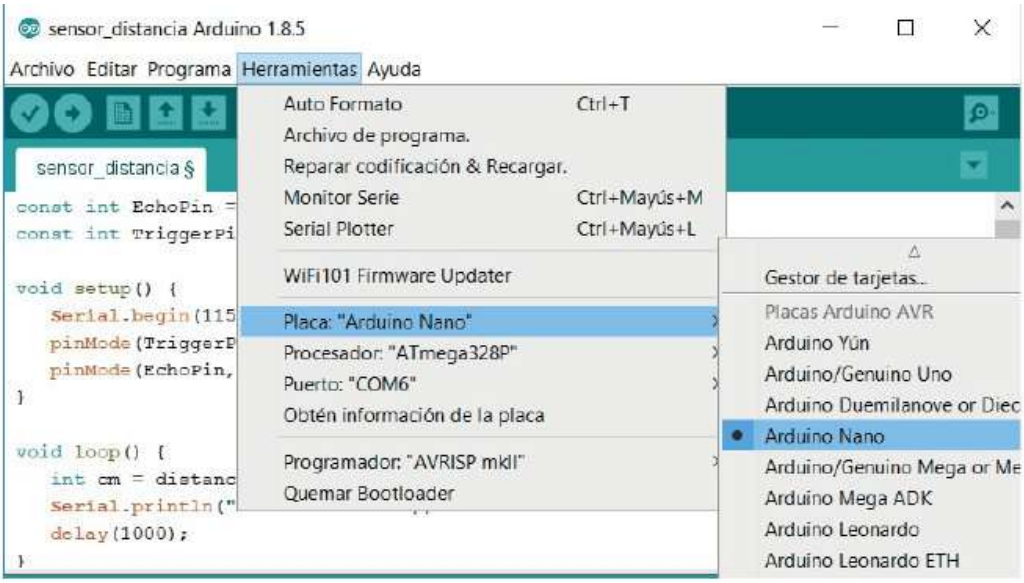
2. Accede a la página https://www.arduino.cc/en/Main/Software y busca la sección Download the Arduino IDE. Selecciona la descarga adecuada según tu sistema operativo e instálalo.
3. Abre el entorno y copia el siguiente código en el editor:

```
const int EchoPin - 9;
const int TriggerPin - 8;

void setup() {
   Serial.begin(115200);
   pinMode(TriggerPin, OUTPUT);
   pinMode(EchoPin, INPUT);
}

void loop() {
   Serial.print("Distancia: ");
   Serial.println(distancia(TriggerPin, EchoPin));
   delay(1000);
}

int distancia(int TriggerPin, int EchoPin) {
   long duracion, distanciaCm;
   digitalWrite(TriggerPin, LOW); //nos aseguramos señal baja al principio
   delayMicroseconds(4);
   digitalWrite(TriggerPin, HIGH); //generamos pulso de 1eus
   delayMicroseconds(10);
   digitalWrite(TriggerPin, LOW);
   duracion - pulseIn(EchoPin, HIGH); //medimos el tiempo del pulso
   distanciaCm - duracion * 10 / 292 / 2;   //convertimos a distancia
   return distanciaCm;
}
```

El lenguaje de programación que utiliza Arduino está basado en C/C++. Siempre han de existir dos métodos. `setup()` que es ejecutado al arrancar el dispositivo para realizar las configuraciones iniciales. En el código se inicializa el puerto serie, con una determinada velocidad, para utilizarlo como salida de consola. Además, se inicializa los dos puertos GPIO que vamos a utilizar. El

método `loop()` será ejecutado de forma continua, una tras otra. En el código, mostramos por el puerto serie "Distancia: " y luego mostramos el resultado obtenido por la función distancia. Finalmente se espera un segundo y se repite el proceso. El funcionamiento de la función `distancia()` resulta sencillo de entender, una vez leído el apartado anterior.

4. Conecta un cable USB a tu ordenador.
5. Selecciona la opción Herramientas/Puerto/COMX. Donde X es el número de puerto donde has conectado el Arduino. Si hay varios donde escoger, desconecta el cable y mira qué número de puerto desaparece.
6. Selecciona la opción Herramientas/Placa/Arduino Nano. O la placa que estés utilizando:



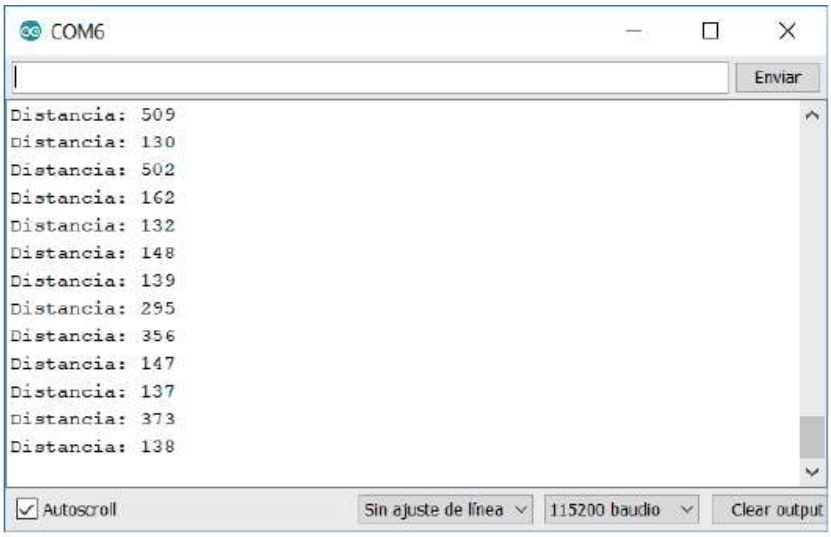
7. Pulsa el botón Subir.

8. Si todo es correcto en la parte inferior ha de aparecer:

9. Pulsa el botón Monitor Serie:

Se abrirá una ventana que me permite comunicarme a través del puerto serie con Arduino.

10. Selecciona en el desplegable de la parte inferior 115200 baudios. La salida se actualizará cada segundo mostrando la distancia entre el sensor y el objeto más cercano:



Una vez descrito como podemos controlar el sensor de distancia con Arduino vamos a ver podemos utilizarlo desde Android Things para acceder al sensor a través del microcontrolador.

Ejercicio: Procesar comandos en Arduino por el puerto serie

1. Modifica el método `loop()` del ejercicio anterior por el siguiente

```
void loop() {
  if (Serial.available() > 0) {
    char command - (char) Serial.read();
    switch (command) {
     case 'H':
       Serial.println("Hola Mundo");
       break;
     case 'D':
       Serial.println(distancia(TriggerPin, EchoPin));
       break;
    }
  }
  Serial.flush();
}
```

   Este código implementa un sistema de comandos con el que podremos solicitar al microcontrolador la lectura de diferentes sensores, enviando un carácter determinado.

2. Sube el nuevo código al dispositivo y abre el Monitor Serie.
3. Escribe el carácter `H` y pulsa en Enviar. Ha de aparecer el texto `Hola mundo`.
4. Escribe el carácter `D` y pulsa en Enviar. Se mostrará el valor del sensor.

Ejercicio: Usar Arduino como esclavo a través de UART

1. Añade la siguiente clase en el proyecto:

```
public class ArduinoUart {

   private UartDevice uart;

   public ArduinoUart(String nombre, int baudios) {
      try {
         uart - PeripheralManager.getinstance().openUartDevice(nombre);
         uart.setBaudrate(baudios);
         uart.setDataSize(8);
         uart.setParity(UartDevice.PARITY_NONE);
         uart.setStopBits(1);
      } catch (IOException e) {
         Log.w(TAG, "Error iniciando UART", e);
      }
   }

   public void escribir(String s) {
      try {
         int escritos - uart.write(s.getBytes(), s.length());
         Log.d(TAG, escritos+" bytes escritos en UART");
      } catch (IOException e) {
         Log.w(TAG, "Error al escribir en UART", e);
      }
   }

   public String leer() {
      String s - "";
      int len;
      final int maxCount - 8; // Máximo de datos leidos cada vez
      byte[] buffer - new byte[maxCount];
      try {
         do {
          len - uart.read(buffer, buffer.length);
            for (int i-0; i<len; i++) {
               s +- (char)buffer[i];
            }
         } while(len>0);
      } catch (IOException e) {
         Log.w(TAG, "Error al leer de UART", e);
      }
      return s;
   }

   public void cerrar() {
      if (uart !- null) {
         try {
            uart.close();
```

```
            uart - null;
         } catch (IOException e) {
            Log.w(TAG, "Error cerrando UART", e);
         }
      }
   }

   static public List<String> disponibles() {
      return PeripheralManager.getinstance().getUartDeviceList();
   }
}
```

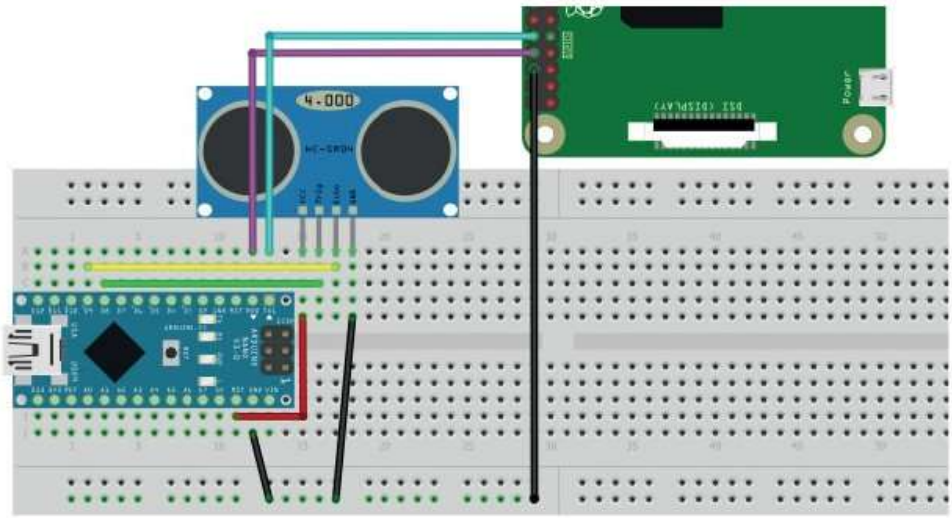
2. Añade la siguiente código en el método `onCreate()`:

```
Log.i(TAG, "Lista de UART disponibles: " + ArduinoUart.disponibles());
ArduinoUart uart - new ArduinoUart("UART0", 115200);
Log.d(TAG, "Mandado a Arduino: H");
uart.escribir("H");
try {
   Thread.sleep(5000);
} catch (InterruptedException e) {
   Log.w(TAG, "Error en sleep{)", e);
}
String s - uart.leer();
Log.d(TAG, "Recibido de Arduino: "+s);
```

3. Puentea los pines TX (8) y RX (10). De esta forma todo lo transmitido será directamente recibido.
4. Ejecuta el programa. El resultado ha de ser similar a:

```
 : Lista de UART disponibles: [MINIUART, UART0]
 : Mandado a Arduino: H
 : 1 bytes escritos en UART
 : Recibido de Arduino: H
```

5. Conecta los pines del puerto UART entre la Raspberry Pi y Arduino, teniendo en cuenta que los bits TX y RX han de estar cruzados. Conecta también GND:



6. Ejecuta el programa. En el log ha de aparecer, la respuesta de Arduino frente a “H”, que ha de ser “Hola Mundo”:
7. Envía el comando “D” y comprueba que se recibe la distancia del sensor.

Preguntas de repaso: Usar Microcontrolador (Arduino) como esclavo

4.8. Controladores de usuario

Como vimos en una sección anterior, interactuar con el chip PCF8591, puede ser un trabajo muy complejo. También vimos cómo utilizar un driver o controlador podría simplificar mucho este trabajo. No solo tenemos esta ventaja, además es muy posible que el driver realice las operaciones de una forma más correcta y está ampliamente testeado, de lo que podríamos hacer nosotros.

Android Things introduce un marco de desarrollo que nos facilita la utilización y creación de controladores, a través de `UserDriverManager`.

4.8.1. Utilizar controladores

Si quieres utilizar un controlador ya desarrollado sigue los siguientes pasos:

1. Busca el driver para tu dispositivo. Por ejemplo, en: https://github.com/androidthings/drivers-samples
2. Añade la dependencia en el build.gradle:

```
dependencies {
    ...
    compile 'com.google.android.things.contrib:driver-button:1.0'
}
```

3. Pide el permiso para usar el tipo de driver en el manifiesto:

```
<uses-permission android:name-
            "com.google.android.things.permission.MANAGE_INPUT_DRIVERS" />
```

Es muy posible que el driver requiera algún permiso adicional, como el de entradas y salidas.

4. Inicializa la clase del controlador con los parámetros adecuados.

```
import com.google.android.things.contrib.driver.button.Button;
import com.google.android.things.contrib.driver.button.ButtonInputDriver;

public class ButtonActivity extends Activity {
   private static final String NOM8RE_DE_PIN - ...;
   private ButtonInputDriver driverBoton;

   @Override protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

```
        try {
            driverBoton - new ButtonInputDriver(NOM8RE_DE_PIN,
            Button.LogicState.PRESSED_WHEN_LOW, KeyEvent.KEYCODE_SPACE);
        } catch (IOException e) {
            Log.e(TAG, "Error configurando pin GPIO", e);
        }
    }
```

5. Recuerda cerrar el recurso cuando dejes de necesitarlo.

```
    @Override protected void onDestroy(){
        super.onDestroy();
        if (driverBoton !- null) {
            try {
                driverBoton.close();
            } catch (IOException e) {
                Log.e(TAG, "Error cerrando driver de botón ", e);
            }
        }
    }
```

6. Activarlo y desactivarlo cuando se necesario:

```
    @Override protected void onStart() {
        super.onStart();
        driverBoton.register();
    }
    @Override protected void onStop() {
        super.onStop();
        driverBoton.unregister();
    }
```

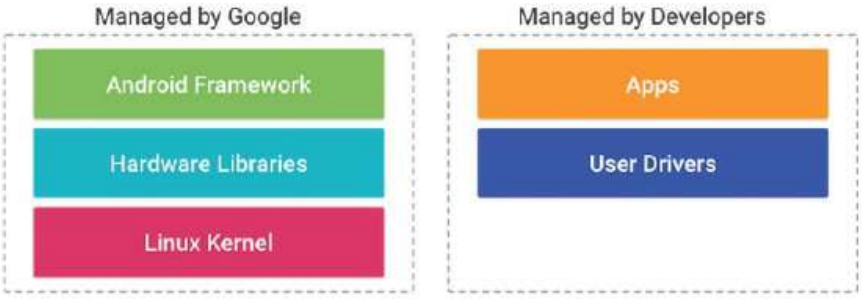
7. El driver podrá interaccionar con el sistema estándar Android. Por ejemplo, modificando la localización del dispositivo, o como en el ejemplo mostrado introducir teclas en la entrada estándar.

```
    @Override public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode -- KeyEvent.KEYCODE_SPACE) {
            // .
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
    @Override public boolean onKeyUp(int keyCode, KeyEvent event) {
        if (keyCode -- KeyEvent.KEYCODE_SPACE) {
            // .
            return true;
        }
        return super.onKeyUp(keyCode, event);
    }
}
```

4.8.2. Escribir controladores de usuario

Para permitir a los desarrolladores de aplicaciones que puedan registrar nuevos controladores de dispositivos, Android Things introduce el concepto de un

controlador de usuario (user driver). Los controladores de usuario son componentes registrados desde las aplicaciones que amplían los servicios existentes del framework Android. Permiten que cualquier aplicación inserte eventos de hardware en el framework que otras aplicaciones puedan procesar utilizando las API estándar de Android.



Permitir que las aplicaciones interaccionen de forma directa con el framework puede hacer al sistema inestable e inseguro. También presenta otros beneficios muy necesarios en el marco de IoT, donde los tipos de dispositivos y periféricos son inmensos. Esta flexibilidad nos va a permitir que nuestro código sea portable, reutilizable y fácil de integrar.

Podemos diferenciar varios tipos de controladores:

Dispositivos de Interfaz Humana (HID): obtienen información proporcionada por el usuario. Algunos ejemplos son teclados, almohadillas táctiles y mando. Solicitar permiso: `MANAGE_INPUT_DRIVERS`. Más info

Ubicación: proporcionan información de ubicación física del dispositivo. Pueden basarse en diferentes tecnologías como GPS, WiFi, Bluetooth, . Solicitar permiso: `MANAGE_GNSS_DRIVERS`. Más info

Sensores: miden las condiciones del entorno físico. Algunos controladores pueden fusionar información de varios sensores físicos en un solo sensor virtual. Esto es particularmente común con sensores de dirección, que combinan acelerómetros, giroscopios y campo magnético. Solicitar permiso: `MANAGE_SENSOR_DRIVERS`. Más info

LowPAN: las redes inalámbricas de área personal de baja potencia permiten que los dispositivos se conecten e intercambien datos a través de redes en malla. Los controladores LoWPAN te permiten integrar un módulo de radio externo al dispositivo para que sigua las especificaciones de la API LoWPAN. Solicitar permiso: `MANAGE_LOWPAN_INTERFACE`. Más info

4.8.2.1. Escribir controladores de Interfaz de Usuario

Estos controladores van a tener el privilegio de insertar eventos de pulsación de teclas o eventos de movimiento sobre superficie táctil[8]. Requieren de la solicitud del permiso:

[8] http://source.android.com/devices/input/overview.html

```
<uses-permission
   android:name-"com.google.android.things.permission.MANAGE_INPUT_DRIVERS" />
```

Veamos un ejemplo, que permite introducir la tecla Escape:

1. Crea una instancia del driver utilizando `InputDriver.Builder`.
2. Registra el driver con `UserDriverManager`.

```
public class ButtonDriverService extends Service {
   private static final String DRIVER_NAME - "EscapeButton";
   private static final int KEY_CODE - KeyEvent.KEYCODE_ESCAPE;
   private InputDriver driver;

   @Override public void onCreate() {
      super.onCreate();
      driver - new InputDriver.Builder() // Creamos instancia de driver
            .setName(DRIVER_NAME)
            .setSupportedKeys(new int[] {KEY_CODE})
            .build();
      UserDriverManager manager - UserDriverManager.getInstance();
      manager.registerInputDriver(driver);
   }

   @Override public IBinder onBind(Intent intent) {
      return null;
   }
}
```

3. Cuando ocurra algún evento hardware por ejemplo se pulsa un botón, llama al siguiente método:

```
private void lanzarEvento(boolean pressed) {
   InputDriverEvent event - new InputDriverEvent();
   event.setKeyPressed(KEY_CODE, pressed);
   driver.emit(event);
}
```

4. Desactiva el registro del driver en `onDestroy()`:

```
@Override public void onDestroy() {
   super.onDestroy();
   UserDriverManager.getInstance().unregisterInputDriver(driver);
}
```

Puedes consultar la implementación del controlador `ButtonInputDriver` que hemos utilizado en la sección anterior en: https://github.com/androidthings/contrib-drivers/tree/master/button

Si en lugar de teclas quieres introducir eventos de movimiento, emulando que un usuario deslice un dedo sobre una pantalla táctil. En la creación del driver re-emplaza las siguientes líneas:

```
driver - new InputDriver.Builder()
      .setName(DRIVER_NAME)
      .setAxisConfiguration{MotionEvent.AXIS_X, 0, 255, 0, 0)
      .setAxisConfiguration{MotionEvent.AXIS_Y, 0, 255, 0, 0)
      .build();
```

Y reemplaza el método:

```
private void lanzarEvento(int x, int y, boolean pressed) {
   InputDriverEvent event - new InputDriverEvent();
   event.setPosition{MotionEvent.AXIS_X, x);
   event.setPosition{MotionEvent.AXIS_Y, y);
   event.setContact{pressed);
   driver.emit(event);
}
```

4.8.2.2. Escribir controladores de Ubicación

Como veremos va a ser muy sencillo. Primero solicita el permiso:

```
<uses-permission
   android:name-"com.google.android.things.permission.MANAGE_GNSS_DRIVERS" />
```

Utiliza como base el driver anterior. En la creación reemplaza:

```
driver - new GnssDriver();
```

Y utiliza el método:

```
private void lanzarEvento(DatosDePosicion datos) {
   Location location - obtenLocalización(datos);
   driver.reportLocation(location);
}
```

Preguntas de repaso: Controladores de usuario en Android Things

4.9. Integrar Google Assistant SDK

Google ha apostado muy fuerte por el control de dispositivos IoT por medio de la voz. Para facilitar la integración de esta tecnología en cualquier dispositivo ha creado Google Assistant SDK for devices. Este SDK brinda dos opciones para integrar el Asistente en tu dispositivo:

Google Assistant Library

Esta biblioteca está escrita en Python y es compatible con dispositivos con arquitecturas linux-armv7l y linux-x86_64. La biblioteca expone una API de alto nivel basada en eventos que es fácil de ampliar. Proporciona las siguientes características lista para usar:

- Activación manos libres: con Hey Google o Ok Google , ¡al igual que con Google Home!
- Captura y reproducción de audio
- Gestión del estado de conversación
- Gestión de temporizador y alarma

Servicio Asistente de Google

Es la opción más flexible y con más amplio soporte. Expone una API de bajo nivel que manipula directamente los bytes de audio de una solicitud y respuesta de un asistente. Está basado en gRPC[9], por lo que puede ser usado desde cualquier plataforma que los admita. No admite las características que acabamos de listar, por lo que tendrías que implementarlas por tu cuenta.

Acciones en Google

El SDK nos permitirle agregar una funcionalidad única para nuestro dispositivo con Acciones en Google. Las acciones pueden ser registradas y creadas a través de la consola de acciones.

Nombre	Dispositivos	Descripción	Frases
OnOff	cualquiera	conectar / desconectar el dispositivo	Turn on / Encender Turn off / Apagar
StartStop	cualquiera	arranque / parada general	Start the washing machine.
Modes	cualquiera	dispositivos que pueden trabajar en diferentes modos.	What mode is the dryer in? Set the dryer todelicate.
Toggles	cualquiera	cambia botones físicos, asociados a una función.	Is my dryer sterilization on? Turn on sterilization for the dryer.
Brightness	Light	nivel de brillo de 0 a 100	Adjust my light to 65% bright-ness. / Ajusta mi luz a un 50% de brillo.
FanSpeed	Air conditioning unit, Air purifier, Fan	velocidad del ventilador	What speed are the fans in the living room?
Tempera-tureSetting	Air conditioning , Thermostat	temperatura de la habita-ción	Initialize Thermostat setting.
RunCycle	cualquiera	dispositivos vasados en ciclos de trabajo	What is the washing machine doing?
Locator	Vacuum, otros móviles	para preguntar ubicación del dispositivo.	Where are my keys?

[9] Sistema de llamadas a procedimiento remoto (RPC) de código abierto desa-rrollado por Google. Puede considerarse una alternativa a REST. Usa HTTP / 2.

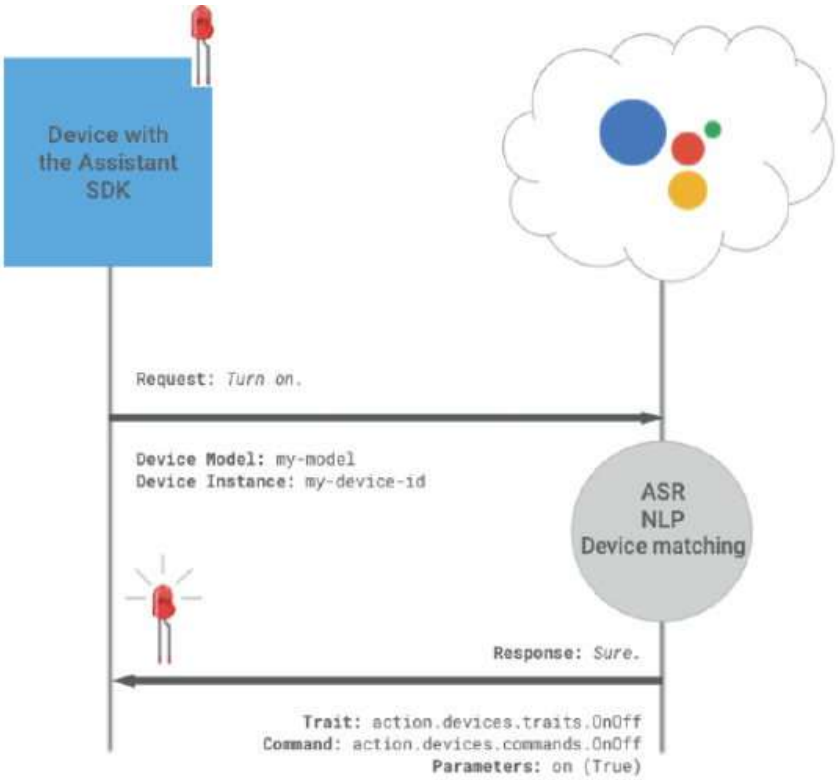
Nota: En la actualidad estas acciones han sido definidas para los idiomas inglés (en), alemán (de), francés (fr) y japonés (ja). Incluso algunas en italiano (it). En la documentación se indica que todavía no se han traducido al español. No obstante, parece que desde junio de 2018 ya funcionan en castellano. En la tabla anterior se indican algunas traducciones que han sido verificadas.

Para implementar un diálogo básico con el Asistente sigue estos pasos:

1. Implementa un cliente gRPC de transmisión de audio bidireccional.
2. Espera a que el usuario active una solicitud (por ejemplo, pulsando un botón).
3. Envía un mensaje `AssistRequest` de tipo `config` (ver `AssistConfig`) con los siguientes campos:
   `audio_in_config`: formato del audio que se va a enviar. (ver `AudioInConfig`)
   `audio_out_config`: formato deseado para el audio recibido (`AudioOutConfig`).
   `device_config`: dispositivo registrado en el Asistente (ver `DeviceConfig`).
   `dialog_state_in`: estado del dialogo, ej. idioma, localización (`DialogStateIn`).
4. Empieza a grabar.
5. Envía mensajes `AssistRequest` con el audio de la consulta.
6. Recoge los mensajes `AssistResponse` entrantes.
7. Extrae los metadatos del mensaje. Por ejemplo, `conversation_state`, si se quiere cambiar el volumen o texto suplementario para visualizar en una pantalla (ver `DialogStateOut`).
8. Detén la grabación cuando reciba un `AssistResponse` con un `event_type` igual a `END_OF_UTTERANCE`.
9. Reproduce el audio de la respuesta del Asistente.
10. Toma el `conversation_state` que extrajiste antes y cópialo en el `DialogStateIn` del mensaje en siguiente `AssistRequest`.

Si dispones de acciones específicas para tu dispositivo añade estos pasos:

1. En los mensajes entrantes, extrae el `device_action` (ver `DeviceAction`).
2. Analiza el campo JSON `device_request_json`. Consulta la página de características de cada tipo de dispositivo para ver la lista de características admitidas. Cada esquema de características muestra una solicitud EXECUTE de ejemplo con los comandos del dispositivo y los parámetros que se devuelven.

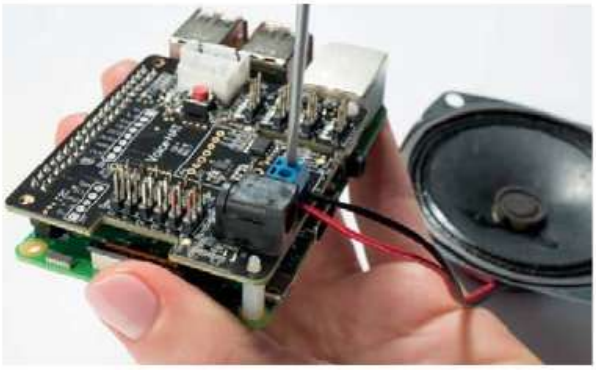


Si dispones de una entrada de texto, como un teclado, puedes enviar este texto en lugar de la voz en el campo `text_query` de `AssistConfig`.

Para demostrarlo vamos a seguir el siguiente tutorial, aunque adaptando las entradas salida, y el idioma por defecto.

https://codelabs.developers.google.com/codelabs/androidthings-assistant

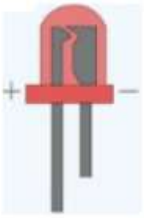
La mayor diferencia es que no vamos a necesitar en Voice Kit. Compuesto por una Raspberry Pi Zero WH, altavoz, pulsador, caja de cartón y VoiceHat. Como se muestra en la siguiente figura, VoiceHat es una placa conectada al bus GPIO, para entrada/salida de audio. Nosotros la reemplazaremos por una mini tarjeta de sonido por USB (imagen de la derecha). También podrías usar un micrófono por Bluetooth.



Ejercicio: Preparar entradas y configuración de Google Assistant

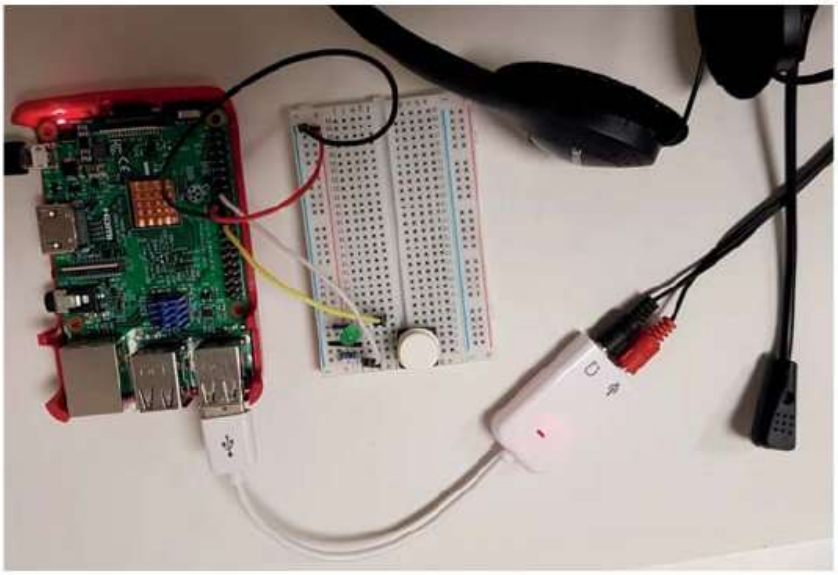
Corresponde a los pasos 2 y 3 del Codelab en que nos basamos.

Material necesario:



- Raspberry Pi 3B
- tarjeta SD con Android Things
- un pulsador
- una resistencia de pull-up[10] de 10 KΩ (color: café, negro, naranja).
- LED y resistencia de ajuste[11]
- tablero de prototipos y cables.
- Entrada salida audio por USB[12]
- Micrófono y auricular

1. Conecta la tarjeta de sonido por USB a una de los cuatro puertos que tiene la Raspberry Pi. Conecta a su vez el micrófono y auricular a la tarjeta.



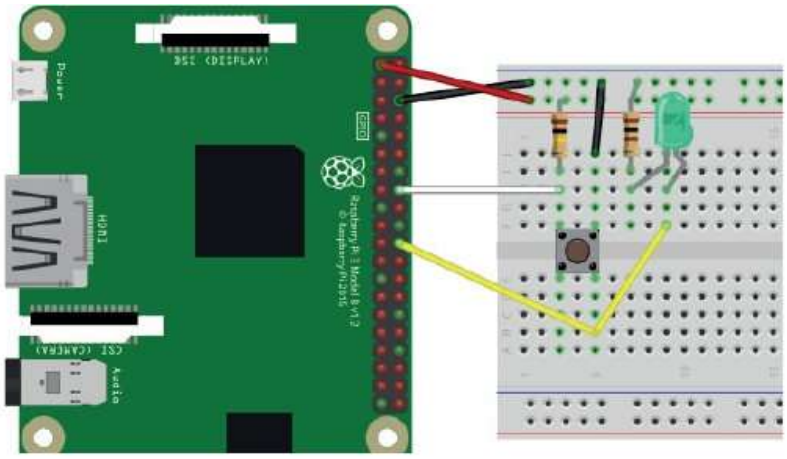
2. Monta el pulsador y el LED en la placa de prototipos y conectarlos a la Raspberry Pi. El siguiente esquema te muestra cómo hacerlo:

---

[10] Ver apartado: Qué es una resistencia pull-up

[11] Ver apartado: LED y resistencia de ajuste

[12] Por ejemplo: http://www.dx.com/p/bstuo-virtual-3d-stereo-7-1-channel-usb-sound-card-silver-472444#.WvRn0IiFM2w



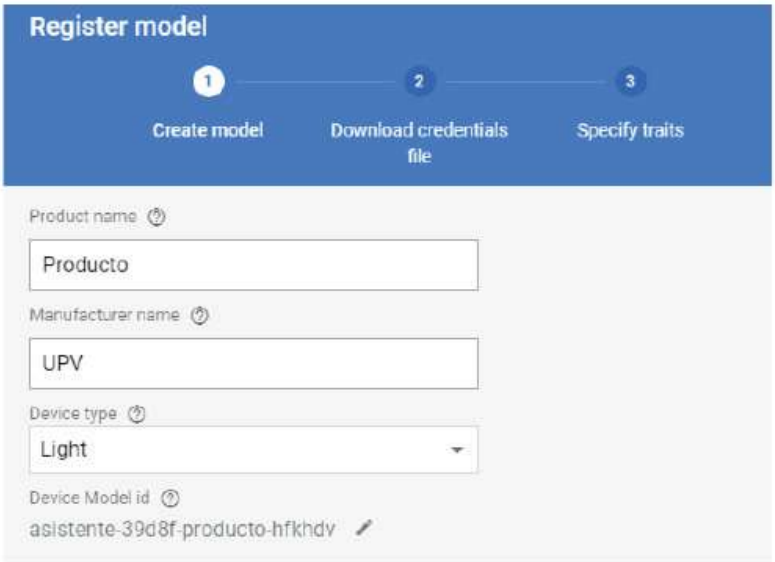
- Conecta una conexión del botón al pin de entrada GPIO BCM23. Conecta el mismo pin a 3.3V a través de una resistencia pull-up de 10 KΩ. Conecta el otro extremo del botón a tierra.
- Conecta la salida GPIO BCM25 al ánodo del LED (pata más larga). Conecta el cátodo del LED (pata más corta) a tierra a través de una resistencia de ajuste (típicamente 100Ω para rojo, amarillo o verde y 10Ω para azul, violeta o blanco).

3. Vamos a configurar las credenciales del proyecto. Entra en Controles de actividad de tu cuenta Google: https://myaccount.google.com/activitycontrols
4. Activa las siguientes funciones:

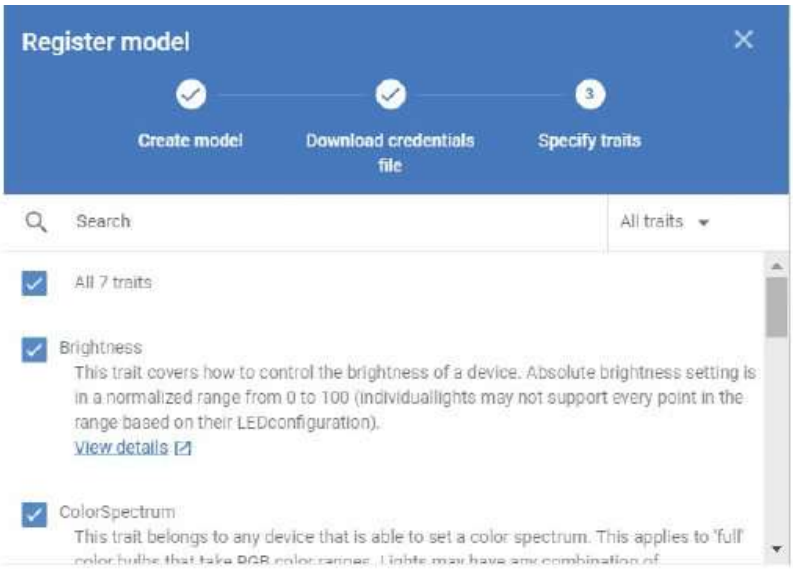
   Actividad en la Web y en Aplicaciones

   Información del dispositivo

   Actividad de Voz y Audio
5. Abre la consola de Acciones: https://console.actions.google.com
6. Crea un proyecto nuevo o selecciona uno existente.
7. Selecciona en el menú de la izquierda la opción Device registration. Introduce los siguientes valores:



8. Pulsa en REGISTER MODEL. En el siguiente paso descarga el fichero credentials.json y guárdalo en la carpeta raíz de tu proyecto.
9. En el último paso selecciona las 7 características (traits) que puede tener un dispositivo tipo Ligth. No las vamos a implementar todas, pero al seleccionarlas vamos a poder interactuar con el asistente para modificar estas características.



10. Pulsa en SAVE TRAITS.
11. Selecciona en el menú de la izquierda la opción Language. Además de English, selecciona Spanish. De esta forma podremos dictar las frases en español.
12. Habilita Google Assistant API en la consola de Google APIs: https://console.developers.google.com/apis/api/embeddedassistant.googleapis.com/overview
13. Desde esta consola puedes consultar las cuotas:

Nombre de cuota	Límite
(Unlabeled quota) por día	500
(Unlabeled quota) por minuto por usuario	60

14. Instala python3, en caso de que no lo tengas ya instalado.
15. En el ordenador de desarrollo abre una consola con permiso de administrador. Ejecuta el siguiente comando:

```
pip install --upgrade pip setuptools wheel
pip install --upgrade google-auth-oauthlib[tool]
```

De esta forma instalamos dos librerías necesarias.

16. Sitúate en la carpeta raíz del proyecto y ejecuta el siguiente comando:

```
google-oauthlib-tool --client-secrets credentials.json
--credentials shared/src/main/res/raw/credentials.json
--scope https://www.googleapis.com/auth/assistant-sdk-prototype --save
```

Esto abrirá un navegador y te pedirá que autorices que la aplicación solicite el asistente en tu nombre.

17. Verifica que en la carpeta shared/src/main/res/raw se ha creado un nuevo fichero credentials.json, aunque esto no es igual al descargado.

Ejercicio: Versión inicial de Google Assistant

Corresponde al paso 4 del Codelab y al módulo step1.

1. Arranca la Raspberry Pi y averigua su dirección IP.
2. Desde el ordenador de desarrollo ejecuta:

```
adb connect <ip-address>
```

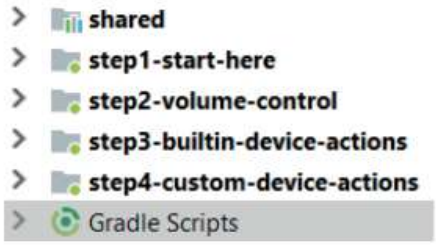
3. Abre Android Studio y selecciona:

   File/New/Project form Version Control/GitHub

4. Indica la siguiente URL:

   https://github.com/googlecodelabs/androidthings-googleassistant.git

5. Verifica como el proyecto está dividido en cinco módulos:



   Uno común (shared) y cuatro que corresponden a versiones cada vez más elaboradas del proyecto. Cada uno de los ejercicios propuestos utilizarán un módulo distinto.

6. En el módulo shared abre la clase `MyDevice`, introduce el valor correcto para `MODEL_ID` según lo obtenido en el ejercicio anterior.

```
public class MyDevice {
   public static final String MODEL_ID - "asistente-39d8f-producto-d91grk";
   public static final String INSTANCE_ID - "id_unica_por_dispositivo";
   public static final String LANGUAGE_CODE - "es-ES"; //"en-US"
}
```

   Utiliza un valor arbitrário en `INSTANCE_ID`. Indica también que quieres utilizar el idioma español.

7. La clase `BroadDefaults` permite definir los puertos GPIO del botón y del LED.
8. Los cuatro AndroidManifest.xml que encontrarás en cada módulo, no instalan la actividad como la de inicio del dispositivo. Si quieres que al reinicializar la Raspberry Pi se ejecute automáticamente, añade las siguientes líneas:

```
<intent-filter>
   <action android:name-"android.intent.action.MAIN" />
   <category android:name-"android.intent.category.IOT_LAUNCHER" />
   <category android:name-"android.intent.category.DEFAULT" />
</intent-filter>
```

9. Otro cambio que has de realizar en los 4 módulos es cambiar la variable.

```
private static final boolean USE_VOICEHAT_DAC - true false;
```

Si no lo cambias, tratará de utilizar el utilizar el VoiceHat y no funcionará.

10. Ejecuta el módulo step1:

11. Mantén pulsado el botón mientras te comunicas con el asistente. Puedes preguntar la hora o cualquier otra información. Al soltar el botón escucharás la respuesta.
12. Si conectas un monitor podrás ver una lista de las frases reconocidas más probables.
13. Si estudias su código

```
@Override public void onButtonEvent(Button button, boolean pressed) {

   mLed.setValue(pressed);
   if (pressed) mAssistantHandler.post(mStartAssistantRequest);
   else         mAssistantHandler.post(mStopAssistantRequest);
}
```

14. El Runnable mStartAssistantRequest pone la grabación en marcha mAudioRecord.startRecording();. Luego configura todos los parámetros de asistente usando converseConfigBuilder

4.9.1. Añadir control de volumen

La API de Google Assistant permite controlar el volumen del dispositivo asistente a través de la voz con consultas como: "baja el volumen" o " pon el volumen a 6". Si prueba esas consultas con el proyecto de inicio, comprobarás que el Asistente todavía no las comprende. Hay que proporcionar información sobre el volumen actual del dispositivo antes de que el Asistente pueda actualizarlo.

En el siguiente ejercicio aprenderás a capturar estas acciones para poder configurar el volumen de la salida de audio del asistente.

Ejercicio: Añadir control de volumen

1. Abre el módulo step-2 del proyecto. Observa como se ha añadido la siguiente variable:

```
private static int mVolumePercentage - 100;
```

2. A la hora de configurar el audio de salida haz el siguiente cambio: (a diferencia del resto de paso, que ya están hechos, este no lo está)

```
AssistConfig.Builder converseConfigBuilder - AssistConfig.newBuilder()
        .setAudioInConfig(ASSISTANT_AUDIO_REQUEST_CONFIG)
        .setAudioOutConfig(ASSISTANT_AUDIO_RESPONSE_CONFIG)
        .setAudioOutConfig{AudioOutConfig.new8uilder{)
                .setEncoding{ENCODING_OUTPUT)
                .setSampleRateHertz{SAMPLE_RATE)
                .setVolumePercentage{mVolumePercentage)
                .build{))
```

```
        .setDeviceConfig(DeviceConfig.newBuilder()
                .setDeviceModelId(MyDevice.MODEL_ID)
                .setDeviceId(MyDevice.INSTANCE_ID)
                .build());
```

3. En el método `onNext()` se procesan las acciones reconocidas por el asistente. En caso de ser necesario actualizaremos el volumen utilizando `mAudioTrack.setVolume()`:

```
@Override public void onNext(AssistResponse value) {

   if (value.getDialogStateOut() !- null) {
      int volume - value.getDialogStateOut().getVolumePercentage();
      if (volume > 0) {
         mVolumePercentage - volume;
         Log.i(TAG, "assistant volume changed: " + mVolumePercentage);
         mAudioTrack.setVolume(AudioTrack.getMaxVolume() *
              mVolumePercentage / 100.0f);
      }
      mConversationState- vaue.getDialogStateOut().getConversationState();
   }
```

4. Ejecuta el módulo step-2. Dicta la frase "pon el volumen a 2".
5. Verifica que en el logCat aparece: `assistant volume changed: 20`

Nota: Aunque el comando es reconocido es posible que no aprecies un cambio en el volumen. Puede ser algún problema de compatibilidad con la tarjeta de sonido.

Nota: El volumen se restablecerá cada vez que reinicie el dispositivo. Utiliza *SharedPreferences* para guardar este nuevo volumen.

4.9.2. Usar acciones predefinidas en Google Assistant

Cuando utilizamos el Asistente de Google para interaccionar con nuestro dispositivo, es posible que queramos realizar ciertas acciones, como encenderlo, apagarlo, configurar su brillo, . Para hacer esto, puede usar acciones predefinidas. Podemos configurar el asistente para que nuestro dispositivo reciba alguna de estas acciones, y así podamos actuar en consecuencia.

Este esquema de interacción se conoce como Acciones (Actions). Puedes encontrar una descripción más detallada en:

https://developers.google.com/actions/smarthome/

Cuando diga un comando que su dispositivo puede admitir, recibirá una carga JSON que le permitirá manejar la consulta directamente.

Cada dispositivo va a tener una serie de características (device traits) que podremos modificar con cada acción. Al principio de este apartado se muestra una tabla con algunas de estas características. La tabla completa puede consultarse en https://developers.google.com/actions/smarthome/traits/.

Ejercicio: Usar acciones predefinidas en Google Assistant

1. En el ejercicio anterior configuramos ciertas características para nuestro dispositivo. Aunque en el tutorial original, te piden que lo hagas ahora, nosotros hemos decidido hacerlo antes.

   Nota: Si modificas las características sobre un modelo ya creado es posible que estas no se activen. Si tienes este problema, crea un nuevo modelo y borra el anterior.

2. En el ordenador de desarrollo abre una consola con permiso de administrador. Sitúate en la carpeta raíz del proyecto y ejecuta el siguiente comando:

```
google-oauthlib-tool --client-secrets credentials.json --scope
https://www.googleapis.com/auth/assistant-sdk-prototype --save
```

   Se abrirá una página Web y te pedirá permiso correspondiente.

3. Ejecuta el siguiente comando, para instalar la herramienta:

```
pip install google-assistant-sdk
```

4. Ejecuta el siguiente comando, reemplazando tu project-id:

```
googlesamples-assistant-devicetool --project-id asistente-39d8f list --
model
```

   Se mostrará una información similar a:

```
Device Model ID: asistente-39d8f-nuevo-d91grk
        Project ID: asistente-39d8f
        Device Type: action.devices.types.THERMOSTAT
        Trait  action.devices.traits.Brightness
        Trait action.devices.traits.ColorSpectrum
        Trait action.devices.traits.ColorTemperature
        Trait action.devices.traits.Dock
        Trait action.devices.traits.OnOff
        Trait action.devices.traits.StartStop
        Trait action.devices.traits.TemperatureSetting
```

5. Ejecuta el siguiente comando, reemplazando los ids marcados:

```
googlesamples-assistant-devicetool --project-id asistente-39d8f register-
device --model asistente-39d8f-nuevo-d91grk --device instan-
ce_id_unica_unica_por_dispositivo_2 --client-type SERVICE
```

6. Te indicará el mensaje "`Device instance registered`"
7. Cuando un usuario pronuncie alguna frase que coincida con alguna de las acciones que modifique alguna característica activada, nuestra aplicación será notificada. En concreto se enviará un JSON similar al siguiente:

```
{"requestId": "ff36a3cc-ec34-11e6-b1a0-64510650abcf",
  "inputs": [{
    "intent": "action.devices.EXECUTE",
    "payload": {
      "commands": [{
        "devices": [{ "id": "123" }],
        "execution": [{
```

```
          "command": "action.devices.commands.OnOff",
          "params": { "on": true }
        }]
      }]
    }
  }]
}
```

8. Para tratar esta respuesta se utiliza el siguiente código, que encontrarás en el módulo step-3 del proyecto.:

```
@Override public void onNext(AssistResponse value) {
   if (value.getDeviceAction() !- null &&
         !value.getDeviceAction().getDeviceRequestJson().isEmpty()) {
      try {
         JSONObject deviceAction - new JSONObject(value.getDeviceAction()
                                                  .getDeviceRequestJson());
         JSONArray inputs - deviceAction.getJSONArray("inputs");
         for (int i - 0; i < inputs.length(); i++) {
            if (inputs.getJSONObject(i).getString("intent")
                  .equals("action.devices.EXECUTE")) {
               JSONArray commands - inputs.getJSONObject(i)
                     .getJSONObject("payload")
                     .getJSONArray("commands");
               for (int j - 0; j < commands.length(); j++) {
                  JSONArray execution - commands.getJSONObject(j)
                        .getJSONArray("execution");
                  for (int k - 0; k < execution.length(); k++) {
                     String command - execution.getJSONObject(k)
                           .getString("command");
                     JSONObject params - execution.getJSONObject(k)
                           .optJSONObject("params");
                     handleDeviceAction(command, params);
                  }
               }
            }
         }
      } catch (JSONException 1 IOException e) {
         e.printStackTrace();
      }
   }
}
```

9. Para tratar cada uno de los comandos, se utiliza el siguiente método:

```
public void handleDeviceAction(String command, JSONObject params)
        throws JSONException, IOException {
   // El parametro que envia ha cambiado desde lo que pone en el ejemplo
   //if (command.equals("action.devices.traits.OnOff")) {
   if (command.equals("action.devices.commands.OnOff")) {
      mLed.setValue(params.getBoolean("on"));
   }
}
```

Si el comando detectado es de tipo OnOff. extraemos el parámetro que indica el estado y encendemos o apagamos el LED según este parámetro.

Nota: Si en el código aparece "action.devices.traits.OnOff" reemplaza “traits” por “commands”.

10. Ejecuta el módulo step-3 del proyecto. Pulsa el botón y di “Encender”/ “Turn on”; el LED se encenderá. Pulsa el botón y di “Apagar”/ “Turn off”; el LED se apagará.

Práctica: **Control del brillo de un LED por voz**

Utiliza un segundo LED para controlar su encendido, deja el LED actual para verificar la pulsación del botón. Este segundo LED ha de conectarse a una salida PWM. Cuando el usuario diga la frase “Ajusta mi luz a un to 60% de brillo” / “Adjust my light to 60% brightness” el LED ha de estar en estado alto el 60% del tiempo.

4.9.3. Definir acciones personalizadas

Además de hablar con el Asistente de Google, es posible que necesites que tu dispositivo realice ciertas acciones no recogidas en las características que acabamos de exponer. Para hacer esto, puede usar acciones de dispositivos personalizados.

Vídeo[Tutorial]: Introduction to Custom Actions for the Google Assistant SDK

Estas acciones se pueden definir con una gramática personalizada que puede incluir parámetros. Cuando diga un comando que su dispositivo puede admitir, recibirás una respuesta JSON que te permitirá manejar la consulta.

El siguiente ejemplo define una acción personalizada para conseguir un parpadeo en el LED. Podemos especificar que parpadee un número determinado de veces, así como la frecuencia que se definirá como un tipo personalizado.

```
{"manifest": {
    "displayName": "Blinky light",
    "invocationName": "Blinky light",
    "category": "PRODUCTIVITY" },
  "actions": [    {
      "name": "com.example.actions.BlinkLight",
      "availability": {
        "deviceClasses": [ {"assistantSdkDevice": {} } ] },
      "intent": {
        "name": "com.example.intents.BlinkLight",
        "parameters": [
          { "name": "number",
            "type": "SchemaOrg_Number" },
          { "name": "speed",
```

```
            "type": "Speed" } ],
        "trigger": {
          "queryPatterns": [
            "parpadea {$Speed:speed)? $SchemaOrg_Number:number veces",
            "parpadea $SchemaOrg_Number:number veces {$Speed:speed)?" ] }
      },
      "fulfillment": {
        "staticFulfillment": {
          "templatedResponse": {
            "items": [
              {"simpleResponse": {
                  "textToSpeech": "Parpadeando $speed.raw $number veces"
                } },
              {"deviceExecution": {
                  "command": "com.example.commands.BlinkLight",
                  "params": {
                    "speed": "$speed",
                    "number": "$number" }}}
            ]
          }
        }
      }
    ],
    "types": [
      { "name": "$Speed",
        "entities": [
          { "key": "lentamente",
            "synonyms": [ "lento" ] },
          { "key": "normal",
            "synonyms": ["regular" ] },
          { "key": "rápidamente",
            "synonyms": [ "rápido" ] }
        ]
      }
    ]
}
```

Ejercicio: Definir acciones personalizadas en Google Assistant

1. Copia el código J SON anterior en el fichero actions.json. Por ejemplo, en la raíz del proyecto.
2. Descarga la herramienta gactions[13].
3. Con esta herramienta ejecuta (reemplazando el id de proyecto):

```
gactions test --action_package actions.json --project asistente-39d8f
```

[13] https://developers.google.com/actions/tools/gactions-cli

Con esto tu dispositivo podrá recibir el comando definido en "com.example.commands.BlinkLight".

4. Dentro de módulo step-4 se ha añadido

```
public void handleDeviceAction(String command, JSONObject params)
                                        throws JSONException, IOException {
   mLedHandler.removeCallbacksAndMessages(null);
   if (command.equals("action.devices.traits.commands.OnOff")) {//ERROR
      mLed.setValue(params.getBoolean("on"));
   } else if (command.equals("com.example.commands.BlinkLight")) {//ERROR
      int delay - 1000;
      int blinkCount - params.getInt("number");
      String speed - params.getString("speed");
      if (speed.equals("slowly")) delay - 2000;
      else if (speed.equals("quickly")) delay - 500;
      for (int i - 0; i < blinkCount * 2; i++) {
         mLedHandler.postDelayed(() -> {
            try {
               mLed.setValue(!mLed.getValue());
            } catch (IOException e) {
               throw new RuntimeException(e);
            }
         }, i * delay);
      }
   }
}
```

5. Ejecuta el módulo step-4 del proyecto. Pulsa el botón y di “parpadea 4 veces”. El LED ha de encenderse y apagarse 4 veces.
6. Pulsa el botón y di “parpadea rápido 4 veces”. Ha de hacer lo mismo, pero de forma más rápida.

Nota: En junio de 2018, este ejercicio no funcionaba en castellano. Aunque sí que funcionaba en inglés. Puedes encontrar la definición de las acciones en inglés en el tutorial original.

Preguntas de repaso: Google Assistant SDK en Android Things

CAPÍTULO 5.

Android Things: Comunicaciones

Por SALVADOR SANTONJA

Como hemos comentado, el término «Internet de las cosas» une los conceptos de «Internet» y «las cosas». En la unidad anterior hemos aprendido cómo funciona el pilar de «**las cosas**», cómo podemos interactuar mediante entradas y salidas genéricas, así como mediante buses de comunicación, con elementos del mundo real. En este segundo capítulo nos enfrentaremos al pilar de «**Internet**» en su sentido más amplio: revisaremos las opciones de conectividad que nos aporta Android Things en general, y la Raspberry Pi 3 en particular, y aprenderemos a trabajar con las opciones de comunicaciones más relevantes que nos propone el universo del Internet de las cosas.

Comenzaremos describiendo las comunicaciones offline, que son aquellas que no disponen de una conexión a través de Internet. Puede parecer algo contradictorio dentro del mundo de «Internet de las cosas», pero es una práctica habitual. Dentro de este tipo de comunicaciones estudiaremos las tecnologías Bluetooth y LoWPAN. Además, veremos la propuesta de Google en este contexto, Nearby Connections.

La segunda parte se centrará en las comunicaciones online, o a través de Internet. Se estudian los modelos request/response y publish/subscribe. Para ilustrar cada modelo se describen el uso de servicios web REST y el protocolo MQTT.

Objetivos:

- Conocer y comprender las características de las comunicaciones en el Internet de las cosas.
- Enumerar las principales alternativas para comunicación offline.
- Descubrir y configurar nodos IoT usando Nearby Connections.
- Comparar los principales modelos de comunicaciones online: request/response y publish/subscribe.
- Desarrollar un servidor web embebido en la Raspberry Pi.
- Utilizar servicios web REST en Android Things.
- Aprender a utilizar MQTT para aplicaciones de IoT.

5.1 Opciones de comunicación en Android Things

La plataforma Android Things integra de forma nativa las comunicaciones a través de redes IP, sobre las que establecemos sockets para intercambiar datos. Esto nos permite utilizar de forma transparente las interfaces más típicas como Ethernet o wifi. Por su origen en el mundo de la tecnología móvil, también integra conectividad Bluetooth (tecnología punto a punto para conectar un equipo principal o maestro con elementos periféricos). Bluetooth es un estándar que ha evolucionado mucho desde su creación, agregando cada vez más opciones y servicios, y permite en sus últimas versiones crear incluso redes de dispositivos para intercambio de datos. Además de esto, Android Things también integra librerías de comunicación LoWPAN, diseñadas para interfaces de comunicación de bajo consumo y baja tasa de transferencia de datos, muy utilizadas en redes inalámbricas de sensores.

Es posible clasificar las tecnologías de comunicación de muchas formas, no solo como cableadas o inalámbricas. Por ejemplo, podemos hablar de comunicaciones offline u online, según si estas ofrecen una conexión a través de Internet o no. Un ejemplo de comunicación offline sería la conexión de un altavoz inalámbrico con el móvil, el intercambio de una foto entre dos móviles por Bluetooth, o incluso el intercambio de ficheros entre dos PCs por wifi a través de un punto de acceso doméstico. En una comunicación online, nuestro equipo estaría conectado a Internet para ofrecer o consumir servicios de él: escritorio remoto con un equipo en otra ubicación, navegar por páginas web, consultar el correo electrónico, utilizar aplicaciones web o un servicio de almacenamiento en la nube.

También podemos clasificar las tecnologías de comunicación como públicas (cuando la infraestructura la provee un operador a aquellos usuarios dispuestos a pagar su cuota, como las redes GSM o 3G) o privadas (cuando la infraestructura la gestiona una organización o particular y no está abierta a todo el público, como una red de área local). Otra opción es clasificarlas por su rango de alcance, como

por ejemplo las tecnologías de área extensa (WAN, wide area networks), de área local (LAN, local area networks) o incluso de área personal (PAN, personal área networks).

Al final, la tecnología de comunicaciones es la responsable de proveernos de un canal físico para el intercambio de datos entre nuestro sistema y un sistema remoto o, dicho de otra forma, de ofrecernos «conectividad». En el mundo del Internet de las cosas, hay tantas aplicaciones posibles que las tecnologías que aparecen en todas estas clasificaciones pueden ser susceptibles de ser utilizadas. Según las necesidades de conectividad, un paso vital en el diseño de una solución IoT será la selección del hardware adecuado para soportar estas comunicaciones.

En nuestro caso, la Raspberry Pi 3 dispone de las siguientes tecnologías de comunicación integradas en placa:

- 10/100 Ethernet
- WLAN a 2.4GHz 802.11b,g,n
- Bluetooth 4.1
- Bluetooth Low Energy

Además, es posible agregar otras interfaces de comunicación mediante USB, o mediante los buses serie de la placa, como por ejemplo GSM/3G, o un módulo 802.15.4 para comunicaciones LoWPAN.

5.2 Comunicaciones offline

En la era de Internet podría parecer extraño dedicar esfuerzos a comunicaciones offline pero, lejos de lo que pueda parecer, este tipo de tecnología tiene multitud de ventajas que la convierten en una solución clave para muchas aplicaciones. Las principales características de una conexión offline son:

- **Baja latencia**: al comunicar directamente dos dispositivos sin someterlos a la carga temporal que sufren los mensajes a través de Internet, los tiempos de envío y recepción son los más bajos que podemos encontrar. Esto hace que las comunicaciones offline sean el sistema ideal para controles en tiempo real, como el control remoto de un robot, o como juegos multiusuario.
- **Altas tasas de transferencia**: al evitar intermediarios, dos equipos conectados directamente mediante una interfaz de comunicación común tendrán la posibilidad de exprimir esa interfaz al máximo. Una conexión Ethernet de 1Gbps tiene un potencial enorme en una red local, pero pierde su sentido al utilizar una conexión a Internet de 50 Mbps. Las aplicaciones de transferencia de datos se ven muy beneficiadas con este tipo de conexiones.
- **Conexiones estables**: una conexión a Internet está sujeta a caídas de servicio, tanto por el propio acceso (cobertura de redes móviles, proveedores con cortes de conexión, zonas remotas con conexiones intermitentes), como por los nodos intermedios de los que depende una conexión extremo a extremo. En una red offline, es más fácil conseguir los requisitos de estabilidad de tu aplicación, y mantenerlos en unos límites acotados.

- **Coste**: en la mayoría de casos, un acceso a Internet incurre en costes con el proveedor de servicios: máquinas de bebidas que controlan su stock, contenedores de basura inteligentes que avisan cuando están llenos, vehículos conectados, etc. Sin embargo, las conexiones offline pueden reducir la cantidad de puntos a conectar a Internet. Por ejemplo, los sensores en una granja pueden conectarse entre ellos con wifi o LoWPAN, y salir por un único punto hacia Internet a través de un concentrador.
- **Seguridad y privacidad**: es importante tener en cuenta los riesgos que conlleva una conexión a Internet. Por un lado, nuestros datos circulan libremente y, según su criticidad, puede ser necesario utilizar mecanismos para asegurar la privacidad de la información y hacer seguras las comunicaciones. Por otro lado, un sistema IoT conectado a Internet está sujeto a posibles ataques de seguridad, que pueden provocar desde denegaciones de servicio hasta tomar su control completo.
- **Consumo energético**: el consumo en comunicaciones de nuestro sistema depende profundamente de la tecnología y de los protocolos de comunicaciones utilizados. Las comunicaciones móviles, como 3G/4G, utilizan sistemas de señalización con las antenas base y elevadas potencias de transmisión, que agotan la batería de los móviles en poco tiempo. WLAN es una tecnología diseñada para imitar una LAN sin cables, por lo que los dispositivos finales deben mantener una conectividad y una señalización constante con el punto de acceso. Sin embargo, los equipos IoT son más propensos a presentar requisitos de bajo consumo, puesto que son equipos sobre los que no se espera realizar un mantenimiento diario, así que incluir un módem 4G en un sistema con baterías resulta prohibitivo. El uso de tecnologías de bajo consumo, como Zigbee o Lora, permite comunicar muchos dispositivos con poco consumo, y alimentar un único dispositivo central que ofrezca el enlace a Internet.

Podemos desarrollar aplicaciones con comunicación offline a través de cualquier interfaz, incluso wifi y Ethernet. Pero también hay tecnologías que han sido desarrolladas de forma nativa para comunicaciones offline, como Bluetooth o Zigbee. Las 3 principales opciones de comunicación offline que nos ofrece el SDK de Android Things son Bluetooth, LoWPAN y Nearby Communications.

5.2.1. Bluetooth

Bluetooth es un estándar de comunicación inalámbrica para el intercambio de voz y datos entre dispositivos. Se encuentra especificado en la norma IEEE802.15.1, y está basado en una comunicación maestro-esclavo. Puede formar picorredes de hasta 7 esclavos, donde todos mantienen un enlace punto a punto con el maestro. El objetivo inicial de Bluetooth es el de eliminar cableado y conectores para facilitar la comunicación entre equipos fijos y móviles (ratones, auriculares, altavoces inalámbricos) y para la sincronización de equipos (intercambio de datos entre el móvil y el PC).

La norma Bluetooth cubre la pila de comunicaciones completa y establece requisitos de hardware, software e interoperabilidad entre equipos. Sus especificaciones están divididas en dos grandes bloques:

- Core specifications: define los componentes básicos de la tecnología que se utilizan para el funcionamiento de Bluetooth y su interoperabilidad con otros dispositivos. Esto incluye la descripción de las capas del protocolo (desde la interfaz radio hasta el enlace de datos), el descubrimiento de servicios, canales básicos de comunicación, controladores de host y emuladores de buses serie, y el cumplimiento de interfaces y modos de test. En general, las core specifications definen cómo funciona la tecnología.
- Profile specifications: están orientadas a cómo se usa la tecnología Bluetooth para dar soporte a las distintas aplicaciones que se pueden encontrar. Cada perfil indica cómo utilizar la tecnología del core specification para implementar un modelo de uso concreto. Existen más de 100 perfiles que incluyen teléfono inalámbrico, auriculares inalámbricos, intercambio de ficheros, sincronización, pulsera deportiva, etc.

La última versión de la norma es la 5.0, y es una agrupación de diversos modos de funcionamiento Bluetooth como el clásico (hasta 3 Mbps), el de alta velocidad (que utiliza wifi para el intercambio de datos, hasta 32 Mbps) y el Low Energy (que reduce alcance y tasa de datos a cambio de un consumo muy bajo de energía, hasta 2 Mbps). El alcance depende de la clase de dispositivo: clase 1 (1 m), clase 2 (10 m) y clase 3 (100 m).

5.2.2. LoWPAN

Low Rate Wireless Personal Area Networks (LR-WPAN o LoWPAN) es una tecnología de red inalámbrica de muy bajo consumo, baja tasa de transferencia y soporte a enlaces intermitentes. Está diseñada para dispositivos agrupados en áreas de pequeño tamaño, como redes de sensores inalámbricas, juguetes inteligentes, controles remotos o domótica. Su especificación base es la norma IEEE802.15.4, que describe la interfaz radio y la capa de enlace de datos. En ella se define una tasa de transferencia de 250 kbps, 40 kbps o 20 kbps; y la utilización de 16 canales en la banda de 2.4 GHz, 10 canales en la de 915 MHz o 1 canal en la de 868 MHz.

Sobre estas especificaciones se construyen las diversas tecnologías que podemos encontrar en la actualidad como Zigbee, MiWi, ISA100.11a, WirelessHART, SNAP, Thread o 6LoWPAN. La tendencia actual es la de incluir soporte IP en este tipo de tecnologías, para que los dispositivos sean direccionables desde Internet y su integración en el Internet de las cosas sea transparente. Sin embargo, estos equipos presentan una fuerte restricción de hardware, energía y tamaño de transmisiones, por lo que se utiliza una versión modificada de IPv6 conocida como 6LoWPAN. La pasarela de la red LoWPAN, encargada de realizar el cambio de un dominio inalámbrico LoWPAN a una conexión a Internet, es también la encargada de adaptar 6LoWPAN a IPv6.

Preguntas de repaso: Comunicaciones offline

5.2.3. Nearby Connections

Nearby Connections es la propuesta de Google para abstraernos de las complejidades inherentes al uso de Bluetooth y wifi a la hora de realizar conexiones directas entre equipos. Es un API de proximidad, con comunicaciones punto a punto completamente offline, que nos permite anunciar nuestro equipo a su entorno, descubrir equipos cercanos, establecer conexiones e intercambiar datos. Las conexiones establecidas son completamente seguras (siempre están encriptadas, y opcionalmente podemos autenticar los equipos), y presentan un elevado ancho de banda y una baja latencia. Esto hace que su uso sea idóneo para aplicaciones como juegos multipantalla (un teléfono o tablet como control, y una Android TV como visualización), partidas multijugador (un jugador establece la partida e invita al resto a unirse), pizarras colaborativas, mensajería de proximidad o transferencias de ficheros sin conexión a Internet.

Por debajo, Nearby Connections utiliza una combinación de Bluetooth, Bluetooth Low Energy y wifi, según las fases de la conexión y el tipo de transferencia a realizar. Las interfaces Bluetooth y wifi se gestionan automáticamente por el API, de forma que no es necesario solicitar al usuario que las active; y al cerrar la app vuelven a su estado anterior. Esto permite que la experiencia del usuario sea mucho menos invasiva.

El funcionamiento de Nearby Connections se resume de la siguiente forma:

- **Fase de preconexión**. Existen dos roles: los **anunciantes** y los **descubridores**. Los anunciantes lanzan balizas para que los descubridores cercanos sepan que están ahí y que tienen algo que ofrecer. Muestran un nombre amigable que permite que los descubridores diferencien distintos anunciantes y puedan decidir a cuál quieren conectarse.
- **Fase de establecimiento de conexión**. Cuando un descubridor desea conectarse a un anunciante, lanza una solicitud que inicia un proceso de autenticación simétrica. Ambos lados pueden decidir si aceptan o no esta conexión. Si se acepta en ambos, se establece una conexión encriptada.
- **Fase de comunicación**. Con la conexión establecida, tenemos un enlace punto a punto entre dos dispositivos, y los roles de anunciante y descubridor desaparecen. A partir de ahora, los roles serán de emisor y receptor, y cada nodo puede utilizar uno de estos roles o ambos, lo que definirá si el canal es de un sentido o full-dúplex. Nearby Communications dispone de 3 tipos de intercambio de datos:
    - **BYTES**: transmisión de un byte array de hasta 32k, ideal para intercambiar mensajes y metadatos.
    - **FILE**: intercambio eficiente de ficheros de cualquier tamaño.

    - **STREAM**: intercambio de datos al vuelo, sin un tamaño conocido previamente, ideal para aplicaciones multimedia.

- **Fase de desconexión**. Uno de los nodos (el anunciante o el descubridor) toma la decisión de finalizar la conexión, y la cierra de forma unilateral. El otro nodo detecta la desconexión y libera sus recursos.

La conexión de los dispositivos puede seguir distintas **estrategias**, según la topología de red que deseemos para nuestra aplicación:

- **P2P_STAR**: para topologías en estrella, que permiten la conexión de 1 a N. Es la estrategia idónea cuando se desea tener un dispositivo anunciante al que conectar varios dispositivos de forma simultánea.
- **P2P_CLUSTER**: establece grupos con conexiones de M a N, es decir, donde cada dispositivo puede iniciar conexiones hacia M dispositivos y recibir conexiones de N dispositivos. Es la estrategia más flexible, puesto que forma una topología de tipo malla. Sin embargo, presenta menor ancho de banda y mayor latencia que el resto de soluciones. Esta estrategia es perfecta cuando se intercambian pequeñas cargas que no necesitan pasar por un punto central.
- **P2P_POINT_TO_POINT**: la estrategia más sencilla, pero recién incorporada en la última versión del SDK. Permite la conexión punto a punto entre dos dispositivos dentro de alcance. Consigue el máximo ancho de banda, pero no permite múltiples conexiones.

Para que los dispositivos anunciantes y descubridores puedan detectarse, deben tener dos parámetros en común:

- La **estrategia**: por motivos de operación, la estrategia para formar y gestionar la red debe ser la misma. P2P_CLUSTER es la estrategia por defecto.
- El **serviceID**: un identificador que permite segmentar fácilmente nuestras aplicaciones. Por ejemplo: tenemos un cartel turístico inteligente con una RP3, que lanza mensajes a los turistas cercanos. Nos interesará que la aplicación del móvil solo busque balizas de este servicio, y no otras balizas relacionadas con otros asuntos. Una buena práctica es utilizar el nombre de paquete de nuestra app (por ejemplo, `com.google.example.mipaquete`).

Nota: montaje hardware para los ejercicios de esta unidad.

El objetivo de esta unidad es conocer las principales tecnologías de comunicaciones utilizadas en el Internet de las Cosas y experimentar con ellas, por lo que la interacción con el hardware no es tan relevante como en la unidad anterior. Para agilizar el desarrollo de los ejercicios, se utilizará un único montaje hardware con un LED como salida y un pulsador como entrada.

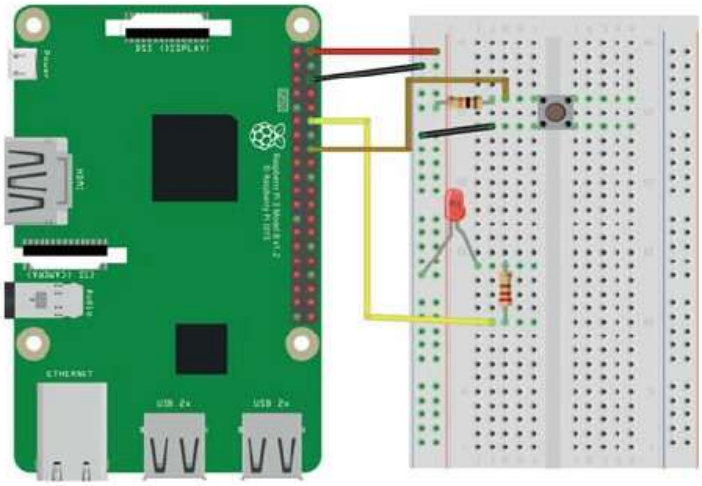
Material necesario:

- un LED

- una resistencia de 220 Ω (color de resistencia rojo, rojo, marrón)

- un pulsador de 4 pines
- una resistencia de 10 kΩ (color de resistencia marrón, negro, naranja)
- tablero de prototipos y cables

Monta los componentes sobre la placa de prototipos y conéctalos a la Raspberry Pi. El siguiente esquema te muestra cómo hacerlo:



+ Para el LED:
  - Conecta el pin de salida GPIO elegido (BCM18) a un extremo de una resistencia de 220 Ω en serie.
  - Conecta el otro extremo de la resistencia al ánodo del LED (pata más larga).
  - Conecta el cátodo del LED (pata más corta) a tierra.

+ Para el pulsador:
  - Conecta uno de los pines con una resistencia de pull-up de 10 kΩ a Vcc, y con un cable al pin de entrada GPIO elegido (BCM23).
  - Conecta el otro pin a tierra.

Ejercicio: Conectividad básica con Nearby Connections

En este primer ejercicio vamos a construir la estructura software necesaria para trabajar con Nearby Connections. Puesto que necesitamos 2 equipos Android para establecer esta comunicación, utilizaremos el terminal móvil y la Raspberry Pi 3. El móvil mostrará una app con un botón para encender y apagar el LED de la RP3, y el API de Nearby Connections nos ofrecerá un enlace directo a ella. En este ejemplo, lo más lógico es que sea la RP3 la que ofrezca el LED a su entorno como un servicio disponible, así que será ella la que estará en modo anunciante. Cuando pulsemos el botón del terminal móvil se realizará su descubrimiento, el proceso de conexión con la RP3, y el envío del comando para actuar sobre el LED.

1. Crea un nuevo proyecto, indicando que vas a desarrollar tanto para teléfono como para Android Things. Esto generará un proyecto con dos módulos java, uno para la app del terminal móvil y uno para la app de la RP3. Utiliza los siguientes datos:

   Application name: Nearby Connections

   O Phone and Tablet

   API 16: Android 4.1 (Jelly Bean)

   O Android Things

   API 27: Android 8.1 (Oreo)

   Add an activity to Mobile: Empty Activity

   Add an activity to Things: Android Things Empty Activity

   Generate a UI layout File

2. Abre el desplegable de los scripts Gradle; verás que aparece un script de proyecto y uno específico para cada módulo. Nearby Connections forma parte del SDK de Google Play Services, así que para utilizarlo será necesario añadir la siguiente dependencia en el build.gradle de los dos módulos (Module: mobile y Module: things):

```
dependencies {
    .
    implementation 'com.google.android.gms:play-services-nearby:11.8.0'

}
```

3. Ahora añade los permisos necesarios en el manifiesto (AndroidManifest.xml) de ambas aplicaciones: la de **mobile** y la de **things**. Los necesarios para el uso de Nearby Communications (Bluetooth y wifi) deberán estar en ambos manifiestos, y el de **USE_PERIPHERAL_IO** solo en el de **things**, para poder utilizar las E/S de nuestra RP3:

```
<manifest .>
<!-Necesario para Nearby Connections -->
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>
<!-Necesario para usar Las GPIO de La RP3 -->
<uses-permission an-
droid:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />

    <application .>
```

   El permiso `ACCESS_COARSE_LOCATION` se considera peligroso, pero es necesario para el uso de Nearby Connections. En el caso de Android Things no hay problema, puesto que los permisos se conceden en el arranque. Pero

en el caso del terminal móvil, será necesario solicitar al usuario este permiso en tiempo de ejecución. Volveremos a esto más adelante.

4. Opcionalmente, puedes aprovechar para incluir en el manifiesto de la aplicación móvil los temas y estilos que más te gusten, como por ejemplo:

```
<application
    .
    android:theme="@style/Theme.AppCompat.Light.NoActionBar">
    <activity
          …
```

5. Comenzamos ahora con el código para la app de nuestra RP3. Abre la clase `MainActivity` del módulo **things**, y añade el siguiente contenido:

```
public class MainActivity extends Activity {
  // Consejo: utiLiza como SERVICE_ID eL nombre de tu paquete
  private static final String SERVICE_ID = "com.example.mipaquete";
  private static final String TAG = "Things:";
  private final String PIN_LED = "BCM18";
  public Gpio mLedGpio;
  private Boolean ledStatus;

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Configuración deL LED
    ledStatus = false;
    PeripheralManager service = PeripheralManager.getInstance();
    try {
      mLedGpio = service.openGpio(PIN_LED);
      mLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
    } catch (IOException e) {
      Log.e(TAG, "Error en el API PeripheralIO", e);
    }
    // Arrancamos modo anunciante
    startAdvertising();
  }

  private void startAdvertising() {
    Nearby.getConnectionsCLient(this).startAdvertising(
      "Nearby LED", SERVICE_ID, mConnectionLifecycleCallback,
      new AdvertisingOptions(Strategy.P2P_STAR))
      .addOnSuccessListener(new OnSuccessListener<Void>() {
        @Override public void onSuccess(Void unusedResult) {
          Log.i(TAG, "Estamos en modo anunciante!");
        }
      })
      .addOnFailureListener(new OnFailureListener()  {
        @Override public void onFailure(@NonNull Exception e) {
          Log.e(TAG, "Error al comenzar el modo anunciante", e);
        }
      });
  }

  private void stopAdvertising() {
    Nearby.getConnectionsCLient(this).stopAdvertising();
```

```
    Log.i(TAG, "Detenido el modo anunciante!");
  }

  private final ConnectionLifecycleCallback mConnectionLifecycleCallback =
    new ConnectionLifecycleCallback() {
      @Override public void onConnectionInitiated(
        String endpointId, ConnectionInfo connectionInfo) {
        // Aceptamos La conexión automáticamente en ambos Lados.
        Nearby.getConnectionsCLient(getApplicationContext())
          .acceptConnection(endpointId, mPayloadCallback);
        Log.i(TAG, "Aceptando conexión entrante sin autenticación");
      }

      @Override public void onConnectionResult(String endpointId,
                                ConnectionResolution result) {
        switch (result.getStatus().getStatusCode()) {
          case ConnectionsStatusCodes.STATUS_OK:
            Log.i(TAG, "Estamos conectados!");
            stopAdvertising();
            break;
          case ConnectionsStatusCodes.STATUS_CONNECTION_REJECTED:
            Log.i(TAG, "Conexión rechazada por uno o ambos lados");
            break;
          case ConnectionsStatusCodes.STATUS_ERROR:
            Log.i(TAG, "Conexión perdida antes de ser aceptada");
            break;
        }
      }

      @Override
      public void onDisconnected(String endpointId) {
        Log.i(TAG, "Desconexión del endpoint, no se pueden " +
          "intercambiar más datos.");
        startAdvertising();
      }
    };

  private final PayloadCallback mPayloadCallback = new PayloadCallback() {
    @Override public void onPayloadReceived(String endpointId,
                                Payload payload) {
      String message = new String(payload.asBytes());
      Log.i(TAG, "Se ha recibido una transferencia desde (" +
        endpointId + ") con el siguiente contenido: " + message);
      disconnect(endpointId);
      switch (message) {
        case "ACTION":
          doRemoteAction();
          break;
        default:
          Log.w(TAG, "No existe una acción asociada a este " +
            "mensaje.");
          break;
      }
```

```
    }

    @Override public void onPayloadTransferUpdate(String endpointId,
                                PayloadTransferUpdate update) {
      // ActuaLizaciones sobre eL proceso de transferencia
    }
  };

  public void doRemoteAction() {
    try {
      if (ledStatus) {
        mLedGpio.setValue(false);
        ledStatus = false;
        Log.i(TAG, "LED OFF");
      } else {
        mLedGpio.setValue(true);
        ledStatus = true;
        Log.i(TAG, "LED ON");
      }
    } catch (IOException e) {
      Log.e(TAG, "Error en el API PeripheralIO", e);
    }
  }

  protected void disconnect(String endpointId) {
    Nearby.getConnectionsCLient(this)
      .disconnectFromEndpoint(endpointId);
    Log.i(TAG, "Desconectado del endpoint (" + endpointId + ").");
    startAdvertising();
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
    stopAdvertising();
    if (mLedGpio != null) {
      try {
        mLedGpio.close();
      } catch (IOException e) {
        Log.e(TAG, "Error en el API PeripheralIO", e);
      } finally {
        mLedGpio = null;
      }
    }
  }
}
```

Vamos a revisar paso a paso qué hace este código:

1. Al arrancar (método `OnCreate()`) configuramos el pin del LED como salida, y lanzamos el método `startAdvertising()` para que la RP3 comience a anunciarse.
2. En el método `startAdvertising()`, llamamos a la librería Nearby Connections para comenzar el modo anunciante, pasando como

parámetros el nombre de nuestra baliza de anuncio (**Nearby LED**), el identificador de servicio, un objeto callback `mConnectionLifecycleCallback` que gestionará todo el proceso de las peticiones de conexión que recibamos, y la estrategia a utilizar. Como queremos que dispositivos externos se conecten a nuestra RP3 para actuar sobre su LED, utilizaremos una estrategia P2P_STAR. El nombre de la baliza y el identificador de servicio serán los mismos que utilizaremos en la app del móvil. Añadimos, además, dos listeners para detectar si hemos conseguido iniciar el modo anunciante.

3. El callback `mConnectionLifecycleCallback` es invocado durante el proceso de conexión de un descubridor con nosotros. Dentro sobrescribimos los siguientes métodos:
    a. `onConnectionInitiated()`: el anunciante es avisado a través de este método de que un descubridor solicita conectar. En nuestro código aceptamos automáticamente esta conexión, pasando como parámetro el objeto callback `mPayloadCallback`, que será donde se gestionen las transferencias de datos. Este también sería el lugar idóneo para realizar un proceso de autenticación: ambos extremos reciben un mismo token de conexión en este método, por lo que pueden mostrarlo al usuario para confirmar que es el mismo en ambos dispositivos, por ejemplo a través de la pantalla. La autenticación es opcional, pero el canal que se cree siempre estará encriptado.
    b. `onConnectionResult()`: se invoca este método cuando el proceso de conexión ha finalizado, y se indica si se ha creado la conexión correctamente o si ha habido un error. Por sencillez, cuando la conexión ha sido correcta, dejamos de anunciarnos para que no conecten nuevos clientes.
    c. `onDisconnect()`: se invoca este método cuando la comunicación entre ambos dispositivos se pierde antes de poder finalizar el proceso completo de conexión. Lo utilizaremos para volver a activar el modo anunciante.
4. El callback `mPayloadCallback` es invocado cuando se reciben datos por una conexión establecida con otro dispositivo. El método `onPayloadTransferUpdate()` nos indica el estado del proceso de transferencia, lo que nos permite mostrar el porcentaje transferido al usuario, o detectar posibles errores. Sin embargo, esto solo es útil para las transferencias de tipo FILE, y nosotros utilizaremos el tipo BYTE. Por otro lado, el método `onPayloadReceived()` recibe la carga tipo BYTE que hemos enviado desde el móvil. Aquí comprobamos el mensaje recibido para encender o apagar el LED de la RP3 mediante `doRemoteAction()`. Cuando recibimos el mensaje, cerramos la conexión con `disconnect()`.
5. El método `disconnect()` lo utilizamos para cerrar nuestro lado de la conexión, lo cual será notificado al descubridor, que cerrará automáticamente su lado. Realizamos la desconexión en el receptor del mensaje para asegurarnos de que hemos recibido la transferencia correctamente antes de liberar la conexión.

6. Pasamos ahora a trabajar con el módulo para el terminal móvil. Comenzaremos con el layout de la aplicación. Dirígete a mobile > res > layout > activity main.xml, y en modo texto, utiliza el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Pulse el botón para comenzar"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.3" />
    <Button
        android:id="@+id/buttonLED"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Acción!"
        style="@style/Widget.AppCompat.Button.Colored"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.6" />
</android.support.constraint.ConstraintLayout>
```

Este layout estará formado por una etiqueta de texto (textView1), que nos dará feedback sobre el estado actual, y un botón (buttonLED) que lanzará las acciones.

7. A continuación, abre la clase `MainActivity` del módulo **mobile**, y añade el siguiente contenido:

```
public class MainActivity extends AppCompatActivity {
 private static final int MY_PERMISSIONS_REQUEST_ACCESS_COARSE_LOCATION=1;
  // Consejo: utiLiza como SERVICE_ID eL nombre de tu paquete
  private static final String SERVICE_ID = "com.example.mipaquete";
  private static final String TAG = "Mobile:";
  Button botonLED;
  TextView textview;

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    textview = (TextView) findViewById(R.id.textViewl);
    botonLED = (Button) findViewById(R.id.buttonLED);
```

```
  botonLED.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
      Log.i(TAG, "Boton presionado");
      startDiscovery();
      textview.setText("Buscando...");
    }
  });

  // Comprobación de permisos peligrosos
  if (ContextCompat.checkSelfPermission(this,
    Manifest.permission.ACCESS_COARSE_LOCATION)
    != PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this,
      new String[]{Manifest.permission.ACCESS_COARSE_LOCATION},
      MY_PERMISSIONS_REQUEST_ACCESS_COARSE_LOCATION);
  }
}

// Gestión de permisos
@Override public void onRequestPermissionsResult(int requestCode,
                  String permissions[], int[] grantResults) {
  switch (requestCode) {
    case MY_PERMISSIONS_REQUEST_ACCESS_COARSE_LOCATION: {
      if (grantResults.length > 0
        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
        Log.i(TAG, "Permisos concedidos");
      } else {
        Log.i(TAG, "Permisos denegados");
        textview.setText("Debe aceptar los permisos para comenzar");
        botonLED.setEnabled(false);
      }
      return;
    }
  }
}

private void startDiscovery() {
  Nearby.getConnectionsClient(this).startDiscovery(SERVICE_ID,
    mEndpointDiscoveryCallback, new DiscoveryOptions(Strategy.P2P_STAR))
    .addOnSuccessListener(new OnSuccessListener<Void>() {
      @Override public void onSuccess(Void unusedResult) {
        Log.i(TAG, "Estamos en modo descubrimiento!");
      }
    })
    .addOnFailureListener(new OnFailureListener()  {
      @Override public void onFailure(@NonNull Exception e) {
        Log.e(TAG, "Modo descubrimiento no iniciado.", e);
      }
    });
}

private void stopDiscovery() {
```

```
    Nearby.getConnectionsClient(this).stopDiscovery();
    Log.i(TAG, "Se ha detenido el modo descubrimiento.");
  }

  private final EndpointDiscoveryCallback mEndpointDiscoveryCallback =
    new EndpointDiscoveryCallback() {

      @Override public void onEndpointFound(String endpointId,
                           DiscoveredEndpointInfo discoveredEndpointInfo) {
        Log.i(TAG, "Descubierto dispositivo con Id: " + endpointId);
        textview.setText("Descubierto: " + discoveredEndpointInfo
               .getEndpointName());
        stopDiscovery();
        // Iniciamos La conexión con aL anunciante "Nearby LED"
        Log.i(TAG, "Conectando...");
        Nearby.getConnectionsClient(getApplicationContext())
               .requestConnection("Nearby LED",endpointId,
          mConnectionLifecycleCallback)
          .addOnSuccessListener(new OnSuccessListener<Void>() {
            @Override public void onSuccess(Void unusedResult) {
              Log.i(TAG, "Solicitud lanzada, falta que ambos " +
                "lados acepten");
            }
          })
          .addOnFailureListener(new OnFailureListener()  {
            @Override public void onFailure(@NonNull Exception e) {
              Log.e(TAG, "Error en solicitud de conexión", e);
              textview.setText("Desconectado");
            }
          });
      }

      @Override public void onEndpointLost(String endpointId) {}
    };

  private final ConnectionLifecycleCallback mConnectionLifecycleCallback =
    new ConnectionLifecycleCallback() {

      @Override public void onConnectionInitiated(
        String endpointId, ConnectionInfo connectionInfo) {
        // Aceptamos La conexión automáticamente en ambos Lados.
        Log.i(TAG, "Aceptando conexión entrante sin autenticación");
        Nearby.getConnectionsClient(getApplicationContext())
               .acceptConnection(endpointId, mPayloadCallback);
      }

      @Override public void onConnectionResult(String endpointId,
                                       ConnectionResolution result) {
        switch (result.getStatus().getStatusCode()) {
          case ConnectionsStatusCodes.STATUS_OK:
            Log.i(TAG, "Estamos conectados!");
            textview.setText("Conectado");
            sendData(endpointId, "ACTION");
```

```
            break;
          case ConnectionsStatusCodes.STATUS_CONNECTION_REJECTED:
            Log.i(TAG, "Conexión rechazada por uno o ambos lados");
            textview.setText("Desconectado");
            break;
          case ConnectionsStatusCodes.STATUS_ERROR:
            Log.i(TAG, "Conexión perdida antes de poder ser " +
                    "aceptada");
            textview.setText("Desconectado");
            break;
        }
      }

      @Override public void onDisconnected(String endpointId) {
        Log.i(TAG, "Desconexión del endpoint, no se pueden " +
               "intercambiar más datos.");
        textview.setText("Desconectado");
      }
    };

  private final PayloadCallback mPayloadCallback = new PayloadCallback() {
    // En este ejemplo, el móvil no recibirá transmisiones de la RP3
    @Override public void onPayloadReceived(String endpointId,
                                Payload payload) {
      // Payload recibido
    }

    @Override public void onPayloadTransferUpdate(String endpointId,
                            PayloadTransferUpdate update) {
      // Actualizaciones sobre el proceso de transferencia
    }
  };

  private void sendData(String endpointId, String mensaje) {
    textview.setText("Transfiriendo...");
    Payload data = null;
    try {
      data = Payload.fromBytes(mensaje.getBytes("UTF-8"));
    } catch (UnsupportedEncodingException e) {
      Log.e(TAG, "Error en la codificación del mensaje.", e);
    }
    Nearby.getConnectionsClient(this).sendPayload(endpointId, data);
    Log.i(TAG, "Mensaje enviado.");
  }
}
```

Puesto que necesitamos el permiso `ACCESS_COARSE_LOCATION`, y este se considera un permiso peligroso, deberemos realizar la solicitud al usuario en tiempo de ejecución. Para ello, utilizamos `checkSelfPermission()` y `requestPermissions()` en el método `onCreate()`. Por otro lado, utilizamos `extends` para heredar de la clase `AppCompatActivity`, y sobrescribimos el método `onRequestPermissionsResult()`, de forma que podamos realizar distintas acciones si disponemos del permiso o no. En este caso, y por

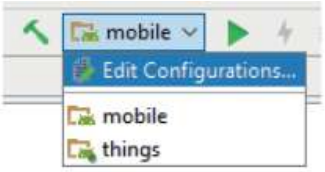
sencillez, si no se dispone del permiso se desactiva el botón, y se muestra un texto informativo al usuario.

Una vez resuelta la gestión de permisos peligrosos, el resto de aplicación funciona de la siguiente forma:

1. En el método `onCreate()` añadimos un listener para detectar el clic del botón. Cuando se realiza, se lanza el método `startDiscovery()`.
2. En el método `startDiscovery()`, llamamos a la librería Nearby Connections para comenzar el modo descubridor, pasando como parámetros el identificador de servicio (el mismo que el utilizado en el módulo de **things**), el objeto callback `mEndpointDiscoveryCallback` que gestionará todo el proceso de descubrimiento de anunciantes, y la estrategia a utilizar (que será la misma que hemos utilizado en el módulo de **things**, es decir, **P2P_STAR**).
3. El callback `mEndpointDiscoveryCallback` es invocado cada vez que se detecta un anunciante (lanzando el método `onEndpointFound()`) o que desaparece (`onEndpointLost()`). Estos anunciantes se registran con un endpointID, que los identifica de forma unívoca. En nuestro caso, y por sencillez, cuando detectemos un anunciante con nuestro identificador de servicio podremos asumir que es el de nuestra RP3, así que conectaremos directamente con él mediante `requestConnection()` y pasaremos el nombre de la baliza de anuncio (el mismo que el utilizado en el módulo de **things**, **Nearby LED**), el endpointID que tiene asociado, y el callback `mConnectionLifecycleCallback` para gestionar el proceso de conexión. Añadimos, además, dos listeners para detectar si hemos conseguido enviar correctamente esta solicitud de conexión.
4. El callback `mConnectionLifecycleCallback` es invocado durante el proceso de conexión con el anunciante. Dentro sobrescribimos los siguientes métodos:
    a. `onConnectionInitiated()`: aunque nosotros lancemos la solicitud de conexión, ambos lados deben aceptarla. Recibiremos un token de seguridad para decidir si queremos conectar o no con el anunciante. En nuestro caso, aceptaremos la conexión automáticamente sin realizar autenticación, pasando como parámetro el objeto callback `mPayloadCallback`, que será el que gestione las transferencias de datos.
    b. `onConnectionResult()`: se invoca este método cuando el proceso de conexión ha finalizado, y se indica si se ha creado la conexión correctamente o si ha habido un error. Si la conexión es correcta, utilizamos el método `sendData()` para enviar el mensaje «ACCIÓN» a la RP3.
    c. `onDisconnect()`: se invoca este método cuando la comunicación entre ambos dispositivos se pierde antes de poder finalizar el proceso completo de conexión.
5. El callback `mPayloadCallback` es invocado cuando se reciben datos del anunciante. En este ejercicio no recibiremos nada, pero lo dejaremos como referencia.

6. El método `sendData()` invoca al método `sendPayload()` de la librería Nearby Connections para enviar el mensaje «ACCIÓN» al anunciante.

8. Carga y lanza cada módulo en su dispositivo de destino. Deberás seleccionar primero uno de los dos módulos, presionar en Run y, cuando acabe, hacer lo mismo con el otro módulo.



9. Comprueba que funciona correctamente la aplicación, y que el LED de la RP3 se enciende y se apaga cuando se pulsa el botón del terminal móvil.

Aunque este es un ejercicio muy sencillo, habrás detectado varias posibilidades de mejora para que el sistema sea más versátil y eficiente:

- Por un lado, estamos conectando nuestro descubridor a una baliza Nearby predefinida, de nombre «Nearby LED». Sin embargo, el sistema sería más versátil si, al entrar en el modo descubridor, mostrara en pantalla una lista de las balizas encontradas y fuera el usuario quien seleccionarara a qué baliza conectar.

- Por otro lado, cada vez que pulsamos el botón Conmutar LED realizamos un proceso que incluye la búsqueda de nuestra baliza «Nearby LED», la conexión, la transferencia y la desconexión. Si el accionamiento del botón es ocasional, no supone un problema, pero si queremos realizar acciones más seguidas, esto supone una pérdida de eficiencia. Si pruebas a encender y apagar varias veces el LED, verás que hay momentos en que la comunicación se alarga incluso hasta decenas de segundos (activación y desactivación de wifi/bluetooth, búsqueda y conexión de dispositivos ¯_(ツ)_/¯). Si quisiéramos utilizar Nearby Connections como control remoto de un robot hecho con Android Things, esto sería excesivamente limitante.

Práctica: Conectividad avanzada con Nearby Connections

Tras realizar el ejercicio anterior, modifica el proyecto para eliminar los 2 problemas que acabamos de comentar:

a) Primero, modifica el módulo **mobile** para que, en lugar de realizar todas las acciones al pulsar el botón Conmutar LED, tengamos los siguientes botones: Scan (para buscar anunciantes), Connect (para conectar con Nearby LED), ON (para encender el LED), OFF (para apagar el LED) y Disconnect (para cerrar la conexión con el anunciante). Ten en cuenta que esto significa que también deberás modificar el código del módulo **things** (ahora ya no hay un mensaje ACCIÓN, sino un mensaje ON y un mensaje OFF), y que la función disconnect que utilizaba el anunciante, ahora la utilizará el descubridor.

b) A continuación, modifica el módulo **mobile** para que la baliza a la que nos conectamos no sea «Nearby LED», sino el nombre de la primera baliza que recibamos en modo descubrimiento. Opcional: muestra en la app móvil un listview de las balizas detectadas, y cuando pulses sobre una de ellas, inicia el proceso de conexión con ella (ya no será necesario el botón Connect).

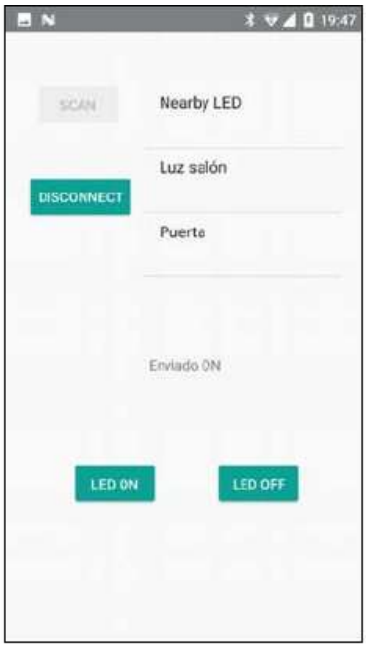
Lanza el nuevo proyecto. Verás cómo la activación y desactivación del LED ahora es instantánea, lo que nos abre la posibilidad de realizar multitud de proyectos de control en tiempo real. A continuación, se muestra un posible ejemplo de cómo quedaría esta aplicación:

Pantalla inicial

Al pulsar SCAN

Clic sobre Nearby LED

Conexión correcta



Al pulsar LED ON

Clic sobre DISCONNECT

Como hemos visto, Nearby Connections es una tecnología muy interesante para realizar todo tipo de interacciones offline, ya que resulta imbatible a la hora de reducir la latencia y aumentar el ancho de banda en las comunicaciones entre dispositivos cercanos. Pero presenta, además, una ventaja fundamental para el universo IoT. Y es que el primer paso a realizar cuando tenemos un nuevo dispositivo inteligente para nuestro hogar es conectarlo a la red inalámbrica, pero la falta de pantallas y teclados es un problema para esta primera configuración. En estas situaciones, realizar una app móvil que permite al usuario emparejarse con el dispositivo IoT para indicarle a qué wifi conectarse y con qué contraseña simplifica enormemente el proceso.

Ejercicio: Configuración remota del wifi del dispositivo IoT

En este ejercicio experimentarás con el uso de Nearby Connections para configurar dispositivos IoT cercanos. En este caso, tendremos una app móvil que buscará balizas Nearby, se conectará a ellas, y añadirá la configuración wifi de tu punto de acceso doméstico a la lista de redes de la RP3. Para ello necesitarás tener a mano el nombre de tu punto de acceso y la contraseña, o crear un nuevo AP con un terminal móvil (distinto al que vayas a utilizar en este ejercicio).

Al añadir una red wifi, esta permanece en la lista de redes, y la RP3 se conectará siempre a ella, estemos en la app que estemos. Por lo tanto, esta es una configuración que afecta al S.O. de nuestra RP3, y no solo a nuestra app; es una configuración global a nivel de sistema.

> **Nota:** según la gestión actual de permisos de Android, solo la app que ha creado una nueva entrada en la lista de redes wifi puede posteriormente eliminarla.

1. Comienza a partir del ejercicio anterior, donde utilizábamos Nearby Connections para actuar sobre el LED de la RP3.
2. En el módulo **things**, añade una nueva clase (botón derecho en MainActivity, y luego haz clic en New > J ava Class). Utiliza como nombre **WifiUtils**.
3. Utiliza el siguiente código para esta nueva clase:

```
public class WifiUtils {
  WifiManager wifiManager;
  WifiConfiguration wifiConfig;
  private static final String TAG = "WifiUtils";
  Context context;

  public WifiUtils(Context context) {
    this.context = context;
    wifiManager = (WifiManager) context.getSystemService(Context
         .WIFI_SERVICE);
    wifiConfig = new WifiConfiguration();
  }

  public void listNetworks() {
    List<WifiConfiguration> redes = wifiManager.getConfiguredNetworks();
    Log.i(TAG, "Lista de redes configuradas:\n " + redes.toString());
  }

  public String getConnectionInfo() {
    Log.i(TAG, "Red actual: " + wifiManager.getConnectionInfo()
      .toString());
    return new String(wifiManager.getConnectionInfo().getSSID() + ", " +
      wifiManager.getConnectionInfo().getLinkSpeed() + " Mbps, {RSSI: " +
      wifiManager.getConnectionInfo().getRssi() + ")");
  }

  public void removeAllAPs() {
    // SoLo se pueden eLiminar Las redes que haya creado esta app!!
    // Si eL resuLtado es faLseJ no se ha eLiminado
    List<WifiConfiguration> redes = wifiManager.getConfiguredNetworks();
    wifiManager.disconnect();
    for (WifiConfiguration red : redes) {
       Log.i(TAG, "Intento de eliminar red " + red.SSID + " con " +
            "resultado " + wifiManager.removeNetwork(red.networkId));
    }
    wifiManager.reconnect();
  }

  public int connectToAP(String networkSSID, String networkPasskey) {
    WifiManager wifiManager = (WifiManager) context.getSystemService
         (Context.WIFI_SERVICE);
    for (ScanResult result : wifiManager.getScanResults()) {
      if (result.SSID.equals(networkSSID)) {
        String securityMode = getScanResultSecurity(result);
        WifiConfiguration wifiConfiguration = createAPConfiguration
               (networkSSID, networkPasskey, securityMode);
        int res = wifiManager.addNetwork(wifiConfiguration);
```

```
        Log.i(TAG, "Intento de añadir red: " + res);
        boolean b = wifiManager.enableNetwork(res, true);
        Log.i(TAG, "Intento de activar red: " + b);
        wifiManager.setWifiEnabled(true);
        boolean changeHappen = wifiManager.saveConfiguration();
        if (res != -1 && changeHappen) {
          Log.i(TAG, "Cambio de red correcto: " + networkSSID);
        } else {
          Log.i(TAG, "Cambio de red erróneo.");
        }
        return res;
      }
    }
    return -1;
  }

  private String getScanResultSecurity(ScanResult scanResult) {
    final String cap = scanResult.capabilities;
    final String[] securityModes = {"WEP", "PSK", "EAP"};
    for (int i = securityModes.length - 1; i >= 0; i--) {
      if (cap.contains(securityModes[i])) {
        return securityModes[i];
      }
    }
    return "OPEN";
  }

  private WifiConfiguration createAPConfiguration(String networkSSID,
                        String networkPasskey, String securityMode) {
    WifiConfiguration wifiConfiguration = new WifiConfiguration();
    wifiConfiguration.SSID = "\"" + networkSSID + "\"";
    if (securityMode.equalsIgnoreCase("OPEN")) {
      wifiConfiguration.allowedKeyManagement.set(WifiConfiguration
            .KeyMgmt.NONE);
    } else if (securityMode.equalsIgnoreCase("WEP")) {
      wifiConfiguration.wepKeys[0] = "\"" + networkPasskey + "\"";
      wifiConfiguration.wepTxKeyIndex = 0;
      wifiConfiguration.allowedKeyManagement.set(WifiConfiguration
            .KeyMgmt.NONE);
      wifiConfiguration.allowedGroupCiphers.set(WifiConfiguration
            .GroupCipher.WEP40);
    } else if (securityMode.equalsIgnoreCase("PSK")) {
      wifiConfiguration.preSharedKey = "\"" + networkPasskey + "\"";
      wifiConfiguration.hiddenSSID = true;
      wifiConfiguration.status = WifiConfiguration.Status.ENABLED;
      wifiConfiguration.allowedGroupCiphers.set(WifiConfiguration
            .GroupCipher.TKIP);
      wifiConfiguration.allowedGroupCiphers.set(WifiConfiguration
            .GroupCipher.CCMP);
      wifiConfiguration.allowedKeyManagement.set(WifiConfiguration
            .KeyMgmt.WPA_PSK);
      wifiConfiguration.allowedPairwiseCiphers.set(WifiConfiguration
            .PairwiseCipher.TKIP);
```

```
        wifiConfiguration.allowedPairwiseCiphers.set(WifiConfiguration
                .PairwiseCipher.CCMP);
        wifiConfiguration.allowedProtocols.set(WifiConfiguration.Protocol
                .RSN);
        wifiConfiguration.allowedProtocols.set(WifiConfiguration.Protocol
                .WPA);
      } else {
        Log.i(TAG, "Modo de seguridad no soportado: " + securityMode);
        return null;
      }
      return wifiConfiguration;
    }
}
```

En esta clase tenemos las herramientas básicas para manipular las redes wifi de nuestra RP3:

1. El método `listNetworks()` nos muestra por el Logcat el listado de redes que tenemos configuradas en la RP3. Lo utilizaremos para hacer pruebas durante el desarrollo.
2. El método `getConnectionInfo()` nos muestra la información de la conexión wifi actual. En el LogCat nos muestra la información completa pero, además, este método nos devuelve un string con información simplificada, ideal para mostrar al usuario en la app móvil.
3. El método `removeAllAPs()` elimina todas las redes inalámbricas que hayan sido configuradas desde esta app.
4. El método `connectToAP()` utiliza como parámetros de entrada el nombre de la red wifi (o SSID) y su contraseña. Este método se encarga de comprobar el método de seguridad que utiliza y, si es compatible, añade la red a la lista de redes. Tras ello, trata de conectar con esta nueva red wifi.
5. Podemos encontrar, además, un par de métodos privados (no visibles desde fuera de esta clase), que se utilizan internamente para gestionar la seguridad del punto de acceso a la hora de añadirlo a la lista de redes.

4. Abre ahora la clase `MainActivity` del módulo **things**, y añade las siguientes líneas para crear un objeto `WifiUtil`:

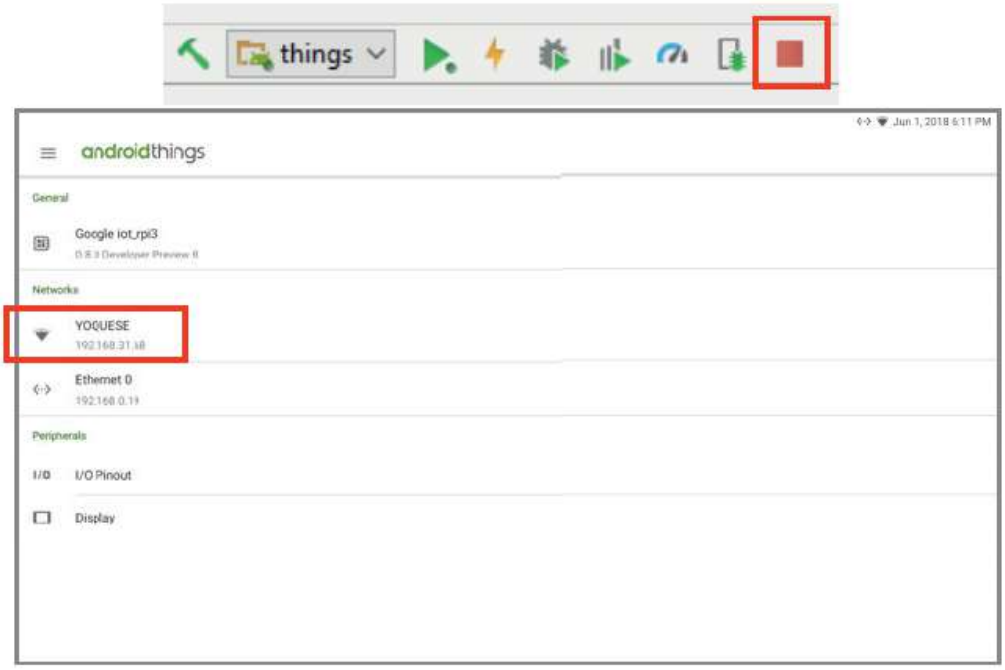
```
public class MainActivity extends Activity {

    private WifiUtils wifiutils;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        wifiutils = new WifiUtils(this);
        .
```

5. Dirígete ahora al método `doRemoteAction()` y modifica su contenido por el siguiente:

```
public void doRemoteAction() {
    wifiutils.connectToAP("SSID", "pass");
    wifiutils.listNetworks();
    wifiutils.getConnectionInfo();
}
```

Modifica el texto “SSID” y “pass” por el nombre de tu red wifi y la contraseña, respectivamente. Este sistema debería funcionar sin problemas en redes WEP, WPA y WPA2-PSK, así como en redes sin contraseña. Tras añadir esta red y conectar con ella, se muestra la lista de redes configuradas por LogCat, e información de la conexión actual.

6. Carga el proyecto y comprueba que, al pulsar en el botón «ACCIÓN!» del móvil, el LogCat de la RP3 muestra la red añadida. Si, además, se ha conectado correctamente, aparecerá la información de esta conexión. Otra forma de comprobar que ha conectado correctamente, y que este cambio es a nivel de sistema, es parar la ejecución de la app actual (con el botón de stop) y, con una pantalla conectada al HDMI de la RP3, revisar las conexiones actuales:



Hemos realizado un ejercicio muy sencillo. En un proyecto más completo tendríamos un wizard que, al abrirse, buscaría dispositivos IoT cercanos para configurar. El usuario seleccionaría el dispositivo deseado de esta lista, estableciéndose la conexión Nearby entre ambos. Una vez conectados, la app mostraría un listado de las redes wifi al alcance. El usuario seleccionaría una de estas redes, introduciría su contraseña y estos parámetros se enviarían al dispositivo IoT. Si la configuración wifi fuera correcta y la conexión hubiera tenido éxito, se informaría a la app móvil de que el proceso ha finalizado, y se mostrarían datos de esta conexión (velocidad o nivel de señal).

Práctica: Configurador wifi avanzado

Tras realizar el ejercicio anterior, modifica el proyecto para darle más flexibilidad y potencia, eligiendo al menos 3 de las siguientes opciones:

a) Al igual que en la práctica anterior, permite que sea el usuario el que controle la conexión Nearby (escanear, elegir a qué baliza conectar, y desconectar cuando desee).

b) Haz que la información de la red a configurar (SSID y pass) la envíe el terminal móvil, en lugar de estar fija en el código de la RP3.

c) Añade un botón para eliminar todas las redes wifi que haya configuradas en la RP3.

d) Haz que la app móvil escanee las redes wifi de su alrededor y que las muestre al usuario. El usuario seleccionará una de ellas, escribirá su contraseña, y esta información (SSID y pass) se enviará a la RP3.

e) Añade un botón para mostrar en la app móvil información sobre la red wifi a la que el usuario está conectado.

f) Añade un botón para mostrar en la app móvil la lista de redes (SSID) que hay configuradas en la RP3.

Preguntas de repaso: API Nearby Communications

5.3 Comunicaciones online

Tal y como su propio nombre indica, los dispositivos de Internet de las cosas alcanzan su mayor potencial cuando se conectan a la red por excelencia: **Internet**. Esta conectividad permite acceder al equipo remotamente desde cualquier ubicación, en cualquier momento y a través de cualquier tipo de red, siempre que ambos extremos estén conectados a Internet. El dispositivo IoT puede utilizar esta conexión tanto para ofrecer servicios a la red como para consumirlos, obteniendo a su vez información de otros nodos IoT, aplicaciones móviles, bases de datos privadas, servicios de información pública (meteorología, tráfico, calidad del aire…), etc. Esta conexión a Internet abre una nueva dimensión de servicios para nuestras «cosas».

En este capítulo vamos a explorar las opciones más relevantes para conectar nuestra Raspberry Pi 3 a Internet e intercambiar servicios con fuentes externas que nos permitan interactuar con ella.

Preguntas de repaso: Comunicaciones online

5.3.1. Servidor web en Android Things

Las páginas web son documentos de hipertexto o hipermedios (texto, imágenes, vídeos…) que nos permiten interaccionar con sistemas remotos para leer noticias, ver anuncios y contenido multimedia, realizar compras online, etc. Su gran versatilidad y universalidad han llevado también al éxito de las aplicaciones web, soluciones software que se ejecutan desde un navegador web, de forma que ya no es necesario tener la aplicación instalada, sino solo un navegador con acceso a Internet.

Como interfaz con un sistema físico, la web nos aporta este mismo beneficio: no es necesario instalar una aplicación, desde cualquier sistema informático con acceso a Internet y un navegador podremos acceder y manipular remotamente nuestro sistema.

La Raspberry Pi 3 tiene la suficiente potencia para albergar un sencillo servidor web HTTP, en el que ofrecer una interfaz web con la que controlar el funcionamiento de los sistemas físicos que conectemos. Para ello, necesitamos seguir los siguientes pasos:

- Montar un servidor web HTTP que gestione las peticiones entrantes.

- Crear una página HTML que contenga la interfaz de usuario a mostrar (controles, visualizadores, etc.).
- Enlazar el funcionamiento del servidor HTTP con las funciones de nuestro código.

Ejercicio: Implementación de servidor HTTP en la RP3

En este ejercicio aprenderás a implementar un servidor web HTTP sencillo en la Raspberry Pi 3, para utilizarlo como interfaz de usuario remota desde cualquier sistema con acceso a Internet. Para ello utilizaremos el projecto NanoHttpd (https://github.com/NanoHttpd/nanohttpd), un servidor HTTP ligero que nos permite realizar proyectos web con un mínimo consumo de recursos.

1. Crea un nuevo proyecto de Android Things.
2. Añade la siguiente dependencia en build.gradle (Module: app):

```
dependencies {
    .
    implementation 'org.nanohttpd:nanohttpd:2.3.1'
}
```

3. Añade los permisos necesarios en el manifiesto (AndroidManifest.xml). Utilizaremos `INTERNET` para poder abrir el socket donde escuchará el servidor web, y `USE_PERIPHERAL_IO` para utilizar las E/S de nuestro sistema:

```
<manifest .>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="com.google.android.things.permission
                                .USE_PERIPHERAL_IO" />
    <application .>
```

4. Crea una nueva clase llamada `WebServer` para recoger los detalles de implementación del servidor y web y manejar las solicitudes entrantes. Esta clase deberá heredar de `NanoHTTPD` y tener el siguiente contenido:

```
public class WebServer extends NanoHTTPD {
  Context ctx;

  public interface WebserverListener {
    Boolean getLedStatus();
    void switchLEDon();
    void switchLEDoff();
  }

  private WebserverListener listener;

  public WebServer(int port, Context ctx, WebserverListener listener) {
    super(port);
    this.ctx = ctx;
    this.listener = listener;
    try {
      start();
```

```
      Log.i(TAG, "Webserver iniciado");
    } catch (IOException ioe) {
      Log.e(TAG, "No ha sido posible iniciar el webserver", ioe);
    }
  }

  private StringBuffer readFile() {
    BufferedReader reader = null;
    StringBuffer buffer = new StringBuffer();
    try {
      reader = new BufferedReader(new InputStreamReader(
        ctx.getAssets().open("home.html"),"UTF-8"));
      String mLine;
      while ((mLine = reader.readLine()) != null) {
        buffer.append(mLine);
        buffer.append("\n");
      }
    } catch (IOException ioe) {
      Log.e(TAG, "Error leyendo la página home", ioe);
    } finally {
      if (reader != null) {
        try {
          reader.close();
        } catch (IOException e) {
          Log.e(TAG, "Error cerrando el reader", e);
        } finally {
          reader = null;
        }
      }
    }
    return buffer;
  }

  @Override
  public Response serve(IHTTPSession session) {
    Map<String, List<String>> parms = session.getParameters();
    // AnaLizamos Los parámetros que ha modificado eL usuario
    // Según estos parámetrosJ ejecutamos acciones en La RP3
    if (parms.get("on") != null) {
      listener.switchLEDon();
    } else if (parms.get("off") != null) {
      listener.switchLEDoff();
    }
    // Obtenemos La web originaL
    String preweb = readFile().toString();
    // Si queremos mostrar aLgún vaLor de saLidaJ La modificamos
    // En este casoJ sustituimos paLabras cLave por strings
    String postweb;
    if (listener.getLedStatus()) {
      postweb = preweb.replaceAll("#keytext", "ENCENDIDO");
      postweb = postweb.replaceAll("#keycolor", "MediumSeaGreen");
      postweb = postweb.replaceAll("#colorA", "#F2994A");
      postweb = postweb.replaceAll("#colorB", "#F2C94C");
```

```
    } else {
      postweb = preweb.replaceAll("#keytext", "APAGADO");
      postweb = postweb.replaceAll("#keycolor", "Tomato");
      postweb = postweb.replaceAll("#colorA", "#3e5151");
      postweb = postweb.replaceAll("#colorB", "#decba4");
    }
    return newFixedLengthResponse(postweb);
  }
}
```

En este código vemos cómo es posible disponer de un servidor web en muy pocos pasos:

1. Creamos una interfaz `WebserverListener` con las funciones que ejecutaremos mediante la interfaz web. En este caso `SwitchLedON` y `SwitchLedOFF`.
2. Creamos un constructor que lanza el servidor mediante el puerto suministrado. El método `start()` inicia el servidor web.
3. Con @Override sobrescribimos el método `serve`, de forma que podamos interceder en las peticiones del usuario e implementar nuestra lógica de control. Su contenido es muy sencillo; si la petición contiene alguno de los parámetros asociados a los controles de la web, leemos su contenido y actuamos en consecuencia. Si queremos lanzar alguna acción, llamamos al método correspondiente del objeto `WebserverListener`. Por el contrario, si queremos mostrar algún nuevo valor en la web (como el de una entrada de la Raspberry Pi 3), editaremos el html de salida antes de enviarlo. En este caso, el html contendrá etiquetas clave (como "#keytext"), que al ser procesadas por este método, se modificarán al texto correpondiente ("ENCENDIDO" o "APAGADO").
4. Con el objeto `readfile` obtenemos el contenido de la web a mostrar (**home.html**), almacenándolo en un StringBuffer listo para ser servido como respuesta a una petición del navegador.

5. Vamos a almacenar la web html en una carpeta de recursos «Assets» de Android. Para crearla pulsa con el botón derecho sobre la carpeta app, y luego ve a New > Folder > Assets Folder.
6. Crea un fichero html con el siguiente contenido, y añádelo a la carpeta de Assets:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
  <head>
    <title>Webserver para control remoto del LED</title>
    <style type="text/css">
        * {
         font-family: arial;
         color: white;
        }
        body {
          background: #colorA;
          background: -webkit-linear-gradient(to right, #colorA, #colorB);
          background: linear-gradient(to right, #colorA, #colorB);
        }
```

```
        .btn {
          border: none;
          color: white;
          padding: 14px 28px;
          cursor: pointer;
        }
        .on {background-color: #2196F3;}
        .on:hover {background: #0b7dda;}
        .off {background-color: #e7e7e7; color: black;}
        .off:hover {background: #ddd;}
    </style>
    <script type="text/javascript">
          function setON() { window.location = '?on=true'; }
          function setOFF() { window.location = '?off=true'; }
    </script>
  </head>
  <body align="center">
    <h1><br><br>Control remoto de LED<br></h1>
    Estado actual:
    <strong><small><span style="color:#keycolor;">#keytext</span></small>
    </strong><br><br>Lanzar acción<br><br>
    <button class="btn on" onclick="setON{};">ON</button>
    <button class="btn off" onclick="setOFF{};">OFF</button>
  </body>
</html>
```

Vemos que es posible generar una página con estilos CSS, puesto que el contenido del html es simplemente texto plano que interpretará el navegador. En este código se generan 2 botones, que al pulsarse llaman a las funciones javascript "`setON()`" y "`setOFF()`". Estas funciones simplemente añaden parámetros a la dirección web de consulta, de forma que podamos saber qué ha pulsado el usuario. También podemos ver algunas de las palabras clave que se modificarán cuando se genere el html para el cliente, como "**#keycolor**" o "**#keytext**".

7. Ya tenemos la implementación del servidor web; ahora hay que integrarla en nuestra app de Android Things. Desde aquí podremos iniciar y parar el servidor web, e implementar la lógica del listener para tomar acciones cuando el usuario nos envía datos a través de la página HTML. Abre la clase `MainActivity` y añade el siguiente contenido:

```
public class MainActivity extends Activity implements WebServer
   .WebserverListener {
  private WebServer server;
  private final String PIN_LED = "BCM18";
  public Gpio mLedGpio;

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    server = new WebServer(8180, this, this);
    PeripheralManager service = PeripheralManager.getInstance();
    try {
      mLedGpio = service.openGpio(PIN_LED);
      mLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
```

```
    } catch (IOException e) {
      Log.e(TAG, "Error en el API PeripheralIO", e);
    }
  }

  @Override protected void onDestroy() {
    super.onDestroy();
    server.stop();
    if (mLedGpio != null) {
      try {
        mLedGpio.close();
      } catch (IOException e) {
        Log.e(TAG, "Error en el API PeripheralIO", e);
      } finally {
        mLedGpio = null;
      }
    }
  }

  @Override public void switchLEDon() {
    try {
      mLedGpio.setValue(true);
      Log.i(TAG, "LED switched ON");
    } catch (IOException e) {
      Log.e(TAG, "Error on PeripheralIO API", e);
    }
  }

  @Override public void switchLEDoff() {
    try {
      mLedGpio.setValue(false);
      Log.i(TAG, "LED switched OFF");
    } catch (IOException e) {
      Log.e(TAG, "Error on PeripheralIO API", e);
    }
  }

  @Override public Boolean getLedStatus() {
    try {
      return mLedGpio.getValue();
    } catch (IOException e) {
      Log.e(TAG, "Error on PeripheralIO API", e);
      return false;
    }
  }
}
```

Hay diversos detalles en este código que conviene explicar:

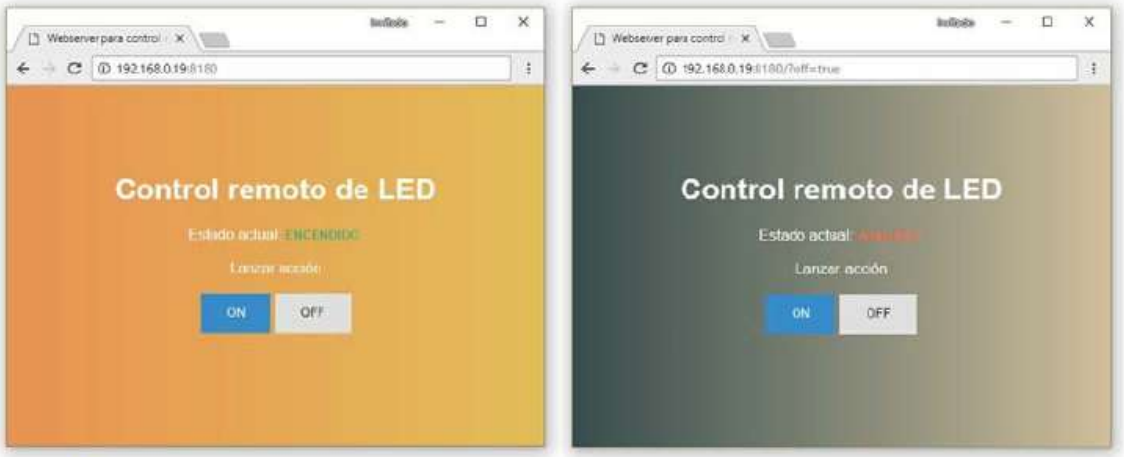
1. Nuestra clase `MainActivity` se declara especificando que implementa el `WebserverListener`, que es la interfaz de callback.
2. Al arrancar la app (`onCreate()`) se crea e inicia el servidor web en el puerto 8180, y pasamos nuestro listener como parámetro.

3. Vemos la implementación de tres métodos del callback: `switchLEDon`, `switchLEDoff` y `getLedStatus`.

8. Carga y lanza el proyecto. Para acceder a la web, utiliza el navegador web del PC con la siguiente URL (donde `android_things_ip` es la IP de tu Raspberry):

```
http://android_things_ip:8180/home.html
```

Comprueba que puedes encender y apagar el LED desde la web, y cómo cambia su contenido en función del estado actual de LED.



Práctica: Lectura remota del fotorresistor

Tras realizar el ejercicio anterior, modifica el proyecto para mostrar en la web el valor del fotorresistor.

Preguntas de repaso: Servidor web embebido en la RaspberryPi3

5.3.2. Protocolos de comunicaciones

Ya hemos visto cómo es posible montar un servidor web con Android Things, de forma que podamos servir una web html para que un usuario interactúe con nuestra RP3. Y es que HTML se creó precisamente como un lenguaje para que las máquinas pudieran interactuar con las personas, una visualización mejorada de texto, controles y multimedia para enriquecer la experiencia de usuario. Sin embargo, esta solución no nos sirve cuando queremos realizar comunicaciones más automatizadas, como comunicaciones entre máquinas (o machine-to-machine, M2M). Es el caso de querer mostrar los datos de nuestros sensores en otro tipo de sistemas que no sean una web (como otra Raspberry, aplicaciones de escritorio _), hacer públicos los datos de sensores para que cualquiera pueda desarrollar aplicaciones que los muestren o, simplemente, querer aportar más flexibilidad a la interacción con la Raspberry.

Con el objetivo de que dos elementos independientes puedan intercambiar datos y entenderse mutuamente, surgen los protocolos estándares de comunicaciones. Al implementar un mismo protocolo en dos sistemas aislados, tendremos la seguridad de que podrán intercambiar información y comprender su contenido. Los protocolos de comunicaciones son la columna vertebral de los sistemas IoT; habilitan su conectividad y la interacción de las aplicaciones.

Existen protocolos de comunicaciones a distintos niveles: capa física y de acceso al medio (wifi, Ethernet…), de red (IP), de transporte (TCP/UDP)… pero lo que estudiaremos aquí son los estándares de capa de aplicación. Estos estándares se encapsulan dentro del resto de capas, por lo que pueden circular a través de cualquier tecnología de red, y permiten que la aplicación de dos sistemas remotos pueda dialogar. En un símil con el lenguaje humano, la capa de aplicación sería el idioma, y la tecnología de acceso sería una carta, la voz humana o un email. Para la aplicación, lo importante es que el idioma sea el mismo. No importa si un extremo envía un email y el otro extremo lo escucha reproducido.

Esto nos aporta una gran versatilidad, puesto que cada dispostiivo accederá a Internet de la forma más cómoda según su situación, pero todos intercambiarán datos en un formato común que entenderán.

Los protocolos de aplicación están fuertemente ligados al modelo de comunicación utilizado. Existen muchos modelos, entre los que cabe destacar:

- **Request/response**: basado en un escenario cliente-servidor, un dispositivo cliente lanza un request al servidor, que procesa la petición y devuelve un response. Es el funcionamiento de la web, mediante el protocolo de aplicación HTTP, o su versión reducida para M2M, llamada CoAP.
- **Polling**: un elemento central maneja completamente el acceso al medio de los dispositivos remotos, que no pueden comunicarse a menos que se les dé permiso. De forma secuencial, el elemento central consulta uno a uno a todos estos dispositivos, y les pregunta si tienen algo que enviar. Si un dispositivo tiene demasiados datos que enviar, es habitual que se le solicite parar para no alterar los tiempos del resto de dispositivos, y en la siguiente vuelta se le vuelve a dar una oportunidad de continuar.
- **Publish/subscribe**: escenario «muchos a muchos», donde los dispositivos publican su información, y pueden suscribirse a información de su interés de otros dispositivos. Este tipo de escenario puede apoyarse en un bróker central que gestione las suscripciones y el reenvío de los datos (como AMQP, MQTT, XMPP), o puede ser completamente descentralizado (como DSS).
- **Push/pull**: también conocido como productor/consumidor, es un escenario basado en colas, donde los productores inyectan datos en una cola, y los consumidores recogen datos de estas colas. Las colas permiten desacoplar los productores y los consumidores, y actúan como un buffer intermedio.
- **Emparejado exclusivo**: consiste en una conexión persistente y full-duplex entre dos dispositivos (cliente-servidor), donde se puede realizar intercambio de mensajes o datos entre ambas aplicaciones. Es una conexión con estado, y el servidor es consciente de todas las conexiones que tiene activas. Es

el caso de los websockets (RFC 6455), que permite un stream de datos a través de una conexión TCP entre navegadores, apps móviles, equipos IoT, etc.

5.3.2.1. Modelo request/response: servicios web RESTful

Los servicios web son un tipo de comunicación para intercambio de datos entre aplicaciones que se apoya sobre la pila de protocolos web, y hace uso de HTTP. Al utilizar protocolos nativos para esta red, los mensajes circulan muy fácilmente a través de Internet, y se ven mínimamente afectados por cortafuegos intermedios. Por el contrario, a nivel de comunicaciones se considera poco efectivo, puesto que está basado en intercambio de texto. Los servicios web son el pilar fundamental sobre el que se construyen las aplicaciones con arquitecturas orientadas a servicios. Por ejemplo, el desarrollo de aplicaciones web suele separarse en un front (la interfaz de usuario) y un back (el servidor) conectado mediante servicios web. El servidor en esta tecnología es un servidor web.

REST (Representational State Transfer) es un conjunto de principios para diseño de servicios web, y está enfocado a los recursos de un sistema, a cómo se gestionan e intercambian sus estados entre un cliente y un servidor. Es una arquitectura sin estado, de forma que cada petición request/response debe ser independiente del resto, y debe contener toda la información necesaria para resolver esa petición.

Se conoce como servicio web RESTful a un API web implementado con HTTP y que utilice los principios REST.

5.3.2.1.1 Servidor REST

Ejercicio: API RESTful para control remoto del LED

En este ejercicio aprenderás a implementar un servidor web sencillo en la Raspberry Pi 3 para gestionar peticiones REST. Para ello utilizaremos el projecto Restlet (https://restlet.com/open-source/), un framework multiplataforma muy utilizado para el desarrollo de este tipo de APIs.

1. Crea un nuevo proyecto para Android Things.
2. Añade la siguiente dependencia en build.gradle (Module: app):

```
dependencies {
    .
    implementation 'org.restlet.android:org.restlet:2.3.7'
    implementation 'org.restlet.android:org.restlet.ext.json:2.3.7'
    implementation 'org.restlet.android:org.restlet.ext.nio:2.3.7'
}
```

3. Añade los permisos necesarios en el manifiesto (AndroidManifest.xml). Utilizaremos `INTERNET` para poder abrir el socket donde escuchará el servidor web, y `USE_PERIPHERAL_IO` para utilizar las E/S de nuestro sistema:

```
<manifest .>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="com.google.android.things.permission
                                .USE_PERIPHERAL_IO" />
    <application .>
```

4. Añade ahora una nueva clase `LEDModel`, y utiliza este código:

```
public class LEDModel {
    private static LEDModel instance = null;
    PeripheralManager service;
    private Gpio mLedGpio;
    private final String PIN LED = "BCM18";

    public static LEDModel getInstance() {
        if (instance == null) {
            instance = new LEDModel();
        }
        return instance;
    }

    private LEDModel() {
        service = PeripheralManager.getInstance();
        try {
            mLedGpio = service.openGpio(PIN LED);
            mLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
        } catch (Exception e) {
            Log.e(TAG, "Error en el API PeripheralIO", e);
        }
    }

    static Boolean setState(boolean state) {
        try {
            getInstance().mLedGpio.setValue(state);
            return true;
        } catch (IOException e) {
            Log.e(TAG, "Error en el API PeripheralIO", e);
            return false;
        }
    }

    public static boolean getState() {
        boolean value = false;
        try {
            value = getInstance().mLedGpio.getValue();
        } catch (IOException e) {
            Log.e(TAG, "Error en el API PeripheralIO", e);
        }
        return value;
    }
}
```

En esta clase modelamos nuestro LED como un objeto. El LED interactuará con el mundo externo mediante dos métodos: `getState()`, que devolverá un valor

de true o false según si está encendido o apagado; y `setState()`, al que le indicaremos el valor al que queremos que cambie. El siguiente paso es convertir estos dos métodos en dos servicios web.

5. Añade una nueva clase de nombre `LEDResource`, que herede de `org.restlet.resource.ServerResource`, y utiliza el siguiente código:

```
public class LEDResource extends ServerResource {

  @Get("json")
  public Representation getState() {
    JSONObject result = new JSONObject();
    try {
      result.put("estado", LEDModel.getState());
    } catch (Exception e) {
      Log.e(TAG, "Error en JSONObject: ", e);
    }
    return new StringRepresentation(result.toString(),
      MediaType.APPLICATION_ALL_JSON);
  }

  @Post("json")
  public Representation postState(Representation entity) {
    JSONObject query = new JSONObject();
    JSONObject fullresult = new JSONObject();
    String result;
    try {
      JsonRepresentation json = new JsonRepresentation(entity);
      query = json.getJsonObject();
      boolean state = (boolean) query.get("estado");
      Log.d(this.getClass().getSimpleName(), "Nuevo estado del LED: " +
            state);
      if (LEDModel.setState(state)) result = "ok";
      else result = "error";
    } catch (Exception e) {
      Log.e(TAG, "Error: ", e);
      result = "error";
    }
    try {
      fullresult.put("resultado", result);
    } catch (JSONException e) {
      Log.e(TAG, "Error en JSONObject: ", e);
    }
    return new StringRepresentation(fullresult.toString(),
      MediaType.APPLICATION_ALL_JSON);
  }
}
```

El objetivo de esta clase es realizar una representación RESTful del LED, es decir, conectar los servicios web del LED con los métodos que acabamos de crear para el objeto LED. Vemos que hay dos anotaciones: `@Get("json")` y `@Post("json")`. Esto significa que las peticiones GET que nos lleguen sobre el LED, y con contenido tipo JSON, utilizarán el método `getState()`, que hace una llamada al método `LEDModel.getState()`. Por otro lado, las peticiones POST utili-

zarán el método `postState()`, y harán uso del método `LEDModel.setState()` para modificar el LED.

6. En general, API REST será una funcionalidad que estará siempre activa en nuestra Raspberry, mientras que en nuestro código principal podemos dedicarnos a realizar otras tareas. La mejor forma de generar un servicio de fondo es mediante `IntentService`, puesto que nos permite manejar peticiones de forma asíncrona y bajo demanda. Para ello, añade una nueva clase `RESTfulService` que herede de `IntentService`:

```
import org.restlet.Component;
import org.restlet.data.Method;
import org.restlet.data.Protocol;
import org.restlet.engine.Engine;
import org.restlet.engine.application.CorsFilter;
import org.restlet.ext.nio.HttpServerHelper;
import org.restlet.routing.Router;
import org.restlet.service.CorsService;

public class RESTfulService extends IntentService {
  private static final String ACTION_START="com.example.mypackage.START";
  private static final String ACTION_STOP="com.example.mypackage.STOP";
  private Component mComponent;

  public RESTfulService() {
    super("RESTfulService");
    Engine.getInstance().getRegisteredServers().clear();
    Engine.getInstance().getRegisteredServers()
      .add(new HttpServerHelper(null));
    mComponent = new Component();
    Router router = new Router(mComponent.getContext()
        .createChildContext());
    // Configuración deL webserver
    mComponent.getServers().add(Protocol.HTTP, 8080);
    mComponent.getDefaultHost().attach("/rest", router);
    router.attach("/led", LEDResource.class); }

  public static void startServer(Context context) {
    Intent intent = new Intent(context, RESTfulService.class);
    intent.setAction(ACTION_START);
    context.startService(intent);
  }

  public static void stopServer(Context context) {
    Intent intent = new Intent(context, RESTfulService.class);
    intent.setAction(ACTION_STOP);
    context.startService(intent);
  }

  @Override protected void onHandleIntent(Intent intent) {
    if (intent != null) {
      final String action = intent.getAction();
      if (ACTION_START.equals(action)) {
        handleStart();
```

```
      } else if (ACTION_STOP.equals(action)) {
        handleStop();
      }
    }
  }

  private void handleStart() {
    try {
      mComponent.start();
    } catch (Exception e) {
      Log.e(getClass().getSimpleName(), e.toString());
    }
  }

  private void handleStop() {
    try {
      mComponent.stop();
    } catch (Exception e) {
      Log.e(getClass().getSimpleName(), e.toString());
    }
  }
}
```

Nota: Utiliza el import *org.restLet.ext.nio.HttpServerHeLper*.

La mayoría de contenido de esta clase está ligado al funcionamiento de `IntentService`. Como una buena práctica, sustituye "`com.example.mypackage`" por el nombre de tu paquete.

La parte más importante para el webserver del API REST la encontramos en el constructor `public RESTfulService()`, donde se realizan las siguientes configuraciones:

- El servidor se lanza a la escucha en el puerto 8080.
- Se enrutarán los mensajes que lleguen por la ruta `/rest`. Esta es una forma de utilizar el servidor web no solo como API REST sino también para alojar webs en otras rutas, así como en la raíz.
- Se añade un primer recurso para las consultas que lleguen por `/rest/led`, y se redirigen a la clase `LEDResource`. Como habíamos visto, esta clase tiene distintas acciones según le lleguen peticiones GET o POST.

7. Ahora sí, ya podemos pasar a dar contenido a nuestra `MainActivity`. Al utilizar `IntentService`, el contenido será mínimo, simplemente el necesario para arrancar y detener el servicio:

```
public class MainActivity extends Activity {
  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    RESTfulService.startServer(this); // Arrancar eL servidor
  }
  @Override protected void onDestroy() {
    super.onDestroy();
```

```
    RESTfulService.stopServer(this); // Detener eL servidor
  }
}
```

8. Finalmente, ya solo nos queda añadir el servicio `RESTfulService` en el manifiesto. Abre AndroidManifest.xml y añádelo entre las etiquetas `<application>` y `<activity>`de la siguiente forma:

```
<application .>
    <service android:name=".RESTfulService"></service>
    <activity .>
```

9. Carga y lanza el proyecto. Realizar la consulta del estado del LED es muy sencillo; al ser una llamada de tipo GET se puede hacer desde la misma barra de direcciones del navegador web del PC. Utiliza la siguiente URL (donde `android_things_ip` es la IP de tu Raspberry):

```
http://android_things_ip:8080/rest/led
```

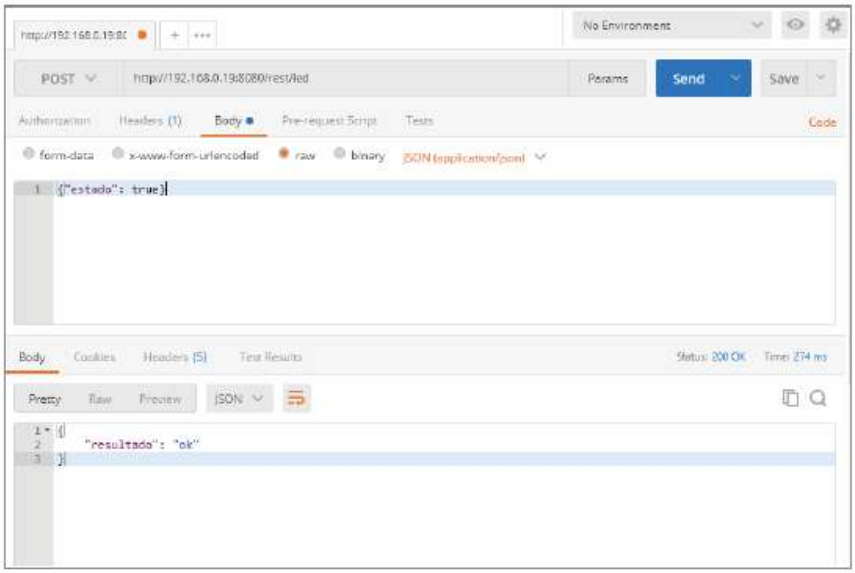
Sin embargo, para los otros métodos debemos utilizar otros sistemas, como curl (Linux), Postman (Extensión de Chrome) o Hurl.io (servicio online). En este ejercicio, explicaremos como se realizaría mediante Postman. Si no lo conoces, es la oportunidad idónea para que descubras una de las herramientas más relevantes para testear APIs web.

10. Busca Postman en las extensiones de Chrome, instálala y ábrela. Prueba a realizar la llamada anterior con GET; verás que es muy sencillo.

Para el POST, selecciona en el desplegable, a la izquierda de la barra de direcciones, el método POST. A continuación, pulsa en Body (el de la mitad superior, ya que el de la mitad inferior es para ver las respuestas del servidor), luego marca raw, y luego selecciona en el desplegable JSON (Application/json). Escribe el contenido del envío POST:

```
{"estado": true}
```

Si funciona correctamente, el LED debería encenderse y el servidor responder con un **OK**.



Ejercicio: Interacción con API REST desde una web

En este ejercicio aprenderás a desarrollar una web que lance consultas al API REST que acabamos de desarrollar en la Raspberry. Para ello vamos a desarrollar una web estática, que podremos tener en nuestro disco duro como un fichero .html que se pueda abrir en cualquier momento. Esta web mostrará un par de botones para encender y apagar el led. Es muy importante remarcar que, a diferencia del ejercicio del servidor web, esta web no está alojada en la Raspberry, sino en nuestro PC, y que la interacción se realiza totalmente mediante servicios web.

1. Abre el Notepad++ o cualquier otro editor de texto, y genera un nuevo fichero "**interruptor.html**".
2. Crea la estructura básica de un archivo html:

```
<html>
  <head>
    <title>Web de control remoto del LED</title>
  </head>
  <body>
    .
  </body>
</html>
```

3. Vamos a añadir 3 botones en la web. Uno para encender el LED, otro para apagarlo, y otro para consultar el valor actual. Añade el siguiente contenido entre las etiquetas body:

```
<body align="center">
  <h1>Control remoto de LED</h1>
  <input type="button" onClick="switchOn()" value="ON"/>
  <input type="button" onClick="switchOff()" value="OFF"/><br><br>
  <input type="button" onClick="getReading()" value="Valor actual"/>
</body>
```

   Abre la web para ver su aspecto. Los botones aún no funcionan, pero podemos ver en el código que al hacer clic sobre ellos se invocan una serie de funciones: `switchOn()`, `switchOff()` y `getReading()`.

4. Vamos a generar ahora el contenido de estas funciones, encargadas de realizar la comunicación con la RP3. Utilizaremos lenguaje javascript y la librería Ajax jquery. Añade el siguiente contenido entre las etiquetas de head:

```
<head>
  <title>Web de control remoto del LED</title>
  <script
    src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"
    type="text/javascript" charset="utf-8"></script>
  <script>
    var android_things_ip = "192.168.0.19";
    var android_things_port = "8080";
    var url = "http://" + android_things_ip + ":" + android_things_port +
      "/rest/led";
    var content_type = "application/json; charset=utf-8";
```

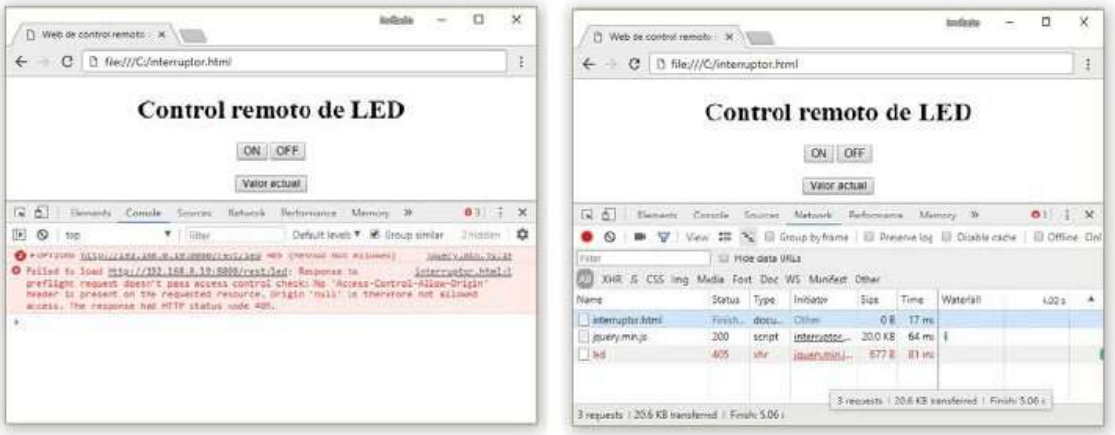
```
    function switchOn(){
      $.ajax({type: "POST", url: url, data: "{\"estado\": true}",
      contentType: content_type, dataType: "json"});
    }
    function switchOff(){
      $.ajax({type: "POST", url: url, data: "{\"estado\": false}",
      contentType: content_type, dataType: "json"});
    }
    function getReading(){
      $.ajax({type: "GET", url: url, success: callback});
    }
    function callback(data, status){
      alert(data);
    }
  </script>
</head>
```

El contenido de este script es muy sencillo: disponemos de una variable con la IP de nuestra RP3, y otra con el puerto. Modifícalas en función de tu situación. A continuación, se genera una variable url a partir de ellas, que apunta a la dirección del API REST.

Por otro lado, tenemos las tres funciones de los botones que aparecen en la web:

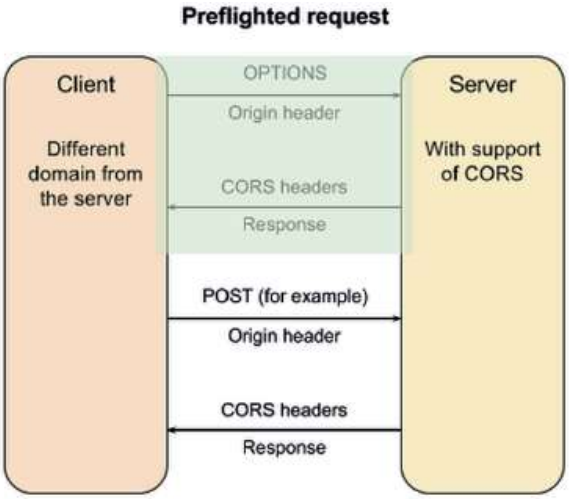
- `switchOn()`: realiza un POST a nuestra API con el contenido {“estado”:true}.
- `switchOff()`: realiza un POST a nuestra API con el contenido {“estado”:false}.
- `getReading()`: lanza un GET a nuestra API, e indica que la respuesta será gestionada por la función callback.
- `Callback()`: muestra un mensaje con el dato recibido.

5. Abre ahora tu fichero HTML desde Chrome. Comprobarás que al pulsar los botones no tiene ningún efecto sobre la Raspberry. Abre el modo desarrollador (F12) en vista de consola o de red:



Al utilizar los servicios RESTful desde una aplicación web, nos encontramos sujetos a la política de CORS, que gestiona el acceso de servicios web entre dis-

tintos dominios. Nuestro navegador está enviando una consulta «preflight», es decir, antes de lanzar el GET a nuestro servidor, lanza un OPTIONS para comprobar permisos de acceso por dominio, autenticación, métodos HTTP disponibles, etc.

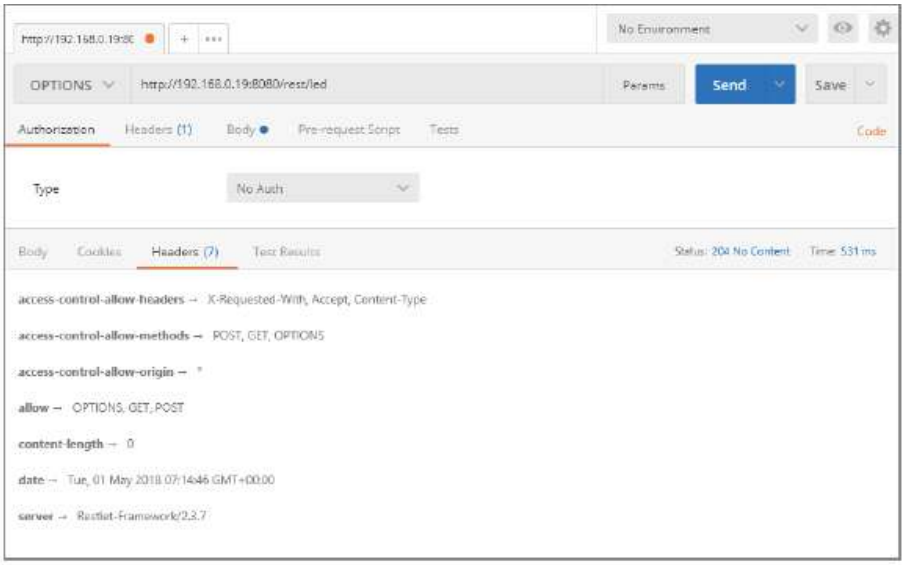


6. Abre la clase `LEDResource`, donde teníamos la gestión de los métodos GET y POST, y añade un nuevo apartado para el método OPTIONS:

```
public class LEDResource extends ServerResource {
  @Options public void getCorsSupport() {
    Set<String> head = new HashSet<>();
    head.add("X-Requested-With");
    head.add("Content-Type");
    head.add("Accept");
    getResponse().setAccessControlAllowHeaders(head);

    Set<Method> methods = new HashSet<>();
    methods.add(Method.GET);
    methods.add(Method.POST);
    methods.add(Method.OPTIONS);

    getResponse().setAccessControlAllowMethods(methods);
    getResponse().setAccessControlAllowOrigin("*");
  }
    .
```

En este código estamos editanto la respuesta de nuestro servidor con `getResponse()`, y le estamos añadiendo diversas cabeceras necesarias para soportar CORS. Por ejemplo, indicamos que los métodos que permite la llamada a /REST/LED son GET, POST y OPTIONS, y que se permite el acceso desde cualquier dominio (Access-Control-Allow-Origin : *).

7. Haz una consulta de tipo OPTIONS con Postman, y comprueba cómo queda la estructura de cabeceras de la respuesta. Averigua qué indica cada una de estas opciones.



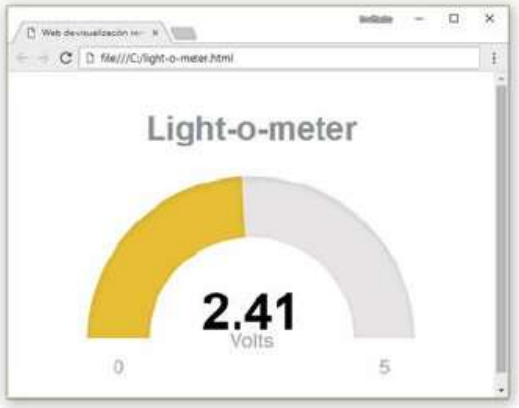
8. Vuelve a cargar la web "**interruptor.html**" y comprueba su funcionamiento. Comprueba que los 3 botones ya tienen funcionalidad, pero si activas el modo desarrollador (F12) y verificas la consola sigue apareciendo un error. ¿Qué nos indica este error? (ver respuesta[1]).
9. Abre la clase `LEDResource` y añade en los métodos GET y POST la siguiente línea, justo antes de la llamada al `return`:

```
    getResponse().setAccessControlAllowOrigin("*");
    return
}
```

10. Vuelve a comprobar el funcionamiento de la web "**interruptor.html**"; verás que ya no aparecen avisos en la consola.

[1] Este error indica que las respuestas recibidas a las consultas GET y POST no contienen la cabecera de "Access-control-allow-origin".

Práctica: API REST para lectura de iluminación



Añade la electrónica para la lectura del fotorresistor. Después, modifica el ejercicio anterior y añade un nuevo recurso REST para él. Deberás crear una clase `LDRmodel` y una clase `LDRResource`, y añadir una ruta **/rest/ldr** al router del webserver. Este recurso solo servirá peticiones OPTIONS y GET, y devolverá un valor float entre 0 y 5, correspondiente a la tensión detectada (que por el funcionamiento del fotorresistor será proporcional a la iluminación). La respuesta tendrá el siguiente formato:

```
{"result": 3.3}
```

Para la visualización, crea una web html con el siguiente código. Deberás editarlo para que apunte a la dirección y puerto de tu servidor:

```
<html>
<head>
<title>Web de visualización remota de iluminación</title>
<script
 src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"
 type="text/javascript" charset="utf-8"></script>
<script
 src="https://cdnjs.cloudflare.com/ajax/libs/raphael/2.1.0/raphael-min.js"
 type="text/javascript" charset="utf-8"></script>
<script src="http://cdn.jsdelivr.net/justgage/1.0.1/justgage.min.js"
 type="text/javascript" charset="utf-8"></script>
<script>
 var android_things_ip = "192.168.0.19";
 var android_things_port = "8080";
 var url = "http://" + android_things_ip + ":" + android_things_port +
    "/rest/ldr";
 function callback(data, status){
    if (status == "success") {
      volts = parseFloat(data.result);
      volts = volts.toFixed(2);
      g.refresh(volts);
      setTimeout(getReading, 1000);
```

```
    } else {
      alert("Hubo un problema");
    }
 }
 function getReading(){
    $.get(url, {}, callback);
 }
 </script>
</head>
<body>
  <div id="gauge" class="200x160px"></div>
  <script>
    var g = new JustGage({
      id: "gauge",
      value: 0,
      min: 0,
      max: 5,
      label: "Volts",
      title: "Light-o-meter"
    });
    getReading();
  </script>
</body>
</html>
```

Preguntas de repaso: Servidor API REST

5.3.2.1.2 Cliente REST

Disponer de un servidor web en nuestra RP3 resulta muy cómodo: no se necesita recurrir a un servidor adicional en Internet. Tanto en estos ejercicios como en la página web alojada en la RP3, no hemos necesitado nada más que la Raspberry y nuestro navegador. Sin embargo, en la práctica es problemático disponer de un servidor web en la RP3, por 3 razones principales:

- Consumo de recursos: un servidor web, incluso ligero, consume muchos recursos. Si se espera que muchos clientes externos puedan acceder a este equipo, la RP3 se encontrará con dificultades para servir todas las peticiones.
- Un dispositivo conectado a Internet con un servicio abierto es un problema de seguridad. Los rastreadores de Internet estarán continuamente accediendo a él para ver lo que ofrece, y cualquier vulnerabilidad puede ser utilizada por un atacante para dejar sin servicio el sistema, o incluso tomar su control.
- Los equipos IoT son equipos externos a Internet, alojados tras routers con NAT. Ofrecer un servicio al exterior requiere tener acceso a ese NAT y gestionar adecuadamente la apertura y redirección de puertos, lo que no siempre es posible o práctico.

Al final, la solución habitual en los proyectos IoT es utilizar un servidor en Internet. Para grandes proyectos con miles de nodos, se recurre a backends clouds, capaces de ingerir grandes cantidades de datos de dispositivos a lo largo de todo el mundo. En estos casos, el servidor del API web estará en este backend, y nuestro dispositivo IoT realizará la función contraria, la de cliente del API.

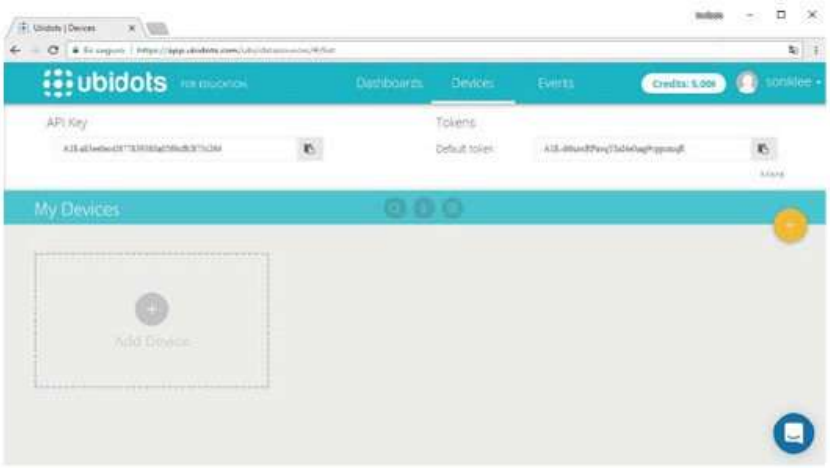
Ejercicio: Interacción con Ubidots mediante API REST

Ubidots es una plataforma muy potente e interesante para proyectos de Internet de las cosas, que nos permite integrar datos de sensores, almacenarlos, mostrarlos en un dashboard y generar distintos tipos de eventos. En este ejercicio aprenderemos a utilizarlo para nuestros proyectos IoT, y a implementar un cliente en Android Things para interactuar con su API REST. Fíjate en que en este caso ya no es necesario disponer de un servidor web para ofrecer servicios REST, sino que estos servicios estarán alojados en el backend IoT de Ubidots y, por tanto, deberás implementar un cliente HTTP para comunicarte con él.

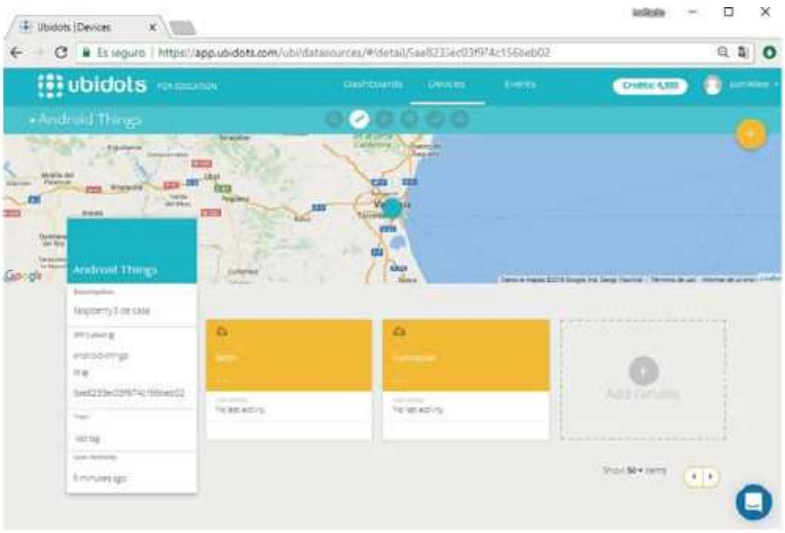
1. Entra en **Ubidots** y crea una cuenta de estudiante:

```
https://app.ubidots.com
```

2. Cuando estés dentro de tu cuenta, haz clic en tu nombre de usuario (arriba a la derecha) y luego en API credentials. Verás que aparece tu API key (no la necesitas, se utiliza para generar tokens válidos) y un **default token**. Este token no caduca, y permite autenticar a un cliente cuando utiliza el API de Ubidots para enviar datos a tu cuenta, así que será el que utilicemos. También es posible solicitar tokens al vuelo, mediante una consulta POST al API, que tienen una caducidad de 6h.



3. Dirígete a Devices y añade un nuevo dispositivo, de nombre «**Android Things**». Haz clic en él para ver su contenido. Edita el campo Description, y añade la ubicación del dispositivo.
4. Pulsa en Add variable > Default para añadir la sensorización que vamos a enviar desde nuestro dispositivo. Utilizaremos una variable **Iluminación** para el valor del fotorresistor, y una variable **Botón** para saber si el botón está pulsado.



5. Accede a la variable Iluminación y edita sus propiedades. Utilizaremos como unidades «volts», y como rango permitido «0-5». Si tratamos de enviar valores fuera de este rango, el API nos devolverá un error, lo que nos aporta un método de controlar la validez de las lecturas desde la propia API. Haz lo mismo con la variable Botón; sin unidades y con un rango «0-1». Fíjate en el campo **ID** de ambas variables; lo utilizaremos más adelante para el envío de los datos a través del API.
6. Crea un nuevo proyecto para Android Things.
7. Comenzamos con las dependencias. Utilizaremos **retrofit** para desarrollar el cliente HTTP (http://square.github.io/retrofit/), y **gson** para la serialización/deserialización de los JSON (https://github.com/google/gson). Añade las siguientes dependencias en build.gradle (Module: app):

```
dependencies {
    .
    implementation 'com.squareup.retrofit2:retrofit:2.3.0'
    implementation 'com.google.code.gson:gson:2.8.2'
    implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
}
```

8. Añade los permisos necesarios en el manifiesto (AndroidManifest.xml). Utilizaremos el permiso `INTERNET` y`USE_PERIPHERAL_IO`:

```
<manifest .>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="com.google.android.things.permission
                                                    .USE_PERIPHERAL_IO" />
    <application .>
```

9. Comenzaremos creando una clase `Data`, donde crearemos nuestro modelo de datos. Según la guía de referencia para el API de Ubidots, cuando queremos enviar los datos de varios sensores debemos utilizar el siguiente formato:

```
[{"variable": "{VAR_ID_1}", "value":41.2}, {"variable": "{VAR_ID_1}", "va-
lue":88.3}]
```

Vemos que tenemos un JSON con un array de parejas "variable" – "valor". Nuestra clase `Data` expondrá estos dos campos, el primero de tipo `String` y el segundo de tipo `Double`.

Crea la clase `Data` y añade el siguiente contenido:

```
public class Data {

  @SerializedName("variable")
  @Expose private String variable;

  @SerializedName("value")
  @Expose private Double value;

  public String getVariable() { return variable; }

  public void setVariable(String variable) { this.variable = variable; }

  public Double getValue() { return value; }

  public void setValue(Double value) { this.value = value; }
}
```

10. A continuación, crea una interfaz de nombre `UbiAPI`, con el siguiente contenido:

```
import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.POST;
import retrofit2.http.Query;

public interface UbiAPI {
  @POST("/api/v1.6/collections/values")
  public Call<ResponseBody> sendValue(
    @Body ArrayList<Data> dataList, @Query("token") String token);
}
```

Esta interfaz representa el API al que invocamos. La anotación `@POST` indica el método que se utilizará al llamar a la interfaz. Este método acepta como entrada un array de objetos `Data`, tal y como los hemos definido en el punto anterior; y un token de seguridad para conectar con el API. También se indica la dirección a la que apuntará el método POST, según lo especificado en la guía de referencia de Ubidots[2].

11. Ahora crearemos el cliente que manejará la comunicación con Ubidots. Crea una clase `UbiClient` y utiliza el siguiente contenido:

```
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class UbiClient {
  private static final String TAG = UbiClient.class.getSimpleName();
  private static final String UBI_BASE_URL = "http://things.ubidots.com/";
```

[2] https://ubidots.com/docs/api/#send-values-to-many-variables

```
  private static UbiClient cLient;
  private UbiAPI api;
  private Retrofit retroClient;

  private UbiClient() {
    retroClient = new Retrofit.Builder()
      .baseUrl(UBI_BASE_URL)
      .addConverterFactory(GsonConverterFactory.create())
      .build();
  }

  public static final UbiClient getClient() {
    if (cLient != null) return cLient;
    cLient = new UbiClient();
    return cLient;
  }

  private UbiAPI getUbiClient() {
    return retroClient.create(UbiAPI.class);
  }

  public void sendData(ArrayList<Data> dList, String token) {
    api = cLient.getUbiClient();
    Call c = api.sendValue(dList, token);
    c.enqueue(new Callback() {
      @Override public void onResponse(Call call, Response response) {
        Log.d(TAG, "onResponse");
        Log.d(TAG, "Result: " + response.isSuccessful() + " - " +
              response.message());
      }
      @Override public void onFailure(Call call, Throwable t) {
        t.printStackTrace();
      }
    });
  }
}
```

Aquí encontramos la URL base para la comunicación con Ubidots, la creación del cliente y el método para el envío de datos: `sendData()`. Este método dispone de un callback que nos indica el resultado de la operación, de forma que podríamos realizar operaciones adicionales según la respuesta del API.

12. Ahora ya podemos desarrollar el código principal de nuestra `MainActivity`. Utiliza el siguiente contenido:

```
public class MainActivity extends Activity {
  // IDs Ubidots
  private final String token = "A1E-s86uiwRPawqYInI4e0aag9vppcxoqR";
  private final String idIluminacion = "5aeec55ec03f97502473e6de";
  private final String idBoton = "5aeec565c03f97516e8480e0";

  private final String PIN_BUTTON = "BCM23";
  private Gpio mButtonGpio;
  private Double buttonstatus = 0.0;
  private Handler handler = new Handler();
```

```
private Runnable runnable = new UpdateRunner();

@Override protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  PeripheralManager service = PeripheralManager.getInstance();
  try {
    mButtonGpio = service.openGpio(PIN_BUTTON);
    mButtonGpio.setDirection(Gpio.DIRECTION_IN);
    mButtonGpio.setActiveType(Gpio.ACTIVE_LOW);
    mButtonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING);
    mButtonGpio.registerGpioCallback(mCallback);
  } catch (IOException e) {
    Log.e(TAG, "Error en PeripheralIO API", e);
  }
  handler.post(runnable);
}

@Override protected void onDestroy() {
  super.onDestroy();
  handler = null;
  runnable = null;
  if (mButtonGpio != null) {
    mButtonGpio.unregisterGpioCallback(mCallback);
    try {
      mButtonGpio.close();
    } catch (IOException e) {
      Log.e(TAG, "Error en PeripheralIO API", e);
    }
  }
}

// CaLLback para envio asincrono de puLsación de botón
private GpioCallback mCallback = new GpioCallback() {
  @Override public boolean onGpioEdge(Gpio gpio) {
    Log.i(TAG, "Botón pulsado!");
    if (buttonstatus == 0.0) buttonstatus = 1.0;
    else buttonstatus = 0.0;
    final Data boton = new Data();
    boton.setVariable(idBoton);
    boton.setValue(buttonstatus);
    ArrayList<Data> message = new ArrayList<Data>() {{add(boton);}};
    UbiClient.getCLient().sendData(message, token);
    return true; // Mantenemos eL caLLback activo
  }
};

// Envio sincrono (5 segundos) deL vaLor deL fotorresistor
private class UpdateRunner implements Runnable {
  @Override public void run() {
    readLDR();
    Log.i(TAG, "Ejecución de acción periódica");
    handler.postDelayed(this, 5000);
  }
```

```
  }
  private void readLDR() {
    Data iluminacion = new Data();
    ArrayList<Data> message = new ArrayList<Data>();
    Random rand = new Random();
    float valor = rand.nextFloat() * 5.0f;
    iluminacion.setVariable(idIluminacion);
    iluminacion.setValue((double) valor);
    message.add(iluminacion);
    UbiClient.getCLient().sendData(message, token);
  }
}
```

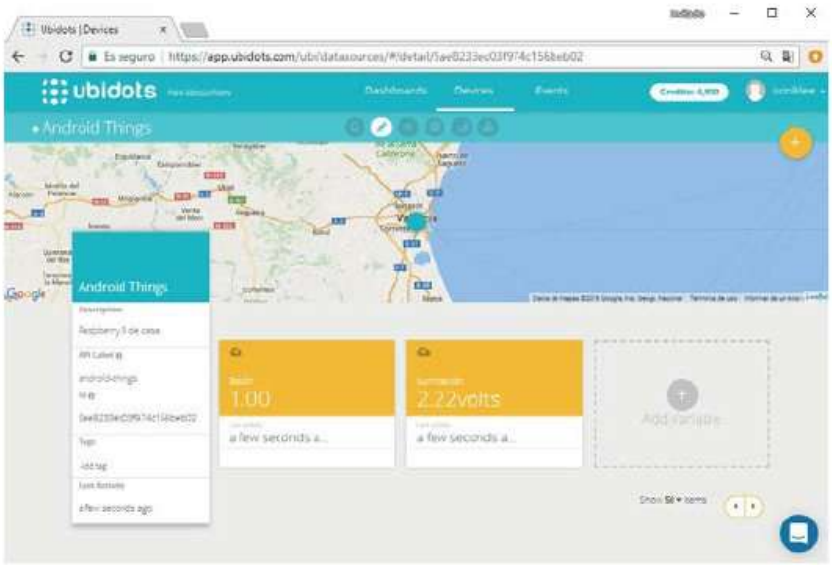
Demos un vistazo a lo que contiene este código. Comenzamos definiendo algunos identificadores claves para la comunicación con el API de Ubidots. Utiliza como `token` tu default token (ver paso 2), y como `idIluminación` e `idBoton` las IDs de las variables que has creado en tu cuenta de Ubidots.

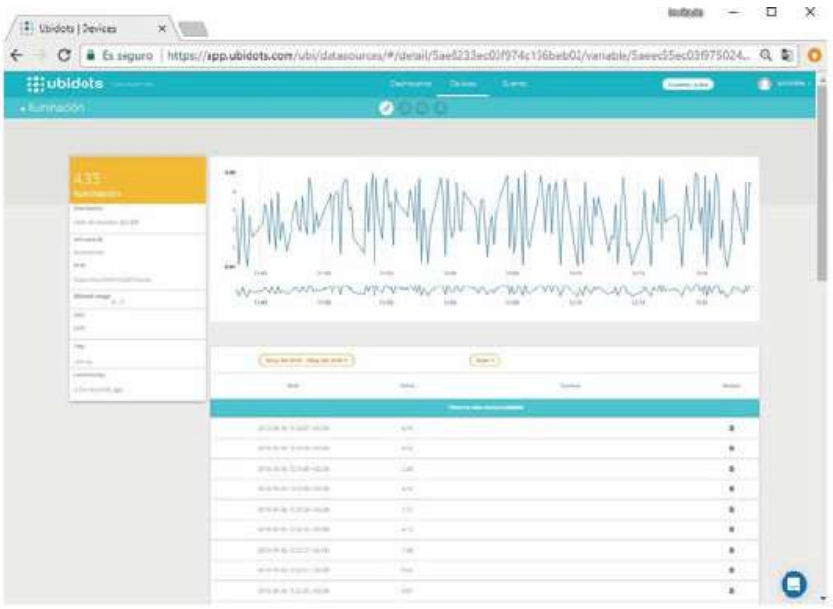
Para cada variable que queremos enviar creamos un objeto `Data`, al que le añadimos el ID de variable y su valor actual. Como el API de Ubidots acepta un array de objetos `Data` como entrada, lo añadimos a un ArrayList, y luego lo enviamos con `sendData()`.

El envío de datos a Ubidots se realiza de dos formas:

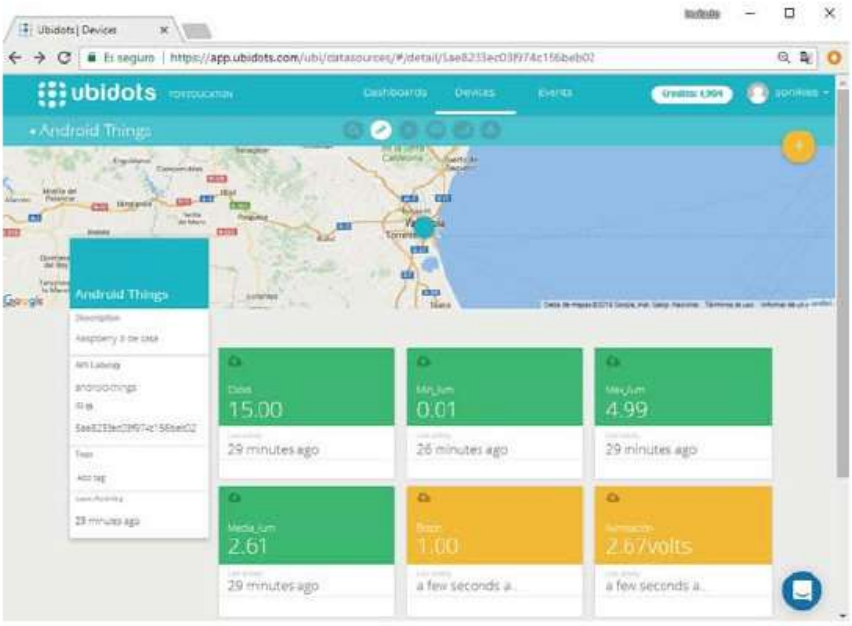
- Pulsador: de forma asíncrona. Cuando presionamos el pulsador, se invoca al callback mCallback, donde construimos el objeto `Data` y llamamos a `sendData()`.
- Fotorresistor: de forma síncrona. Mediante el uso de las clases `Handler` y `Runnable`, generamos un evento cada 5 segundos, donde se lee el valor de fotorresistor, se genera el objeto `Data`, y se envía con `sendData()`.

13. Carga y prueba el proyecto. La vista de variables en Ubidots debería comenzar a recibir valores. Presiona el pulsador de la placa de prototipado para ver cómo se actualiza su lectura. Si accedes a las variables, puedes ver el histórico en formato tabla y gráfico.

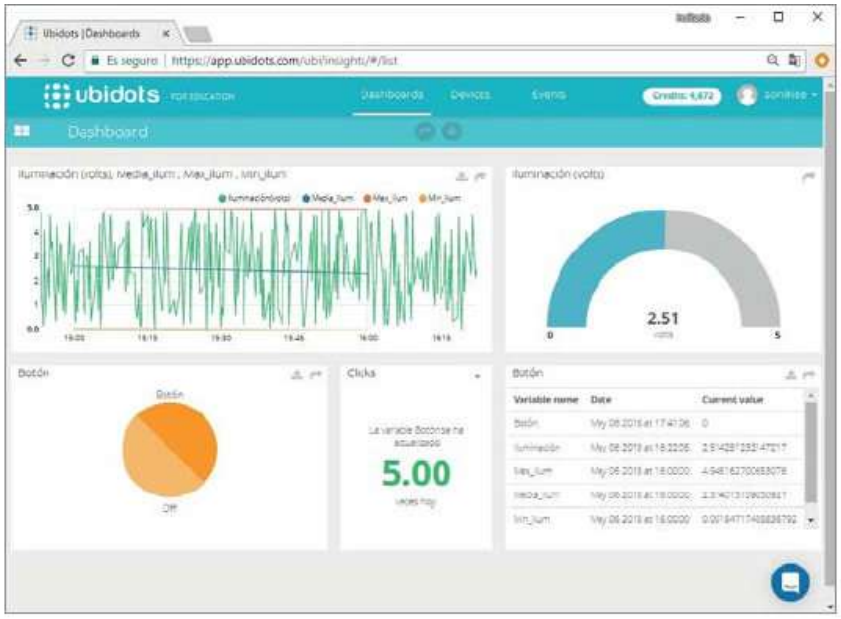




14. Una característica muy interesante de las plataformas IoT es la generación de variables virtuales, formadas a partir de operaciones sobre las variables físicas. Accede de nuevo al dispositivo Android Things en Ubidots y crea 4 nuevas variables de tipo Rolling window. Crea una con la media horaria del valor de iluminación, otra con el máximo horario, y otra con el mínimo horario. Después, crea una variable con la cuenta diaria de clics del botón. Estas variables se muestran en color verde para diferenciarlas de las variables físicas (en amarillo).



15. Actualmente, estamos utilizando Ubidots como un servicio de almacenamiento, donde se ingieren las medidas de nuestro dispositivo y se almacenan en forma de histórico. Ubidots nos ofrece, además, dos formas de explotar estos datos: **dashboards** (para visualización) y **events** (para acciones). Vamos a revisar la función de dashboards. Intenta generar el siguiente panel:



Nota: Los dashboards son elementos muy importantes en aplicaciones de supervisión, mando y control, puesto que permiten conocer de un vistazo el estado de nuestra flota de dispositivos, o de los procesos que monitorizan. Es importante realizar un diseño inteligente de este tipo de paneles, ya que no siempre más información es mejor. En general se prefiere menor información pero de mayor calidad: por ejemplo, en industria, este tipo de indicadores suelen referirse como "key performance indicators" o KPIs.

Preguntas de repaso: Plataformas IoT en la nube

5.3.2.2. Modelo publish/subscribe: MQTT

MQTT (Message Queuing Telemetry Transport) es un protocolo de aplicación que utiliza un modelo de comunicación publish/subscribe, estandarizado como ISO/IEC PRF 20922. Existen 3 roles: publicador, suscriptor y bróker. Los publicadores son fuentes de información, que envían al bróker etiquetada con un «topic». Los suscriptores son consumidores de información. Indican al bróker su interés en uno o varios «topics». El bróker es el mediador entre ambas entidades: tiene un registro de los suscriptores y de aquellos topics que les interesan, y al recibir el mensaje de un publicador, lo reenvía a todos los suscriptores que están interesados en ese topic. Un mismo elemento final puede actuar tanto como publicador de los datos de sus sensores, como como suscriptor a los datos de otros elementos.

MQTT es un protocolo agnóstico al contenido, lo que significa que provee al desarrollador de un canal de comunicaciones entre un publicador y varios suscriptores, y en ese canal puede enviar todo tipo de información (datos binarios, texto, XML, JSON ... etc.). No ofrece un mecanismo propio de seguridad; esta debe garantizarse en otros niveles de la comunicación. Por otro lado, es un sistema sin memoria, por lo que un elemento que se acabe de conectar no podrá recibir men-

sajes pasados, solo el último mensaje recibido por el bróker y los futuros mensajes.

MQTT presenta las siguientes operaciones:

- `CONNECT`: establecimiento de conexión con el bróker, indicando su IP y puerto. Esto requiere el establecimiento de un socket TCP, y su mantenimiento mediante keepalives.
- `PUBLISH`: mensaje enviado al bróker, que contiene un topic y una carga útil. El topic sirve para clasificar los mensajes, filtrarlos, facilitar la suscripción y, en general, poder diferenciarlos por intereses. La carga útil puede ser de cualquier tipo, son las aplicaciones de ambos extremos (publicador y suscriptor) las responsables de entender su contenido.
- `SUBSCRIBE`: mensaje enviado al bróker, indicando a qué topic deseamos suscribirnos para recibir sus mensajes.
- `DESUBSCRIBE`: mensaje enviado al bróker para eliminar nuestra suscripción de un topic y dejar de recibir sus mensajes.
- `DISCONNECT`: cierre de la conexión con el bróker y liberación de los recursos de la sesión TCP/IP.

También ofrece un control de entrega de mensajes, al que se refiere como «nivel de QoS». Con esto, podemos indicar si queremos que la conexión bróker-cliente asegure la entrega de mensajes o no. Esto permite realizar un diseño adaptado a las necesidades: mayor robustez para luchar contra nodos en redes con mayor tasa de pérdidas, menor robustez para mensajes menos prioritarios, redes más estables o con mayor cantidad de nodos, etc.

MQTT se considera el protocolo estrella de Internet de las cosas por diversas razones:

- Es un estándar abierto que todo el mundo puede implementar, y da libertad a utilizar el contenido (payload) que se requiera para cada aplicación.
- Sus requisitos de recursos computacionales son muy bajos, por lo que la huella necesaria en los dispositivos finales es mínima.
- Su implementación y funcionamiento es muy sencillo.
- El bróker no modifica los mensajes, solamente los redirige, por lo que es capaz de manejar tráfico de grandes cantidades de dispositivos con un impacto bajo.
- Presenta una latencia muy baja: tal y como se genera un mensaje, lo reciben los suscriptores.
- Los niveles de QoS permiten adaptarse a redes poco robustas y propensas a la pérdida de datos, como las inalámbricas.

Ejercicio: Conexión y publicación con MQTT

En este ejercicio aprenderás cómo realizar la configuración y conexión con un bróker MQTT. Para ello utilizaremos un código muy sencillo que conectará y publicará un «Hello World!» en un topic, y que visualizaremos desde un cliente MQTT de escritorio.

Para este ejercicio no es necesario realizar ningún montaje hardware.

1. Crea un nuevo proyecto para Android Things.
2. Comenzamos añadiendo las dependencias. Para implementar MQTT podemos encontrar multitud de librerías Java, pero de entre todas ellas utilizaremos **Eclipse Paho**, que nos ofrece las siguientes ventajas: es un proyecto Eclipse estable, bien documentado y mantenido por una extensa comunidad, y dispone de soporte Android nativo (https://eclipse.org/paho/clients/android/).

   En primer lugar, debemos añadir la ruta al repositorio en el build.gradle (Project):

```
repositories {
    google()
    jcenter()
    mavenCentral()
}
```

   A continuación, añade las siguientes dependencias en el build.gradle (Module:app):

```
dependencies {
    .
    implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.1'
}
```

3. Añade los permisos necesarios en el manifiesto (AndroidManifest.xml). Utilizaremos el permiso `INTERNET`:

```
<manifest .>
    <uses-permission android:name="android.permission.INTERNET" />
    <application .>
```

4. Este primer ejercicio utilizará una estructura muy sencilla, directamente en la `MainActivity`, para mostrar los pasos básicos de una publicación en MQTT. Utiliza el siguiente código:

```
public class MainActivity extends Activity {
  private static final String TAG = "Things";
  private static final String topic = "<user>/test";
  private static final String heLLo = "Hello world!";
  private static final int qos = 1;
  private static final String broker = "tcp://iot.eclipse.org:1883";
  private static final String cLientId = "Test134568789";

  @Override protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    try {
      MqttClient client = new MqttClient(broker, clientId,
        new MemoryPersistence());
      Log.i(TAG, "Conectando al broker " + broker);
      client.connect();
      Log.i(TAG, "Conectado");
      Log.i(TAG, "Publicando mensaje: " + helLo);
      MqttMessage message = new MqttMessage(helLo.getBytes());
      message.setQos(qos);
      client.publish(topic, message);
      Log.i(TAG, "Mensaje publicado");
      client.disconnect();
      Log.i(TAG, "Desconectado");
    } catch (MqttException e) {
      Log.e(TAG, "Error en MQTT.", e);
    }
  }
}
```

En este código comenzamos declarando diversas variables para el funcionamiento de MQTT:

- `topic`: cadena completa del topic donde publicaremos el mensaje. Pon cualquier nombre o alias en el campo `<user>`, para que tus mensajes no se crucen con los de tus compañeros.
- `hello`: mensaje "Hello world!" que enviamos al conectar con el bróker.
- `qos`: nivel de calidad de servicio que queremos para nuestras comunicaciones MQTT con el bróker. El nivel 1 es un compromiso entre garantía de entrega y bajo intercambio de mensajes de control.
- `broker`: dirección del servidor y puerto de escucha donde se encuentra el bróker MQTT al que conectamos. Utilizamos un bróker público de Eclipse, disponible para los usuarios para el testeo de aplicaciones, y que no requiere de autenticación. En caso de que no estuviera disponible, el bróker Mosquitto[3] también dispone de un bróker abierto de pruebas. En esta dirección[4] se mantiene una lista actualizada de brókers abiertos.
- `clientId`: identificador único de cliente en el bróker al que conectamos. Modifícalo aleatoriamente para que no coincida con ningún otro usuario, por ejemplo cambiando los números del final.

El resto de código se encuentra en el método `onCreate()`. En primer lugar, se crea un cliente MQTT de tipo `MqttClient`, indicando la dirección del bróker, el clientId, y un buffer de memoria para tratar los mensajes. Con la llamada al método `connect()` solicitamos una conexión con este bróker.

[3] https://test.mosquitto.org

[4] https://github.com/mqtt/mqtt.github.io/wiki/public_brokers

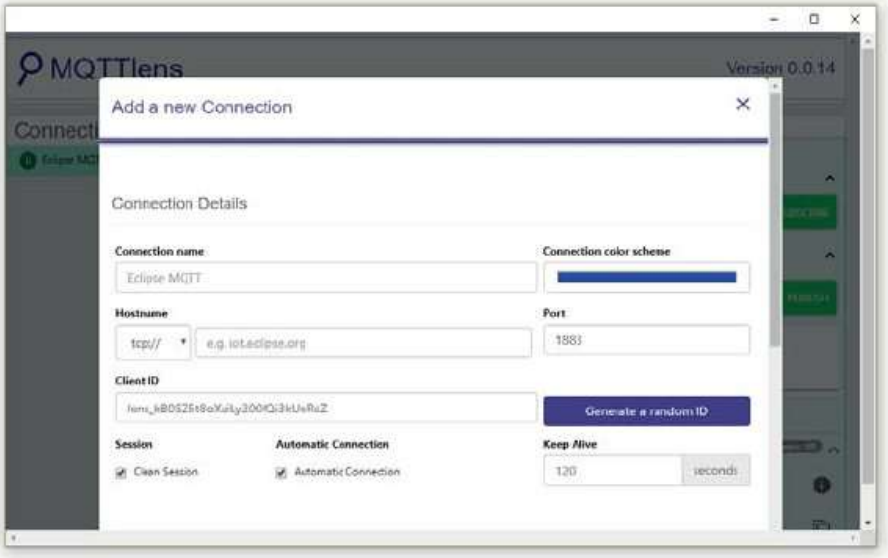
A continuación, creamos un mensaje MQTT con un objeto `MqttMessage`. Este objeto, además del mensaje, contiene otros campos modificables como la QoS del mensaje.

Para enviar el mensaje, utilizamos el método `publish()`. Si hubiera algún problema, como por ejemplo que no hayamos podido conectar con el bróker, nos saltaría una excepción que recogería el `catch()`. Si todo es correcto, cerramos nuestra conexión con el bróker, liberando los recursos de la sesión.

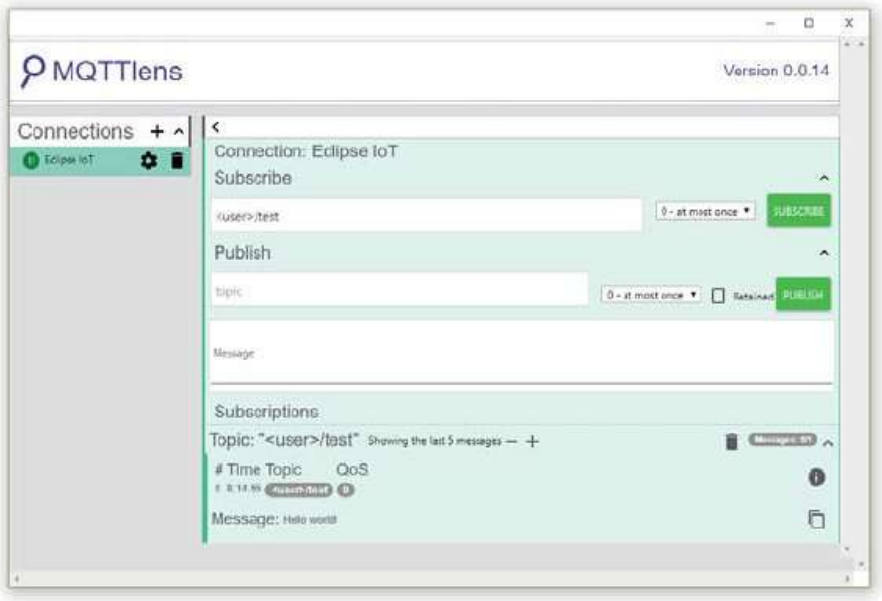
5. Para comprobar el mensaje que la Raspberry envía al bróker MQTT, necesitamos conectarnos a este bróker con un cliente y suscribirnos al tópic donde va a publicar. Podemos encontrar multitud de clientes para móvil (MyMQTT para Android y MQTTInspector para IOS), escritorio (MQTTLens como extensión de Chrome, MQTT.fx para Win/MacOSX/Linux, y MQTT-spy en Java multiplataforma) o incluso online (HiveMQ y Mitsuruog). A la hora de configurar estos clientes, hay que tener en cuenta si la conexión al bróker debe hacerse de forma directa (MQTT a puerto 1883) o con websockets (generalmente, los clientes online).

   En este ejercicio utilizaremos como ejemplo MQTTLens; instálalo y lánzalo.

6. Añade una nueva conexión con los siguientes datos (utiliza como Client ID cualquier valor aleatorio):



7. Una vez añadido, comprueba que estás correctamente conectado al bróker, lo que se indica con un icono verde al lado de la conexión.
8. Suscríbete al tópic `<user>/test`, utilizando como `<user>` el nombre/alias que hayas utilizado.
9. Carga y lanza el proyecto de Android Things. Comprueba que recibes el mensaje “Hello World!” en MQTTLens. Si tienes algún problema, comprueba que los tópics en MQTTLens y en el proyecto de Android Things coinciden.



Fíjate en la información que nos indica MQTTLens del mensaje recibido. Tenemos el tópic, importante cuando nos suscribimos con un comodín, y podemos recibir mensajes de distintos tópics, la marca temporal de recepción de mensaje, la QoS y el contenido del mensaje. Si pinchamos en el icono de información de la derecha, nos mostrará también si existen mensajes duplicados.

Práctica: Publicación bajo demanda

Modifica el ejercicio anterior para realizar una aplicación que publique un mensaje MQTT cada vez que se presione un pulsador conectado a la Raspberry Pi. En resumen, esta nueva aplicación debe:

- Publicar los mensajes en el topic `<user>/boton`, donde `<user>` es tu nombre o alias.
- Al conectarse al bróker, publicar el siguiente mensaje de bienvenida: «Hello world! Android Things conectada».
- Hacer el montaje hardware de un pulsador. Al presionarlo, publicar el siguiente mensaje: «click!».
- Realizar la desconexión del bróker en el método `onDestroy()`.

Ejercicio: Suscripción para control remoto del LED

Ahora que ya hemos comprobado lo sencillo que es publicar mensajes en MQTT, vamos a centrarnos en el proceso de suscripción. Para ello vamos a desarrollar un ejercicio en el que, publicando desde MQTTLens un mensaje de «ON» o «OFF»,

actuaremos sobre el LED de la placa. Además, ampliaremos un poco nuestro código base para que el programa de conexión a MQTT sea más completo.

1. Crea un nuevo proyecto para Android Things.

   Añade las dependencias para Eclipse Paho. Primero, la ruta al repositorio en el build.gradle (Project):

```
repositories {
    google()
    jcenter()
    mavenCentral()
}
```

   A continuación, añade las siguientes dependencias en el build.gradle (Module:app):

```
dependencies {
    .
    implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.1'
}
```

2. Añade los permisos necesarios en el manifiesto (AndroidManifest.xml). Utilizaremos los permisos `INTERNET` y `USE_PERIPHERAL_IO`:

```
<manifest .>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="com.google.android.things.permission
                                .USE_PERIPHERAL_IO" />
    <application .>
```

3. En la `MainActivity`, utiliza el siguiente código:

```
public class MainActivity extends Activity implements MqttCallback {
  private static final String TAG = "Things";
  private final String PIN_LED = "BCM18";
  public Gpio mLedGpio;
  private static final String topic_gestion = "<user>/gestion";
  private static final String topic_Led = "<user>/led";
  static final String heLLo = "Hello world! Android Things conectada.";
  private static final int qos = 1;
  private static final String broker = "tcp://iot.eclipse.org:1883";
  MqttClient client;

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    PeripheralManager service = PeripheralManager.getInstance();
    try {
      mLedGpio = service.openGpio(PIN_LED);
      mLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
    } catch (IOException e) {
      Log.e(TAG, "Error en el API PeripheralIO", e);
    }
    try {
      String clientId = MqttClient.generateClientId();
      client = new MqttClient(broker, cLientId, new MemoryPersistence());
      client.setCallback(this);
      MqttConnectOptions connOpts = new MqttConnectOptions();
      connOpts.setCleanSession(true);
```

```
      connOpts.setKeepAliveInterval(60);
      connOpts.setWill(topic_gestion, "Android Things desconectada!"
            .getBytes(), qos, false);
      Log.i(TAG, "Conectando al broker " + broker);
      client.connect(connOpts);
      Log.i(TAG, "Conectado");
      Log.i(TAG, "Publicando mensaje: " + heLLo);
      MqttMessage message = new MqttMessage(heLLo.getBytes());
      message.setQos(qos);
      client.publish(topic_gestion, message);
      Log.i(TAG, "Mensaje publicado");
      client.subscribe(topic_Led, qos);
      Log.i(TAG, "Suscrito a " + topic_Led);
    } catch (MqttException e) {
      Log.e(TAG, "Error en MQTT.", e);
    }
  }

  @Override protected void onDestroy() {
    super.onDestroy();
    try {
      if (client != null && client.isConnected()) {
        client.disconnect();
      }
    } catch (MqttException e) {
      Log.e(TAG, "Error en MQTT.", e);
    }
    if (mLedGpio != null) {
      try {
        mLedGpio.close();
      } catch (IOException e) {
        Log.e(TAG, "Error en el API PeripheralIO", e);
      } finally {
        mLedGpio = null;
      }
    }
  }

  @Override public void connectionLost(Throwable cause) {
    Log.d(TAG, "Conexión perdida...");
  }

  @Override public void messageArrived(String topic, MqttMessage message)
       throws Exception {
    String payload = new String(message.getPayload());
    Log.d(TAG, payload);
    switch (payload) {
      case "ON":
        mLedGpio.setValue(true);
        Log.d(TAG, "LED ON!");
        break;
      case "OFF":
        mLedGpio.setValue(false);
```

```
        Log.d(TAG, "LED OFF!");
        break;
      default:
        Log.d(TAG, "Comando no soportado");
        break;
    }
  }

  @Override public void deliveryComplete(IMqttDeliveryToken token) {
    Log.d(TAG, "Entrega completa!");
  }
}
```

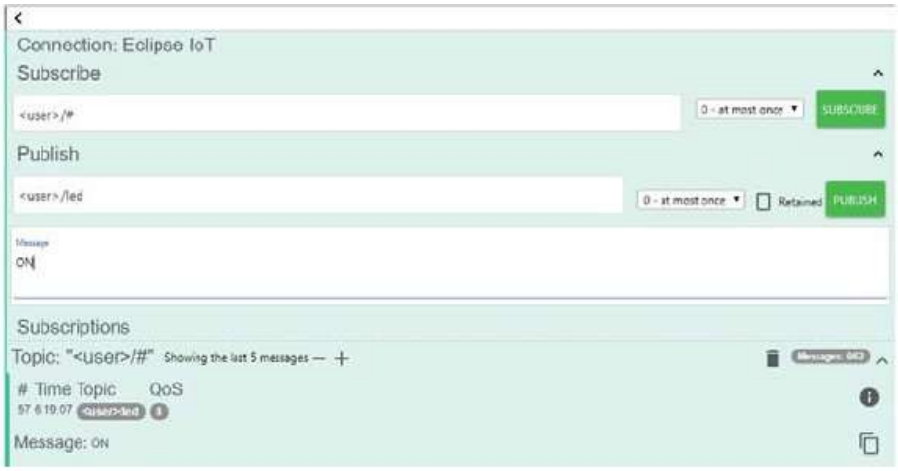
En resumen, en este código nos encontramos con lo siguiente:

- Esta vez se indican 2 topics, uno para mensajes de gestión y otro para los comandos del LED.
- Ya no especificamos un clientID. Para asegurarnos de que sea aleatorio y distinto cada vez, utilizamos el método `generateClientId()`.
- Utilizamos por primera vez el objeto `MqttConnectOptions` para establecer opciones de nuestra conexión MQTT con el bróker. Indicaremos que queremos iniciar siempre una sesión limpia, que se envíe un keepalive cada minuto, y estableceremos un mensaje de testamento. Estas opciones se pasan al realizar la llamada a `connect`.
- Nos suscribimos al topic de control de LED con el método `client.subscribe()`, indicando además el nivel de QoS que queremos mantener con el bróker para los mensajes recibidos.
- Se indica que nuestra clase implementa `MqttCallback`, para poder gestionar adecuadamente la recepción de mensajes MQTT a los topics a los que estemos suscritos.
- Con `client.setCallback(this)` indicamos que queremos establecer esta clase para la recepción de los callbacks.
- Al implementar la clase `MqttCallback`, añadimos tres métodos: `connectionLost()`, `messageArrived()` y `deliveryComplete()`. El primero se invoca cuando se pierde la conexión con el bróker, y podemos utilizarlo para lanzar alguna operación local como encender un LED de aviso, y para tratar de reconectar. Como hemos indicado que la sesión se inicia de forma limpia, deberemos volver a indicar a qué queremos suscribirnos. El segundo se invoca cuando la publicación de un mensaje se completa con éxito, lo que está ligado al nivel de QoS solicitado. Si hay QoS, este método no se invocará hasta que no se reciban los mensajes del bróker de confirmación de recepción. El tercer método es el que gestionará la recepción de un mensaje MQTT al que estamos suscritos.

4. Abre MQTTLens, conéctate al mismo bróker y suscríbete al siguiente tópic (donde `<user>` es tu nombre o alias):

```
<user>/#
```

5. Carga y prueba el proyecto. Cuando la Raspberry haya conectado con el bróker, deberás visualizar el mensaje de bienvenida. Fíjate en que, al haberte suscrito al topic `<user>/#`, te llegarán todos los mensajes de los topics que cuelguen de esta raíz, lo que incluye los dos topics utilizados en el proyecto: `<user>/gestion` y `<user>/led`.
6. Prueba a activar y desactivar el LED. Para eso envía un mensaje con el texto «ON» o «OFF» al topic `<user>/led`.



7. Cuando termines las pruebas, pulsa STOP en Android Studio para detener la ejecución del programa en la Raspberry. Comprueba que se recibe el mensaje de testamento.

Ejercicio: Cliente web de MQTT mediante websockets

El protocolo MQTT soporta tanto la conexión por un socket a un puerto estándar (1883) como la conexión mediante websockets. Los websockets son una herramienta fundamental para las aplicaciones web, pues permiten establecer sockets mediante los puertos 80 y 8080, de forma que pueden intercambiar datos a través de Internet sin sufrir los problemas asociados a cortafuegos y configuraciones NAT. Los brokers más habituales soportan la conexión de clientes tanto mediante puerto estándar como mediante websockets, tanto en sus versiones seguras como en las no seguras.

En este ejercicio aprenderás a generar un cliente MQTT web mediante websockets, en el que realizaremos la conexión, suscripción, visualización de mensajes entrantes, publicación de mensajes, y desconexión de bróker. Para ello utilizaremos la librería Paho de Eclipse para Javascript[5]. Esto te permitirá diseñar todo tipo de aplicaciones para interactuar con nodos MQTT, tanto para visualización como de control.

1. Abre el Notepad++, o cualquier otro editor de texto, y genera un nuevo fichero "mqtt_webclient.html".

[5] https://www.eclipse.org/paho/clients/js/

2. Crea la siguiente estructura básica:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
     <title>Cliente Websocket de MQTT</title>
   </head>
   <body align="center">
   </body>
</html>
```

3. Comenzamos con el contenido de la cabecera. Inserta los siguientes scripts entre las etiquetas <head> _ </head>:

```
<head>
 <script
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"
  type="text/javascript"></script>
 <script
  src="https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.1/mqttws31.js"
  type="text/javascript"></script>
 <script type="text/javascript">
   var client = new Paho.MQTT.Client("iot.eclipse.org", Number(80), "/ws",
        "myclientid_" + parseInt(Math.random() * 100, 10));
    //Se invoca si la conexión MQTT se pierde
    client.onConnectionLost = function (responseObject) {
      alert("connection lost: " + responseObject.errorMessage);
    };
    //Se invoca cuando se recibe un mensaje de nuestras suscripciones
    client.onMessageArrived = function (message) {
      $('#messages').append('<span><small><b>Topic:</b> ' +
      message.destinationName + ' <b>I Mensaje:</b> ' +
      message.payloadString + '<small></span><br/>');
    };
    //Opciones de conexión
    var options = {
      timeout: 3,
      //Se invoca si la conexión ha sido satisfactoria
      onSuccess: function () {
        alert("Connected");
      },
      //Se invoca si no se ha podido establecer la conexión
      onFailure: function (message) {
        alert("Connection failed: " + message.errorMessage);
      }
    };
    //Publicación de mensajes
    var publish = function (payload, topic, qos) {
      var message = new Paho.MQTT.Message(payload);
      message.destinationName = topic;
      message.qos = qos;
```

```
      client.send(message);
    }
  </script>
</head>
```

En este código hay dos partes principales: la carga de las librerías de jquery y paho-mqtt, y el código del script en sí mismo.

En el código del script, comenzamos generando un cliente MQTT. Según la documentación del bróker de Eclipse, la conexión por websockets se realiza a la dirección:

```
iot.eclipse.org:80/ws
```

Esta dirección está formada por 3 partes: el host, el puerto, y el path. Para generar el cliente, el método utiliza la siguiente estructura, donde introduciremos un clientId aleatorio:

```
new Client(host, port, path, clientId)
```

A continuación, tenemos un par de callbacks: `onConnectionLost` para realizar acciones si la conexión MQTT se pierde, y `onMessageArrived` para gestionar los mensajes recibidos sobre los topics a los que estamos suscritos. En ese caso, cuando recibimos un mensaje, lo mostramos en pantalla.

También se incluye un bloque para la gestión de la conexión, y otro para el método de publicación.

4. Vamos ahora con el contenido del cuerpo. Agrega el siguiente código entre las etiquetas `<body> . </body>`:

```
<body align="center">
 <h1>Cliente Websocket de MQTT<br></h1>
 <button onclick="client.connect(options);">1. Conectar</button>
 <button
   onclick="client.subscribe('<user>/#', {qos: 1}); alert('Subscribed');">
   2.   Suscribir</button>
 <button onclick="publish('Hello world! Soy el cliente websocket',
   '<user>/gestion',1);"> 3. Publicar</button>
 <button onclick="client.disconnect();">4. Desconectar</button><br><br>
 <div id="messages"></div>
</body>
```

Este código genera 4 botones en la página web: conectarse al bróker, suscribirse al topic `<user>/#` (modifica la etiqueta `<user>` por tu nombre o alias), publicar un mensaje de Hello en el topic `<user>/gestion` (modifica también esta etiqueta), y desconectar del bróker.

5. Guarda el fichero y ábrelo desde un navegador web.
6. Desde MQTTLens, suscríbete a `<user>/#` y publica mensajes en `<user>/test` Deberías visualizarlos tanto en MQTTLens como en el cliente web.
7. Pulsa en Publicar. Deberías visualizar los mensajes tanto en el cliente web como en MQTTLens. Esto se debe a que ambos clientes están suscritos a los mismos topics, así que estén donde estén recibirán estos mensajes.

Práctica: Control de Raspberry mediante cliente web MQTT

Haz el montaje de la Raspberry Pi con un LED externo y un pulsador. A partir de los dos últimos ejercicios, desarrolla un proyecto en el que podamos controlar desde una web, mediante el cliente MQTT websockets, el encendido y apagado del LED de la Raspberry. Utiliza también la interfaz web para indicar la acción sobre el pulsador.

Ejercicio: Shake it! MQTT en el móvil

En este ejercicio aprenderás a utilizar MQTT también en el móvil. Tras ello, ya podremos desarrollar clientes MQTT para móvil, Android Things y web, de forma que podremos comunicar aplicaciones entre distintas plataformas con un mínimo coste computacional y una elevada escalabilidad, lo cual es ideal en proyectos de IoT.

Para este ejercicio vamos a desarrollar una aplicación móvil con botones de encendido y apagado del LED y, además, al sacudir el teléfono, el LED parpadeará 4 veces.

1. Crea un módulo para Android Things y otro para móvil/tablet.

   Añade las dependencias para Eclipse Paho. Primero, la ruta al repositorio en el build.gradle (Project):

```
repositories {
    google()
    jcenter()
    mavenCentral()
}
```

   A continuación, añade las siguientes dependencias en el build.gradle de ambos módulos (Module:app y Module:mobile):

```
dependencies {
    .
    implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.1'
}
```

2. Añade los permisos necesarios en el manifiesto de módulo para Android Things (AndroidManifest.xml). Utilizaremos los permisos `INTERNET` y `USE_PERIPHERAL_IO`:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.things.permission
                                                .USE PERIPHERAL IO" />
```

3. Ahora añade los permisos para el módulo del móvil:

```
<uses-permission android:name="android.permission.INTERNET" />
```

4. Comenzamos con la aplicación móvil. Utiliza el siguiente layout en activity main.xml:

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/buttonConnect"
        style="@style/Widget.AppCompat.Button.Colored"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Conectar"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.198"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.177" />
    <Button
        android:id="@+id/buttonDisconnect"
        style="@style/Widget.AppCompat.Button.Colored"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Desconectar"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.851"
        app:layout_constraintLeft_toLeftOf="parent"
```

```
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.177" />
    <Button
        android:id="@+id/buttonON"
        style="@style/Widget.AppCompat.Button.Colored"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ON"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.222"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.777" />
    <Button
        android:id="@+id/buttonOFF"
        style="@style/Widget.AppCompat.Button.Colored"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OFF"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.831"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.777" />
</android.support.constraint.ConstraintLayout>
```

5. Ahora crea una nueva clase `ShakeListener`, que implemente `SensorListener`. Esta clase será la responsable de detectar la sacudida del teléfono móvil e invocar al listener onShake:

```
public class ShakeListener implements SensorListener {
  private static final int FORCE_THRESHOLD = 350;
  private static final int TIME_THRESHOLD = 100;
  private static final int SHAKE_TIMEOUT = 500;
  private static final int SHAKE_DURATION = 1000;
  private static final int SHAKE_COUNT = 3;
  private SensorManager mSensorMgr;
  private float mLastX = -1.0f, mLastY = -1.0f, mLastz = -1.0f;
  private long mLastTime;
  private OnShakeListener mShakeListener;
  private Context mContext;
  private int mShakeCount = 0;
  private long mLastShake;
  private long mLastForce;

  public interface OnShakeListener {
    public void onShake();
  }

  public ShakeListener(Context context) {
    mContext = context;
```

```
    resume();
  }

  public void setOnShakeListener(OnShakeListener listener) {
    mShakeListener = listener;
  }

  public void resume() {
    mSensorMgr = (SensorManager) mContext.getSystemService(Context
        .SENSOR_SERVICE);
    if (mSensorMgr == null) {
      throw new UnsupportedOperationException("Sensores no soportados");
    }
    boolean supported = mSensorMgr.registerListener(this,
      SensorManager.SENSOR_ACCELEROMETER,
      SensorManager.SENSOR_DELAY_GAME);
    if (!supported) {
      mSensorMgr.unregisterListener(this, SensorManager
          .SENSOR_ACCELEROMETER);
      throw new UnsupportedOperationException("Aceler. no soportado");
    }
  }

  public void pause() {
    if (mSensorMgr != null) {
      mSensorMgr.unregisterListener(this, SensorManager
          .SENSOR_ACCELEROMETER);
      mSensorMgr = null;
    }
  }

  public void onAccuracyChanged(int sensor, int accuracy) { }

  public void onSensorChanged(int sensor, float[] values) {
    if (sensor != SensorManager.SENSOR_ACCELEROMETER) return;
    long now =  System.currentTimeMiLLis();
    if ((now - mLastForce) > SHAKE_TIMEOUT) {
      mShakeCount = 0;
    }
    if ((now - mLastTime) > TIME_THRESHOLD) {
      long diff = now - mLastTime;
      float speed = Math.abs(values[SensorManager.DATA_X] +
          values[SensorManager.DATA_Y] + values[SensorManager.DATA_Z] -
          mLastX - mLastY - mLastz) / diff * 10000;
      if (speed > FORCE_THRESHOLD) {
        if ((++mShakeCount >= SHAKE_COUNT) && (now - mLastShake >
            SHAKE_DURATION)) {
          mLastShake = now;
          mShakeCount = 0;
          if (mShakeListener != null) {
            mShakeListener.onShake();
          }
        }
```

```
        mLastForce = now;
      }
      mLastTime = now;
      mLastX = values[SensorManager.DATA_X];
      mLastY = values[SensorManager.DATA_Y];
      mLastz = values[SensorManager.DATA_Z];
    }
  }
}
```

6. En la clase `MainActivity`, utiliza el siguiente código, modificando los campos `<user>` por tu nombre o alias:

```
public class MainActivity extends AppCompatActivity implements
     MqttCallback, ShakeListener.OnShakeListener {
  private static final String TAG = "Mobile";
  private static final String topic_gestion = "<user>/gestion";
  private static final String topic_Led = "<user>/led";
  static final String hello = "Hello world! Android Mobile conectado.";
  private static final int qos = 1;
  private static final String broker = "tcp://iot.eclipse.org:1883";
  MqttClient client;
  MqttConnectOptions connOpts;
  Button botonConnect, botonDisconnect, botonON, botonOFF;
  TextView textview;

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ShakeListener test = new ShakeListener(this);
    test.setOnShakeListener(this);
    textview = (TextView)  findViewById(R.id.textview);
    botonConnect = (Button) findViewById(R.id.buttonConnect);
    botonDisconnect = (Button) findViewById(R.id.buttonDisconnect);
    botonON = (Button) findViewById(R.id.buttonON);
    botonOFF = (Button) findViewById(R.id.buttonOFF);

    try {
      String clientId = MqttClient.generateCLientId();
      client = new MqttClient(broker, clientId, new MemoryPersistence());
      client.setCallback(this);
      connOpts = new MqttConnectOptions();
      connOpts.setCleanSession(true);
      connOpts.setKeepAliveInterval(60);
      connOpts.setWill(topic_gestion, ("Android Mobile " +
        "desconectado!").getBytes(), qos, false);
    } catch (MqttException e) {
      Log.e(TAG, "Error en MQTT.", e);
    }

    botonConnect.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        Log.i(TAG, "Boton presionado");
        textview.setText("Conectando...");
```

```
        try {
          Log.i(TAG, "Conectando al broker " + broker);
          client.connect(connOpts);
          Log.i(TAG, "Conectado");
          Log.i(TAG, "Publicando mensaje: " + hello);
          MqttMessage message = new MqttMessage(hello.getBytes());
          message.setQos(qos);
          client.publish(topic_gestion, message);
          Log.i(TAG, "Mensaje publicado");
          textview.setText("Conectado");
        } catch (MqttException e) {
          Log.e(TAG, "Error en MQTT.", e);
          textview.setText("Error al conectar");
        }
      }
    });
    botonDisconnect.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        Log.i(TAG, "Boton presionado");
        textview.setText("Desconectando...");
        try {
          if (client != null && client.isConnected()) {
            client.disconnect();
          }
          textview.setText("Desconectado");
        } catch (MqttException e) {
          Log.e(TAG, "Error en MQTT.", e);
          textview.setText("Error al desconectar");
        }
      }
    });
    botonON.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        Log.i(TAG, "Boton presionado");
        try {
          String mensaje = "ON";
          Log.i(TAG, "Publicando mensaje: " + mensaje);
          MqttMessage message = new MqttMessage(mensaje.getBytes());
          message.setQos(qos);
          client.publish(topic_Led, message);
          Log.i(TAG, "Mensaje publicado");
          textview.setText("Publicado ON");
        } catch (MqttException e) {
          Log.e(TAG, "Error en MQTT.", e);
          textview.setText("Error al publicar");
        }
      }
    });
    botonOFF.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        Log.i(TAG, "Boton presionado");
        try {
          String mensaje = "OFF";
```

```
            Log.i(TAG, "Publicando mensaje: " + mensaje);
            MqttMessage message = new MqttMessage(mensaje.getBytes());
            message.setQos(qos);
            client.publish(topic_Led, message);
            Log.i(TAG, "Mensaje publicado");
            textview.setText("Publicado OFF");
          } catch (MqttException e) {
            Log.e(TAG, "Error en MQTT.", e);
            textview.setText("Error al publicar");
          }
        }
      });
    }

    @Override protected void onDestroy() {
      super.onDestroy();
      try {
        if (client != null && client.isConnected()) {
          client.disconnect();
        }
      } catch (MqttException e) {
        Log.e(TAG, "Error en MQTT.", e);
      }
    }

    @Override public void connectionLost(Throwable cause) {
      Log.d(TAG, "Conexión perdida...");
    }

    @Override public void messageArrived(String topic, MqttMessage message)
      throws Exception { }

    @Override public void deliveryComplete(IMqttDeliveryToken token) {
      Log.d(TAG, "Entrega completa!");
    }

    @Override public void onShake() {
      Log.i(TAG, "Shake!");
      try {
        String mensaje = "Shake!";
        Log.i(TAG, "Publicando mensaje: " + mensaje);
        MqttMessage message = new MqttMessage(mensaje.getBytes());
        message.setQos(qos);
        client.publish(topic_Led, message);
        Log.i(TAG, "Mensaje publicado");
        textview.setText("Publicado Shake!");
      } catch (MqttException e) {
        Log.e(TAG, "Error en MQTT.", e);
        textview.setText("Error al publicar");
      }
    }
}
```

En el método `onCreate()` creamos el listener para el movimiento del teléfono, con `new ShakeListener(this)` y `setOnShakeListener(this)`. Cuando se sacuda el teléfono, se invocará el método `onShake()`, donde publicamos el mensaje "Shake!". El resto de código es muy similar a lo que ya hemos visto hasta ahora: tenemos 4 botones para conectar y desconectar del bróker y para encender y apagar el LED, y ninguna suscripción porque no tenemos que realizar ninguna acción con la recepción de mensajes.

7. Vamos ahora con la aplicación para Android Things. Reutiliza el código del **Ejercicio:** Suscripción para control remoto del LED. Añade en el método `messageArrived()` la acción para el mensaje "Shake!":

```
case "Shake!":
  Log.d(TAG, "Parpadeo!");
  for (int i = 0; i < 4; i++) {
    mLedGpio.setValue(true);
    Thread.sLeep(500);
    mLedGpio.setValue(false);
    Thread.sLeep(500);
  }
  break;
```

8. Carga el código en el móvil y en la Raspberry y comprueba que funciona.

Preguntas de repaso: MQTT

PARTE 3.

Código Nativo e Ingeniería Inversa

Por Miguel García y Jesús Tomás

CAPÍTULO 6.
Programación en código nativo

Por MIGUEL GARCÍA

Todo el código utilizado hasta ahora en el SDK ha sido desarrollado en Java. El desarrollo en Java presenta grandes ventajas, siendo la más importante que el código obtenido tras compilar el programa (conocido como bytecode) puede ejecutarse en cualquier equipo, independientemente del tipo de procesador. Sin embargo, también presenta inconvenientes. El más importante es la velocidad de ejecución. Los bytecode no pueden ser ejecutados directamente por el procesador, sino que han sido pensados para ser ejecutados en una máquina virtual (JVM). Por lo tanto, estas instrucciones tendrán que ser interpretadas por software, lo que provocará un retardo mayor a diferencia de si estas instrucciones fueran ejecutadas directamente por el hardware.

El desarrollo de aplicaciones en Android no está limitado a Java. También vamos a poder crear aplicaciones formadas por código máquina de un procesador específico. A este tipo de código se le conoce como código nativo. Utilizar esta opción limita nuestra aplicación, dado que solo se podrá ejecutar en un tipo concreto de procesador. No obstante, prácticamente todos los teléfonos móviles y tabletas que se distribuyen con Android en la actualidad tienen el procesador ARM, por lo que esta limitación no es importante. En un futuro es posible que esto cambie. Por ejemplo, algunos smartTV permiten ejecutar aplicaciones Android. Sin embargo, estos dispositivos se basan en otro tipo de procesador y, por lo tanto, no son compatibles con el mismo código nativo para ARM, sino que deberían ser compilados para otro tipo de procesador. En conclusión, si nuestra aplicación tiene código nativo, solo funcionará con un tipo de procesador. Si por el contrario está escrita íntegramente en Java, funcionará en todos los terminales independientemente del procesador. Aunque utilizar código nativo en Android tiene sus inconvenientes, en ciertos casos puede resultar adecuado. Si nuestro software tiene que ejecutar algoritmos de cómputo masivo, su ejecución en la máquina virtual Java puede ser demasiado lenta y, por lo tanto, será más conveniente compilarlo en el código nativo del procesador.

Para poder desarrollar aplicaciones en código nativo debemos utilizar una herramienta conocida como Android NDK (Native Development Kit). Es un conjunto de herramientas que permite incluir en nuestra aplicación código escrito en C o C++ que será compilado a código nativo. NDK se distribuye separadamente del SDK. A lo largo de este capítulo se describirá cómo instalar esta herramienta y cómo utilizarla mediante ejemplos.

La organización que vamos a seguir en esta unidad es la siguiente: en primer lugar, haremos una descripción de las características que nos aporta el entorno de desarrollo de Android NDK, cuándo utilizar código nativo y cuál es el contenido de Android NDK. Seguidamente, se mostrará cómo instalar el entorno de desarrollo de código nativo para Android. Después, será explicado el funcionamiento y la estructura básica de Android NDK. A continuación, veremos la interfaz JNI para poder hacer llamadas desde código Java a código en C/C++ y viceversa; este aspecto será explicado mediante un ejemplo. Por último, presentamos dos ejercicios paso a paso más complejos, para observar el rendimiento de la programación nativa y el procesado de imágenes.

Objetivos:

- Describir las características básicas del kit de desarrollo de Android.
- Conocer cuándo debe utilizarse código nativo en lugar de código Java.
- Analizar el contenido de Android NDK. Conocer las herramientas de desarrollo que aporta al programador, la documentación existente y ejecutar algunas aplicaciones de ejemplo.
- Instalar el kit de desarrollo de código nativo (Android NDK).
- Describir los elementos necesarios para el desarrollo de una aplicación nativa en Android.
- Desarrollar varias aplicaciones para aprender el funcionamiento de la interfaz nativa de Java (JNI).
- Manejar el Android NDK para el procesado de imágenes.

6.1. Android NDK

Android NDK es un conjunto de herramientas que permite incorporar los componentes que hacen uso de código nativo en las aplicaciones de Android.

Las aplicaciones de Android se suelen ejecutar en la máquina virtual Dalvik. El NDK permite implementar parte de sus aplicaciones utilizando código nativo a partir de lenguajes de programación como son C y C++. Esto puede proporcionar beneficios para ciertas clases de aplicaciones, ya sea por la reutilización de código existente y/o por el aumento de velocidad en algunas aplicaciones. Android NDK es:

- Un conjunto de herramientas para crear archivos que se utilizan para generar bibliotecas de código nativo en C y C++.
- Una manera de integrar las bibliotecas nativas en un archivo de paquete de aplicaciones (.apk) que posteriormente puede ser ejecutado en dispositivos Android.
- Un conjunto de cabeceras y bibliotecas nativas del sistema que se apoyarán en todas las futuras versiones de la plataforma Android, a partir de Android 1.5.

La última versión de la NDK es compatible con los conjuntos de instrucciones ARM, x86, MIPS, etc.

Lo más frecuente hoy en día es compilar para todas las arquitecturas soportadas por Android NDK. De esta forma estaremos haciendo nuestra aplicación más compatible con más dispositivos, pero a su vez un poco más pesada debido a que existirá una biblioteca para cada arquitectura de procesador. Al realizar un programa en código nativo, el desarrollador puede crear una aplicación para uno o varios conjuntos de instrucciones (realizando un cambios en el build.gradle o incluyendo dicha información en el fichero Application.mk). La construcción para las diferentes arquitecturas se realizará al mismo tiempo y todo será almacenado en la aplicación final.apk Además Android NDK ofrece encabezados estables para libc (la biblioteca de C), libm (la biblioteca matemática), OpenGL ES (biblioteca de gráficos 3D), la interfaz J NI y otras bibliotecas.

6.1.1. Cuándo utilizar código nativo

El desarrollo de aplicaciones con Android NDK no aporta beneficios en la mayoría de los casos. Como desarrollador, necesitamos encontrar el equilibrio entre beneficios e inconvenientes, ya que el uso de código nativo no se traduce en un aumento de rendimiento automático debido a la mejora en la potencia de los procesadores y la buena integración del API de Android; pero siempre aumenta la complejidad de la aplicación. En general, solo se debe utilizar código nativo si es esencial para su aplicación, no solo porque se prefiera programar en C o C++.

Ejemplos típicos y buenos candidatos para el desarrollo mediante Android NDK son aplicaciones con un uso intensivo de CPU, como el procesamiento de la señal, la simulación de la física, etc. El simple hecho de recodificar un método para ejecutarlo en C no siempre da lugar a un gran aumento de rendimiento. Para saber si debemos desarrollar una aplicación en código nativo o no, lo primero que tenemos que hacer es evaluar las necesidades y comprobar si la API de Android puede proporcionarnos la funcionalidad que necesitamos. Si es así, lo mejor será seguir la API; en caso contrario, podemos ir pensado en desarrollar parte de nuestra aplicación en código nativo. Android NDK, sin embargo, puede ser una forma eficaz de volver a utilizar un gran cuerpo de código existente en C/C++.

Existen dos maneras de utilizar el código nativo en Android:

- Desarrollar nuestra aplicación utilizando el entorno y SDK de Android, y utilizar la interfaz nativa de Java (JNI) para acceder a las API proporcionadas por Android NDK. Esta técnica es la más utilizada ya que proporciona la comodidad de utilizar el entorno de Android; pero además presenta la opción de escribir código nativo cuando sea necesario.
- Desarrollar una aplicación totalmente nativa. Si trabajamos de esta forma, las devoluciones de llamada (callback) de ciclo de vida tendrán que ser en código nativo. El SDK de Android proporciona la clase `NativeActivity`, que es una clase de conveniencia que notifica el código nativo de cualquier actividad de las devoluciones de llamada de ciclo de vida (`onCreate()`, `onPause()`, `onResume()`, etc). Se pueden implementar los callbacks en el código nativo para controlar cuándo se producen estos eventos.

No se puede acceder a las funciones como los servicios y proveedores de contenido de forma nativa, por lo que si desea utilizar las API o cualquier otra utilidad de Android, se deberá usar el JNI.

Enlaces de interés:

Información sobre la clase `NativeActivity`:
http://developer.android.com/reference/android/app/NativeActivity.html

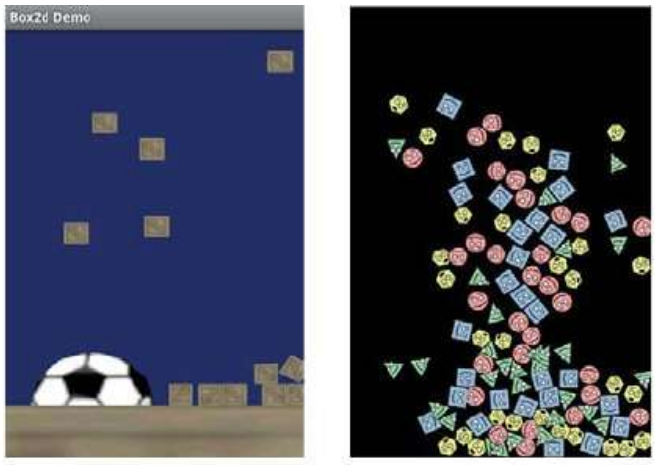
Ejercicio: Diferenciación entre programas desarrollados en código nativo y/o Java.

Para comparar las diferencias de rendimiento entre desarrollar ciertos algoritmos en Java y en código nativo te recomendamos que realices el siguiente ejercicio. Vamos a comparar dos aplicaciones en Android que utilizan la biblioteca Box2D. Box2D es una biblioteca que permite emular interacciones de objetos físicos en un espacio 2D. Esta magnífica biblioteca es de código abierto y ha sido desarrollada por Erin Catto.

1. Descarga de Android Market la aplicación Box2d Demo  . Esta aplicación está basada en la biblioteca JBox2D, una transcripción a Java de la biblioteca de Erin Catto. Pulsa sobre la pantalla para introducir nuevos objetos y observa cómo estos caen atraídos por la gravedad y chocan entre ellos.
2. Descarga ahora del Market la aplicación AndEngineExamples  . Selecciona la opción Physics y luego Using Physics. Igual que antes, pulsa sobre la pantalla para introducir nuevos objetos. Esta aplicación está basada en el motor de Juegos AndEngine que incorpora la biblioteca Box2D; pero esta vez compilada en código nativo.

3. Compara la velocidad de ejecución de ambas aplicaciones. Para ello introduce un número elevado de objetos en la primera aplicación. Observa como a partir de 10 objetos empieza a moverlos con lentitud. Utiliza ahora la aplicación AndEngineExamples y observa como la aplicación no pierde prestaciones, aunque se introduzcan más de 100 objetos.

Nota: Es posible que si utilizas un dispositivo muy rápido no aprecies esta diferencia de velocidad. En dispositivos antiguos la diferencia es abrumadora.



6.1.2. Contenido de Android NDK

El Android NDK contiene las API, documentación y aplicaciones de ejemplo que nos ayudarán a desarrollar aplicaciones en código nativo.

6.1.2.1. Las herramientas de desarrollo

Android NDK incluye un conjunto de herramientas cruzadas (compiladores, enlazadores, etc.) que permiten generar los binarios nativos en las plataformas Linux, OS X y Windows. Esta herramienta de desarrollo proporciona un conjunto de cabeceras del sistema para las API estables nativas que garantizan soporte en todas las versiones posteriores de la plataforma.

El Android NDK también proporciona un sistema de construcción que permite trabajar eficientemente con nuestras fuentes, sin tener que manejar los detalles de toolchain/platform/CPU/ABI. Podemos crear archivos muy cortos que describan las fuentes para compilar y qué aplicación Android las utiliza; el sistema de compilación compila las fuentes y los lugares de las bibliotecas compartidas directamente en el proyecto. A partir de la versión 2.2 de Android Studio 2.2 o posteriores, se puede agregar código C y C++ a nuestra app al realizar una compilación en una biblioteca nativa que Gradle puede empaquetar con nuestra APK haciendo uso de los complementos NDK de Android para la versión 2.2.0 de Gradle o superior. Nuestro código Java podrá llamar a funciones en nuestra biblioteca nativa a través de la interfaz nativa de Java (JNI). A día de hoy, la herramienta de compilación predeterminada de Android Studio para bibliotecas nativas es CMake. Android Studio también es compatible con ndk-build debido al gran número de proyectos existentes que usan el paquete de herramientas de compilación para compilar su

código nativo. Es por ello que en este capítulo estudiaremos la dos formas de trabajar.

6.1.2.2. Documentación

El paquete Android NDK incluye un conjunto de documentación que describe las capacidades del NDK y cómo utilizarlo al crear bibliotecas compartidas para las aplicaciones de Android. En la última versión, la documentación se proporciona en la web https://developer.android.com/ndk/guides/index.html. A continuación se detallan los archivos más comunes (lista parcial):

- **Getting Started with the NDK:** describe cómo instalar y configurar el Android NDK.
- **NDK Programmer's Guide:** proporciona una visión general de las capacidades y del uso del Android NDK.
  - Concepts: muestra conceptos básicos de como utilizar NDK para desarrollar aplicaciones nativas en Android y describe cómo implementar las actividades nativas.
  - Samples: muestra ejemplos de uso de código nativo.
  - Building: describe el uso de la herramienta ndk-build, el uso del archivo Android.mk y del archivo Application.mk.
  - Architectures and CPUs: describe las arquitecturas de CPU soportadas y cómo dirigirse a ellas.
  - Libraries: Describe las APIs estables soportadas, el soporte a C++ e informa acerca de cómo funcionan las bibliotecas predefinidas compartidas y estáticas.
  - Debugging: describe cómo utilizar el depurador de código nativo.

Además de los archivos expuestos en este capítulo existe mucha más documentación en el mismo directorio que puede ser de ayuda a los desarrolladores en caso de que quieran desarrollar una aplicación con algunas características más específicas.

6.1.2.3. Aplicaciones de ejemplo

En este punto vamos a explicar brevemente algunas de las aplicaciones ejemplo que aporta Android NDK. A través de estas aplicaciones podemos ver cómo utilizar el código nativo en aplicaciones Android. Como se ha expuesto en el punto 7.1.2.1 existen dos modos de trabajar con el código nativo en Android. A través de CMake, cuyos aplicaciones ejemplo se encuentran en: https://github.com/googlesamples/android-ndk y haciendo uso de ndk-build cuyas aplicaciones ejemplo se encuentran en https://github.com/googlesamples/android-ndk/tree/android-mk. Algunas de las aplicaciones que encontramos al instalar el Android NDK son:

- hello-jni: aplicación sencilla que carga una cadena de un método nativo implementado en una biblioteca compartida y luego lo muestra en la interfaz de usuario de la aplicación.

- two-libs: aplicación sencilla que carga una biblioteca compartida dinámicamente y llama a un método nativo proporcionado por la biblioteca. En este caso, el método se implementa en una biblioteca estática importado por la biblioteca compartida.
- san-angeles: aplicación sencilla que renderiza gráficos 3D a través de la API nativa de OpenGL ES, mientras que la gestión de la actividad del ciclo de vida es realizada por un objeto `GLSurfaceView`
- hello-gl2: sencilla aplicación que hace un triángulo con OpenGL ES 2.0.
- hello-neon: sencilla aplicación que muestra cómo utilizar la biblioteca `cpufeatures` para comprobar las capacidades de la CPU en tiempo de ejecución, utilizando intrínsecos NEON si son compatibles con la CPU.
- bitmap-plasma: aplicación sencilla que muestra cómo obtener acceso a los buffers de píxeles de los objetos Android `Bitmap` desde el código nativo.
- native-activity: sencilla aplicación que muestra cómo utilizar la biblioteca estática `native-app-glue` para crear una actividad nativa.
- native-plasma: versión de bitmap-plasma implementado con una actividad nativa.

Para cada muestra, la NDK incluye el correspondiente código fuente en C/C++ y el archivo CMakeLists.txt en el caso de usar CMake, y los archivos Android.mk y Application.mk en el caso de utilizar ndk-build. Todos estos archivos se encuentran en bajo la carpeta cpp en el caso de utilizar CMake o bajo la carpeta jni en el caso de los ejemplos con ndk-build.

Preguntas de repaso: Android NDK.

6.2. Instalación de Android NDK

A raíz de la versión 2.2 de Android Studio ya es posible el desarrollo de aplicaciones nativas de manera integrada con el mismo IDE. Hasta la versión 1.4 de Android Studio el desarrollo de aplicaciones nativas no estaba soportado de manera integrada en este IDE y la propia web de desarrolladores de Android recomendaba utilizar Eclipse como IDE para el desarrollo de aplicaciones nativas en Android. Aun así era posible realizar dicho desarrollo con Android Studio. En este punto se van a plasmar los dos modos de instalación de Android NDK. En el punto 7.2.1 se abordará el proceso de instalación en Android Studio 2.2 o superior, ya que se trata de la última forma que recomienda Android para implementar el desarrollo de aplicaciones nativas. Tambien se explicará el proceso en versiones inferiores a Android Studio en el punto 7.2.3, aunque dicho proceso funciona se recomienda el método presentado en el punto 7.2.1.

6.2.1. Instalación Android NDK en Android Studio 2.2 o superior

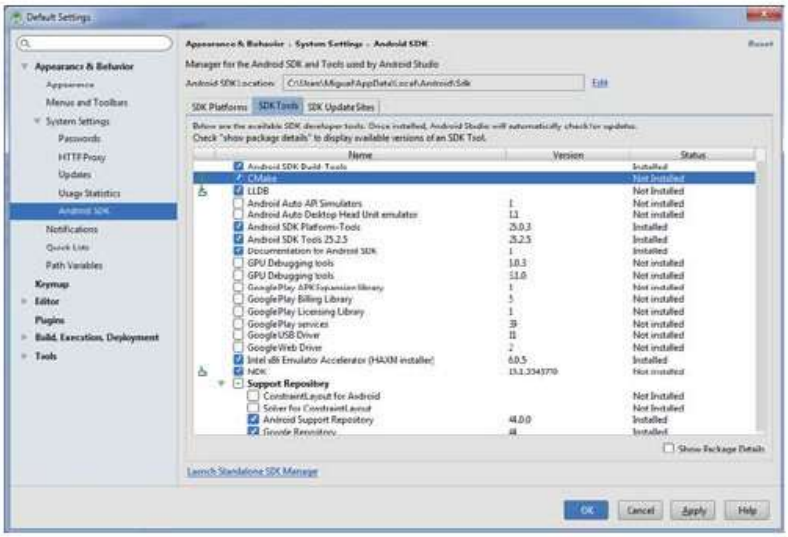
Para poder compilar y depurar el código nativo que exista en nuestra app, se necesitan los siguientes componentes:

- El kit de desarrollo nativo (NDK): conjunto de herramientas que permiten utilizar código C y C++ con Android y proporciona las bibliotecas de cada plataforma que permiten manejar actividades nativas y acceder a componentes de dispositivos físicos, como sensores y entrada táctil.
- CMake: herramienta de compilación externa que funciona junto con Gradle para compilar la biblioteca nativa. No se necesita este componente si solo se planea utilizar ndk-build.
- LLDB: depurador que Android Studio utiliza para depurar código nativo.

La instalación de estos componentes se realizará a través de SDK Manager, como se observa en el siguiente ejercicio.

Ejercicio: Instalación de Android NDK para trabajar con Android Studio 2.2.

1. Desde la ventana de inicio, seleccionamos Configure ➔ SDK Manager o desde un proyecto abierto, seleccionamos desde la barra de herramientas SDK Manager.
2. Hacemos clic en la pestaña SDK Tools.
3. Seleccionamos las casillas LLDB, CMake y NDK tal y como se muestra en la siguiente figura.



4. Hacemos clic en Apply y después en OK.

5. Empezará la descarga y la instalación de las herramientas, tal y como se observa en la siguiente figura.

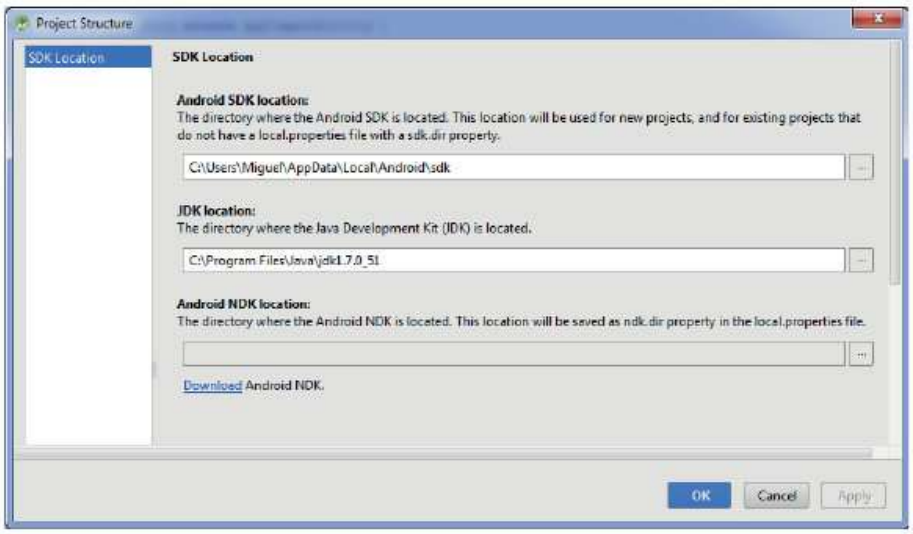
6. Cuando se complete la instalación, haremos clic en Finish y OK.

6.2.2. Instalación Android NDK en Android Studio 2.1 o inferior

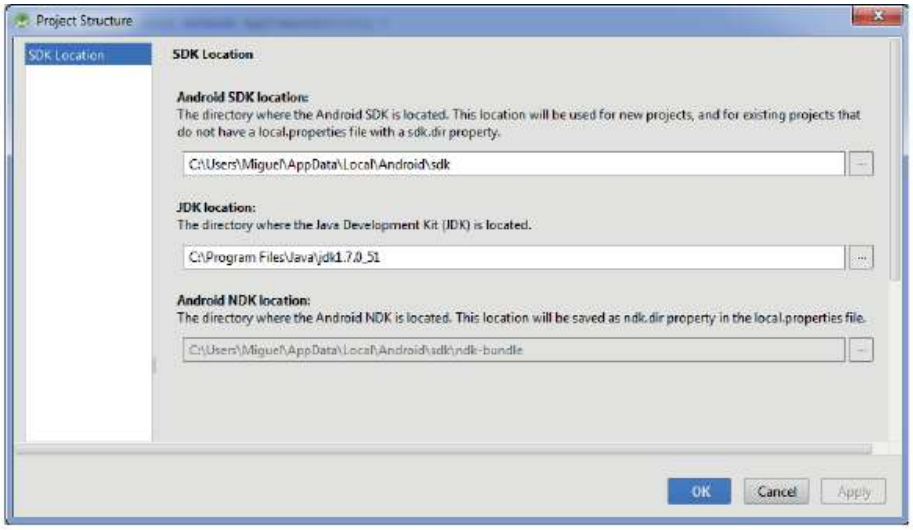
Debemos recordar que este proceso de instalación no es el recomendado por Android. Una vez tenemos el IDE Android Studio 2.1 o inferior instalado y funcionando de forma correcta. Lo primero que debemos realizar es comprobar que el soporte de Android NDK está habilitado en el IDE. Para ello vamos a File ➔ Settings ➔ Plugins. Si no está habilitado el plugin "Android NDK Support" habilitarlo y en caso de que no aparezca actualice el Android Studio a la última versión.

Ejercicio: Instalación de Android NDK en Android Studio 1.5

1. El primer será descargar el Android NDK y guardar el directorio en la variable ndk.dir del fichero local.properties. Esto lo realiza Android Studio de forma automática. Para ello vamos a File ➔ Other Settings ➔ Default Project Structure y pulsamos sobre Download Android NDK (la descarga y la posterior instalación durará unos minutos).

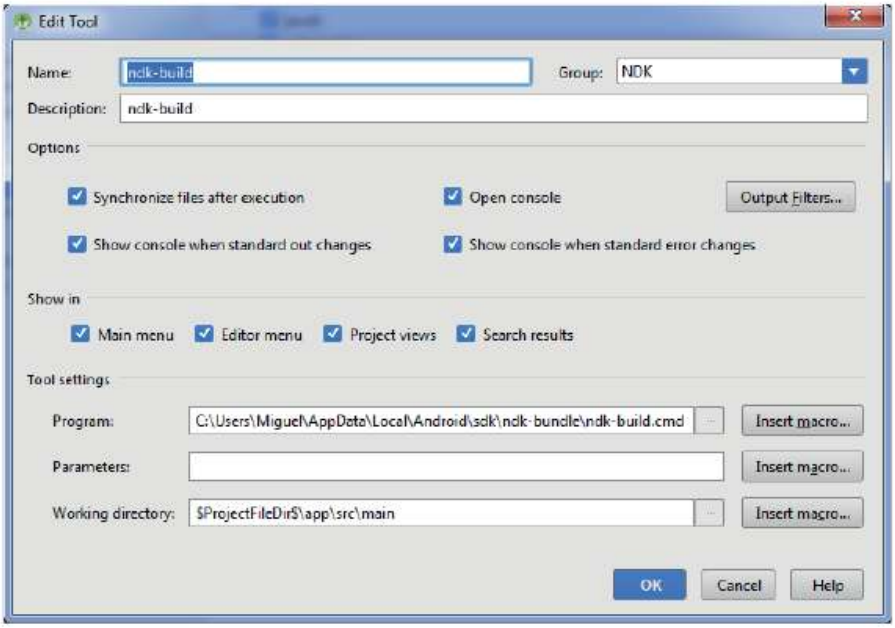


2. Una vez instalado si volvemos a acceder a esta pantalla obtendremos donde está localizado el Android NDK.



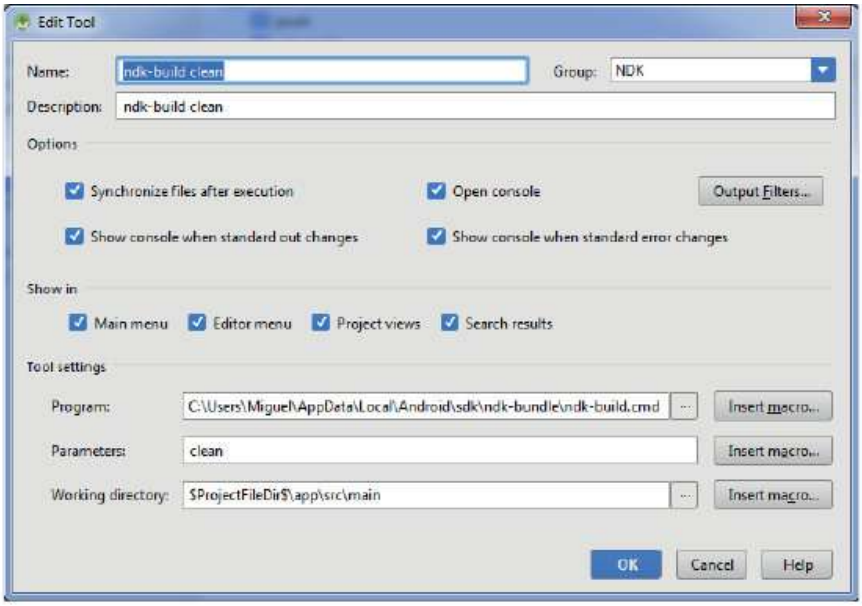
3. Una vez instalado vamos a preparar las herramientas ndk-build para realizar la compilación del código nativo de forma automática desde Android Studio. Para ello iremos a File ➔Settings ➔ Tools ➔ External Tools. Pusaremos sobre el +.

4. Se abrirá una ventana e introduciremos la siguiente información. Name: ndk-build, Group: NDK, Description: ndk-build, habilitaremos todas las opciones y en Tool settings indicaremos donde está la herramienta ndk-build.cmd en el caso de Windows (ndk-build en el caso Linux/Unix), y en el working directory: $ProjectFileDir$\app\src\main (el directorio donde están los archivos en código nativo).



5. A continuación, pulsamos sobre OK y en la pantalla del punto 3 volvemos a crear otra herramienta. Esta será un ndk-build clean, es decir, llamaremos a ndk-build pero con el parámetro clean para realizar el borrado de los binarios

creados anteriormente. La configuración será la misma que en el punto 4 pero indicando como parámetro clean.



6.2.3. Un primer ejemplo con Android NDK

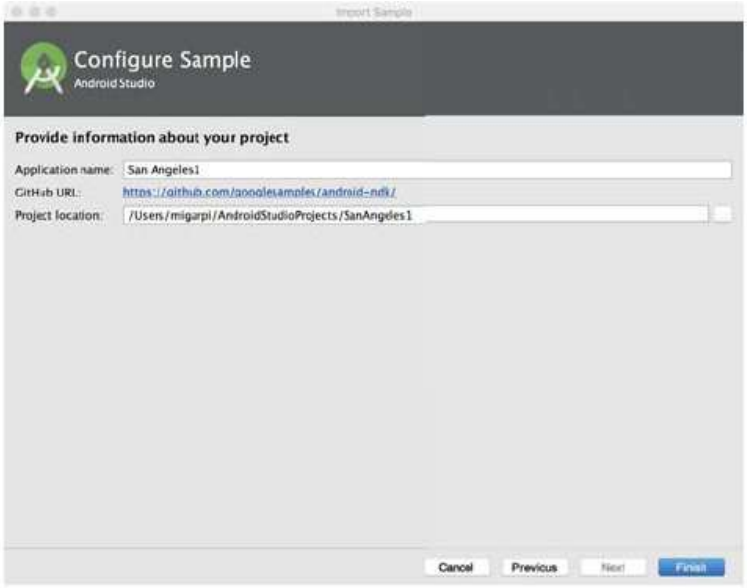
En este punto vamos a ver mediante un ejercicio paso a paso la instalación de Android NDK y la ejecución de un ejemplo del propio NDK.

Ejercicio: Compilación y ejecución de un ejemplo del Android NDK con Android Studio 2.2. o superior.

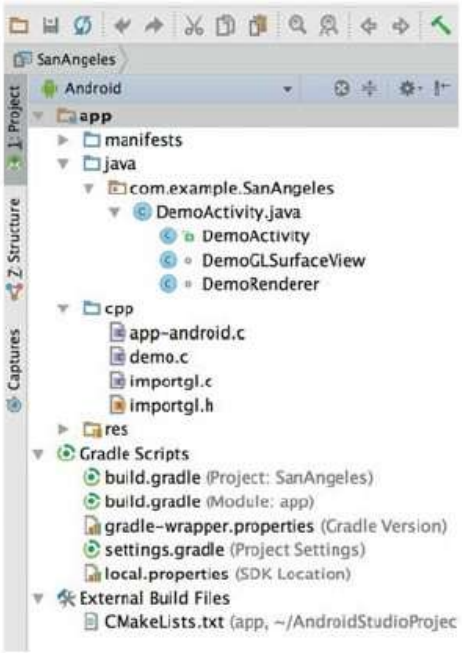
Para compilar un ejemplo del Android NDK, lo primero que debemos hacer es descargar e importar el proyecto correspondiente a Android Studio. Para ello realizaremos las siguientes tareas:

1. Abre nuestro Android Studio. Selecciona File ➔ New ➔ Import Sample.

2. Después selecciona el directorio Ndk y San Angeles y haz clic sobre Next.

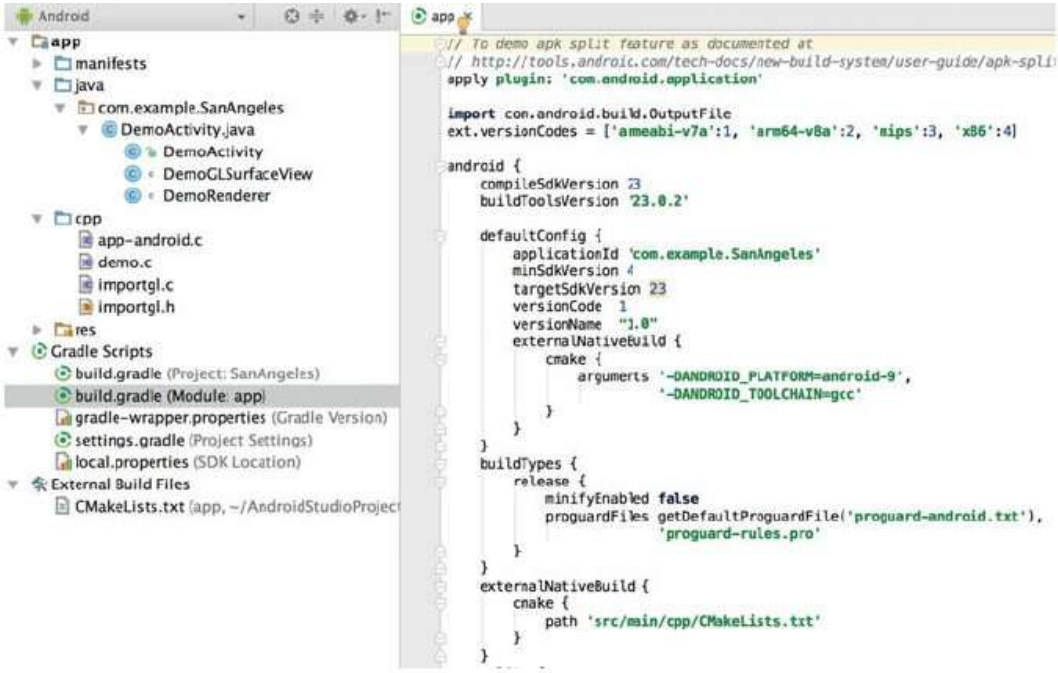


3. A continuación, introducimos el nombre de la aplicación y el lugar donde guardar el proyecto.
4. Finalmente pinchamos sobre Finish.
5. Al finalizar, podremos comprobar que dicho ejemplo se ha copiado en nuestro directorio de trabajo. Cuando Android Studio termine de crear el proyecto, al abrir el subpanel Project del lado izquierdo del IDE y seleccionar la vista de Android, observamos que aparece el directorio cpp y External Build Files.



6. En el directorio cpp podemos encontrar todos los archivos de origen nativos, encabezados y bibliotecas compiladas previamente que forman parte de tu proyecto. En External Build Files puedes encontrar las secuencias de comandos de compilación para CMake o ndk-build. Así como los archivos

build.gradle que indican a Gradle la manera de compilar tu app, CMake o ndk-build. En este ejemplo se utiliza CMake, tal y como se observa en la siguiente figura.



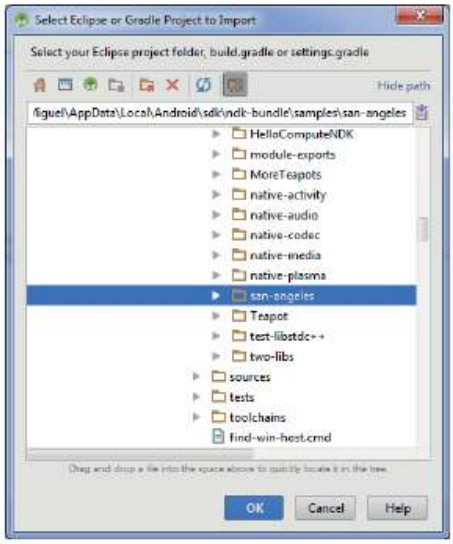
7. Ahora queda compilar y construir la aplicación. Para realizar esta tarea seleccionamos en el menú superior Project ➔ Build Project. Si todo funciona correctamente se creará la aplicación San Angeles.apk.

8. Por último lanzaremos la aplicación con el menú Run ➔ run. Esta aplicación será lanzada al emulador que tenemos configurado o la enviará al terminal móvil. La aplicación final tendrá la siguiente apariencia:

Ejercicio: Compilación y ejecución de un ejemplo del Android NDK en Android Studio 2.1. o inferior.

Para compilar un ejemplo del Android NDK, lo primero que debemos hacer es importar el proyecto correspondiente a Android Studio. Para ello realizaremos las siguientes tareas:

1. Abre nuestro Android Studio. Selecciona File ➔ New ➔ Import Project ▁ Indicamos el directorio donde tengamos el ndk instalado, carpeta samples y seleccionamos san-angeles.



2. Se realizará la importación y una vez importado lo primero que debemos realizar es comprobar que el fichero build.gradle posea el nombre del módulo ndk y la información de los ficheros fuente. En caso de que no insertarlo.

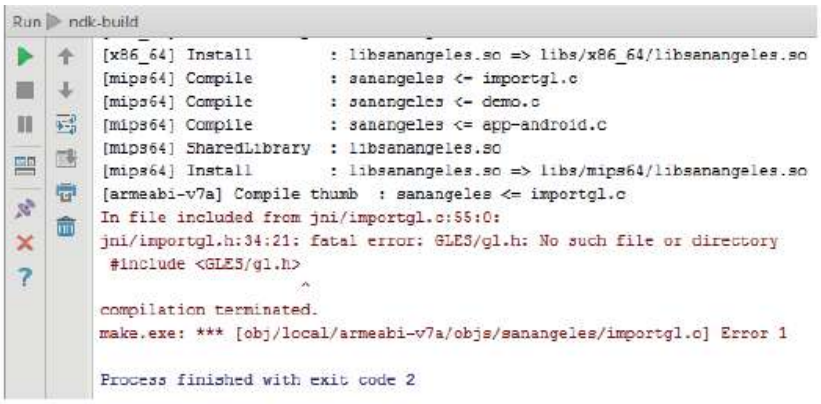
```
defaultConfig {
    applicationId "com.example.SanAngeles"
    minSdkVersion 4
    targetSdkVersion 4

    ndk {
        moduleName "sanangeles"
    }

    sourceSets.main {
        jni.srcDirs = []
        jniLibs.srcDir "src/main/libs"
    }
}
```

3. Lo siguiente será construir el proyecto, para ello pulsaremos Build ➔ Make Project ▁ Aparecerá el siguiente error: "Error:(12, 0) Error: NDK integration is deprecated in the current plugin. Consider trying the new experimental plugin. For details, see http://tools.android.com/tech-docs/new-build-system/gradle-

experimental. Set "android.useDeprecatedNdk=true" in gradle.properties to continue using the current NDK integration".

4. Para solucionarlo introduciremos "android.useDeprecatedNdk=true" en el fichero gradle.properties. En caso de que dicho fichero no exista crearlo.

5. Ahora vamos a compilar el código nativo para que se creen las bibliotecas correspondientes para ello seleccionaremos la vista Project, sobre la carpeta jni pulsaremos botón derecho ➔ NDK ➔ ndk-build. Aparecerá el siguiente error.



6. Esto es debido a que el ejemplo trabaja con instancias a "EGL/egl.h" que a partir de la API 15 de Android ya no se encuentra. Para solucionarlo accede al fichero Application.mk del directortio jni e inserta la siguiente línea "APP_PLATFORM := android-14".

7. Guarda y vuelve a realizar el paso 5.

8. Por último lanzaremos la aplicación con el menú Run. Esta aplicación será lanzada al emulador que tenemos configurado o la enviará al terminal móvil.

6.3. Funcionamiento y estructura de Android NDK

Como hemos indicado anteriormente, Android NDK es un conjunto de herramientas que permite a los desarrolladores de aplicaciones Android incrustar código máquina nativo compilado en C y/o C++ a los archivos de código fuente en sus paquetes de aplicaciones.

Actualmente Android permite la utilización de dos herramientas para el desarrollo de este tipo de aplicaciones. Por un parte CMake y otra ndk-build.

- CMake: es una herramienta multiplataforma de generación o automatización de código. El nombre es una abreviatura para "cross platform make" (make multiplataforma). CMake es una familia de herramientas diseñada para construir, probar y empaquetar software. CMake se utiliza para controlar el proceso de compilación del software usando ficheros de configuración sencillos e independientes de la plataforma. Cmake genera makefiles nativos y espacios de trabajo que pueden usarse en el entorno de desarrollo deseado. El

proceso de construcción se controla creando uno o más ficheros CMakeLists.txt, que contienen diversos comandos para su ejecución.

- Ndk-build: es una secuencia de comandos de shell introducida en Android NDK r4. Cuyo propósito es invocar el script de construcción NDK de forma correcta. Si estamos trabajo con Windows el comando se denomina ndk-build.cmd, mientras que en Linux o MacOS podemos invocarlos como ndk-build desde el directorio de instalación del NDK. Al igual que el comando anterior ndk-build requiere del archivo Andriod.mk y Application.mk (este último opcional) para la ejecución correcta del mismo.

La máquina virtual de Android permite que el código fuente de la aplicación pueda llamar a métodos implementando en código nativo a través de la interfaz J NI. En pocas palabras, esto significa que:

- El código fuente de su aplicación debe ser declarado a través de uno o más métodos con la palabra clave `native`, para indicar que se implementan a través de código nativo. Por ejemplo:

```
native byte[] cargarFichero(String rutaFichero);
```

- Además, se deberá proporcionar una biblioteca compartida nativa que contenga la aplicación de estos métodos, que se empaquetará en la aplicación. apk. Esta biblioteca debe tener el nombre de acuerdo a la norma que veremos en el punto 5, un ejemplo podría ser:

```
libFichero.so
```

- La aplicación deberá cargar explícitamente la biblioteca. Esta debe ser cargada en el inicio de la aplicación. Para realizar esta acción solo debemos añadir el siguiente código fuente, donde Fichero es el nombre de la biblioteca:

```
static {
   System.LoadLibrary ("Fichero");
}
```

6.3.1. Desarrollo práctico de Android NDK con CMake

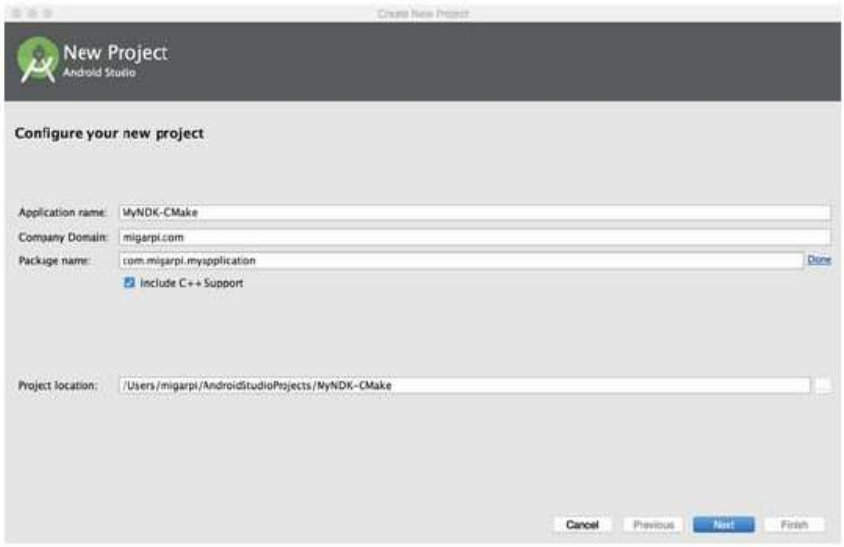
El desarrollo de una aplicación mediante Android NDK con CMake puede realizarse fácilmente a partir de la información expuesta anteriormente y través de los siguientes pasos:

1. Crear un proyecto compatible con C/C++.
2. Seleccionar los parámetros de compilación nativa e inclusión del código nativo en su correspondiente directorio.
3. Generar la llamada a la secuencia de comandos de compilación externa (CMakeLists.txt).
4. Revisar el build.gradle para comprobar la llamada a CMake y CMakeLists.txt.

5. Compilar y ejecutar la aplicación a través de los medios habituales para poder tener la aplicación final.

Ejercicio: Creación de una app nativa simple a través del Wizard y con CMake.

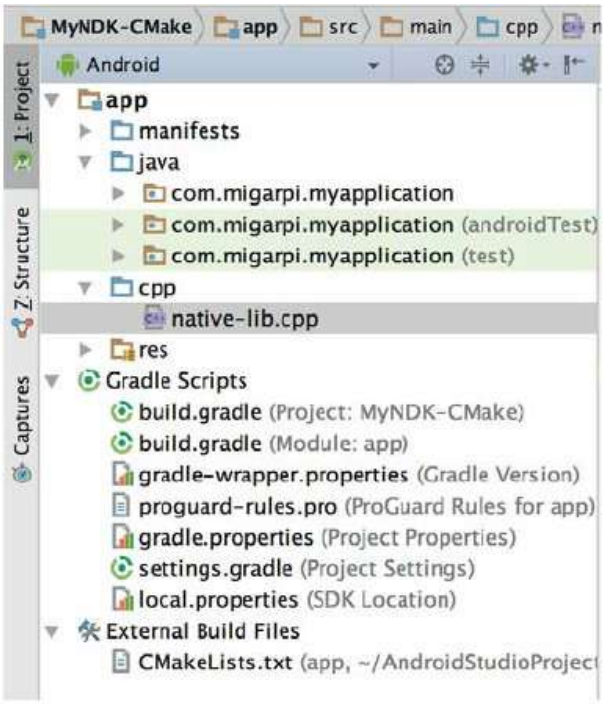
1. Crea un proyecto nuevo con el nombre MyNDK-CMake.
2. Selecciona la casilla Include C++ Support.



3. Haz clic en Next.
4. Completa los otros campos y las siguientes secciones del asistente como de costumbre.
5. En la sección Customize C++ Support del asistente, puedes personalizar tu proyecto con las siguientes opciones:
    a. C++ Standard: usa la lista desplegable para seleccionar la estandarización de C++ que desees usar. Al seleccionar Toolchain Default, se usará la configuración predeterminada de CMake.
    b. Exceptions Support: marca esta casilla y así habilitaremos la compatibilidad con el manejo de excepciones de C++. Al habilitarlo, Android Studio agrega la marca -fexceptions a cppFlags en tu archivo build.gradle, que lo que significa que Gradle le pasará esa información a CMake.
    c. Runtime Type Information Support: marca esta casilla si deseas admitir RTIs. Si se habilita, Android Studio agrega el indicador -frtti a cppFlags en tu de archivo build.gradle, el cual se lo pasará a CMake.
6. Haz clic en Finish.

6.3.1.1. Situación del código nativo

Cuando Android Studio termina de crear el proyecto nuevo, si abres el subpanel Project del lado izquierdo de IDE y seleccionas la vista de Android. Se puede observar que Android Studio ha agrega los directorios cpp y External Build Files. En el grupo cpp puedes encontrar todos los archivos de origen nativos, encabezados y bibliotecas compiladas previamente que forman parte de tu proyecto. Para proyectos nuevos, Android Studio crea un ejemplo de archivo de origen de C++, native-lib.cpp y lo ubica en el directorio src/main/cpp/ de tu módulo de app. Este código de ejemplo proporciona una función simple de C++, stringFromJNI(), que muestra la string "Hello from C++".



Debemos recordar que esta vista no refleja la jerarquía actual de archivos en el disco, pero en ella se agrupan archivos similares para simplificar la navegación de tu proyecto.

6.3.1.2. La herramienta CMake

CMake es una herramienta multiplataforma de generación o automatización de código. CMake es una familia de herramientas diseñada para construir, probar y empaquetar software. Todo el proceso de construcción se controla creando uno o más ficheros CMakeLists.txt, que contienen diversos comandos para su ejecución.

Para entender y poder depurar los posibles problemas de compilación de CMake, es útil conocer los argumentos de compilación específicos que usa Android Studio cuando se compila. Android Studio guarda los argumentos de construcción que utiliza para ejecutar una compilación de CMake, en un archivo denominado cmake_build_command.txt. Android Studio crea una copia para cada ABI con la configuración específica. Para poder mostrar este ejemplo, lo mejor es

desde Android Studio, pulsar 2 veces shit e introducir cmake_build_command.txt. A continuación se observa una ejemplo:

```
Executable
/Users/migarpi/Library/Android/sdk/cmake/3.6.3155560/bin/cmake
arguments
-H/Users/migarpi/AndroidStudioProjects/MyNDK-ndk-build/app
-B/Users/migarpi/AndroidStudioProjects/MyNDK-ndk-
build/app/.externalNativeBuild/cmake/debug/arm64-v8a
-GAndroid Gradle - Ninja
-DANDROlD_ABl=arm64-v8a
-DANDROlD_NDK=/Users/migarpi/Library/Android/sdk/ndk-bundle
-
DCMAKE_LlBRARY_OUTPUT_DlRECTORY=/Users/migarpi/AndroidStudioProjects/MyNDK
-ndk-build/app/build/intermediates/cmake/debug/obj/arm64-v8a
-DCMAKE_BUlLD_TYPE=Debug
-
DCMAKE_MAKE_PROGRAM=/Users/migarpi/Library/Android/sdk/cmake/3.6.3155560/b
in/ninja
-DCMAKE_TOOLCHAlN_FlLE=/Users/migarpi/Library/Android/sdk/ndk-
bundle/build/cmake/android.toolchain.cmake
-DANDROlD_NATlVE_APl_LEVEL=21
-DCMAKE_CXX_FLAGS=-std=c++11 -frtti -fexceptions
jvmArgs
```

A continuación se muestran los principales argumentos de compilación de CMake para Android:

- -G <build-system>: Tipo de archivos de compilación generados por CMake. Para proyectos en Android Studio con código nativo, <build-system> será Android Gradle – Ninja.
- -DANDROID_ABI <abi>: Es la ABI objetivo. Se introduce el nombre de la ABI que queramos, siempre que este soportada por Android NDK.
- -DANDROID_NDK <path>: Ruta absoluta al directorio raíz de la instalación de NDK.
- -DCMAKE_LIBRARY_OUTPUT_DIRECTORY <path>: Ubicación donde CMake coloca los archivos de destino LIBRARY cuando se construye.
- -DCMAKE_BUILD_TYPE <type>: Similar a los tipos de compilación para la herramienta ndk-build . Los valores válidos son Release y Debug.
- -DANDROID_NATIVE_API_LEVEL <level>: Nivel de API de Android que CMake compila.
- -DANDROID_TOOLCHAIN <type>: La cadena de herramientas del compilador que utiliza CMake. Los valores válidos son clang (predeterminado) y gcc (obsoletos).

También es posible hacer uso de variables para realizar llamadas específicas de CMake a través del archivo build.gradle. Estas variables se introducen den-

tro del parámetro cmake {}, tal y como se verá en el punto 7.3.1.4. A continuación se muestran las variables más utilizadas:

- ANDROID_TOOLCHAIN: Especifica la cadena de herramientas del compilador que CMake debería utilizar. Sus posibles argumentos son: gcc o clang.
- ANDROID_PLATFORM: Especifica el nombre de la plataforma Android destino. Por ejemplo, android-18.
- ANDROID_CPP_FEATURES: Especifica ciertas características de C ++ que CMake debe utilizar al compilar su biblioteca nativa. Por ejemplo, rtti (indica que nuestro código utiliza RTTI) y exceptions (indica que nuestro código utiliza excepciones de C++).

Se puede encontrar más información en: https://developer.android.com/ndk/guides/cmake.html

6.3.1.3. Fichero cMakeLists.txt

En el apartado External Build Files puedes encontrar secuencias de comandos de compilación para CMake en el archivo CMakeLists.txt. CMake, al igual que ndk-build, requiere de una secuencia de comandos de compilación para conocer la manera de compilar la biblioteca nativa. Para los proyectos nuevos, Android Studio crea una secuencia de comandos de CMake denominada CMakeLists.txt y la situa en el directorio raíz del módulo.

En esta sección se explican algunos de los comandos básicos que debes incluir en tu secuencia de comandos para indicar a CMake las fuentes que debe usar cuando se cree la biblioteca nativa. Para ello nos fijaremos en el fichero CMakeLists.txt creado, aunque incluremos más conceptos sobre este fichero para el conocimiento del lector.

Para indicar a CMake que cree una biblioteca nativa desde el código fuente nativo, debemos agregar los comandos cmake_minimum_required() y add_library() en nuestra secuencia de comandos de compilación. Al primer comando debemos indicarle la versión mínima del compilador CMake y al comando add_library() primero especificaremos el nombre de la biblioteca, si queremos que sea compartida o estática y donde se encuentran los ficheros con código en C/C++.

```
# Sets the minimum version of CMake required to build your native library.
# This ensures that a certain set of CMake features is available to
# your build.

cmake_minimum_required(VERSlON 3.4.1)

# Specifies a library name, specifies whether the library is STATlC or
# SHARED, and provides relative paths to the source code. You can
# define multiple libraries by adding multiple add.library() commands,
# and CMake builds them for you. When you build your app, Gradle
# automatically packages shared libraries with your APK.

add_library( # Specifies the name of the library.
```

```
    native-lib
# Sets the library as a shared library.
    SHARED

# Provides a relative path to your source file(s).
    src/main/cpp/native-lib.cpp )
```

Cuando agregamos un archivo o una biblioteca de origen a nuestra secuencia de comandos de compilación de CMake usando add_library(), Android Studio también muestra los archivos de cabecera asociados en la vista Project una vez que sincronizas tu proyecto. Sin embargo, para que CMake ubique tus archivos de cabecera durante el tiempo de compilación, debes agregar el comando include_directories() al archivo CMakeLists.txt y especificar la ruta de acceso para tus encabezados:

```
add_library(...)

# Specifies a path to native header files.
include_directories(src/main/cpp/include/)
```

Por convención, el CMake nombra el archivo de nuestra biblioteca de la siguiente manera, siempre y cuando sea una biblioteca compartida:

liblibrary-name.so

Por ejemplo, si especificamos “native-lib” como nombre para nuestra biblioteca compartida en la secuencia de comandos de compilación, CMake crea un archivo llamado libnative-lib.so. Sin embargo, cuando se cargue esta biblioteca en nuestro código Java, usaremos el nombre que especificamos en la secuencia de comandos de compilación de CMake:

```
static {
    System.loadLibrary("native-lib");
}
```

Nota: Si volvemos a nombrar o eliminar una biblioteca en el archivo CMakeLists.txt de CMake, debes limpiar tu proyecto para que Gradle aplique los cambios o elimine la versión anterior de la biblioteca de nuestra APK. Para limpiar nuestro proyecto, seleccionamos Build ➔ Clean Project en la barra de menú.

Uso de bibliotecas de la API de NDK

El NDK de Android proporciona un conjunto de APIs y bibliotecas nativas que pueden resultar muy útiles. Puedes usar cualquiera de estas APIs incluyendo las bibliotecas del NDK en el archivo CMakeLists.txt de tu proyecto. Puedes consultar las bibliotecas nativas de Android NDK en: https://developer.android.com/ndk/guides/stable_apis.html

Las bibliotecas del NDK previamente compiladas ya existen en la plataforma de Android, por lo cual no necesitamos compilarlas ni empaquetarlas en tu APK. Debido a que las bibliotecas del NDK ya forman parte de la ruta de búsqueda de CMake, solo debemos proporcionar a CMake el nombre de la biblioteca que deseemos utilizar y vincularla con nuestra propia biblioteca nativa.

Para ello, agregamos el comando find_library() a nuestra secuencia de comandos de compilación de CMake para disponer una biblioteca del NDK y almacenar su ruta de acceso como una variable. Esta variable se usa para hacer referencia a la biblioteca del NDK en otras partes de la secuencia de comandos de compilación. En el siguiente ejemplo se busca la biblioteca log y se almacena su ruta en log-lib.

```
find_library( # Defines the name of the path variable that stores the
              # location of the NDK library.
              log-lib

              # Specifies the name of the NDK library that
              # CMake needs to locate.
              log )
```

Android, en su web denomina a todas las bibliotecas con una l delante de su nombre, por ejemplo la biblioteca log la denomina llog, pero nosotros debemos indicar solo el nombre sin la primera ele. Por ejemplo si quisiéramos utilizar la biblioteca OpenGL ES 3.0, lo haríamos de la siguiente manera.

```
find_library( # Defines the name of the path variable that stores the
              # location of the NDK library.
              GLESv3-lib

              # Specifies the name of the NDK library that
              # CMake needs to locate.
              GLESv3 )
```

Para que tu biblioteca nativa llame a funciones de la biblioteca log, debemos vincular las bibliotecas usando el comando target_link_libraries(), tal que así:

```
find_library(...)

# Links your native library against one or more other native libraries.
target_link_libraries( # Specifies the target library.
                       native-lib

                       # Links the log library to the target library.
                       ${log-lib} )
```

El NDK también incluye algunas bibliotecas como código fuente. La cuales debemos compilar y vincular a nuestra biblioteca nativa. Podemos compilar el código fuente de una biblioteca nativa usando el comando add_library(), pero deberemos de proporcionar una ruta dicho código fuente. El siguiente comando indica a CMake que compile android_native_app_glue.c, que administra eventos de ciclo de vida de NativeActivity y la entrada táctil en una biblioteca estática y por último lo vincule a native-lib:

```
add_library( app-glue
             STATIC
             ${AN-
DROID_NDK}/sources/android/native_app_glue/android_native_app_glue.c )

# You need to link static libraries against your shared native library.
```

```
target_link_libraries( native-lib app-glue ${log-lib} )
```

Agregar otras bibliotecas compiladas previamente

El proceso de agregar una biblioteca compilada previamente es similar al de especificar otra biblioteca nativa para que CMake realice la compilación. Sin embargo, debido a que la biblioteca ya está compilada, debemos hacer uso de la palabra clave IMPORTED para indicar a CMake que solo deseamos importar la biblioteca a nuestro proyecto:

```
add_library( imported-lib
             SHARED
             lMPORTED )
```

Luego debes especificar la ruta de acceso a la biblioteca con el comando set_target_properties(), como se muestra a continuación:

Algunas bibliotecas proporcionan paquetes separados para arquitecturas de CPU específicas, o interfaces binarias de aplicación (ABI), y las organiza en directorios separados. Este enfoque permite que las bibliotecas aprovechen determinadas arquitecturas de CPU y, al mismo tiempo, dan la opción de utilizar solo las versiones de bibliotecas que deseemos. Para agregar varias versiones de ABI de una biblioteca a nuestro CMakeLists.txt sin necesidad de escribir varios comandos para cada versión de la biblioteca, puedes usar la variable de ruta de acceso ANDROID_ABI. Esta variable usa una lista de las ABI predeterminadas que el NDK admite, o una lista filtrada de ABI que podemos configurar manualmente para que Gradle la use, aspecto que se verá más adelante.

```
add_library(...)
set_target_properties( # Specifies the target library.
                       imported-lib

                       # Specifies the parameter you want to define.
                       PROPERTlES lMPORTED_LOCATlON

                       # Provides the path to the library you want to im-
port.
                       imported-lib/src/${ANDROlD_ABl}/libimported-lib.so
)
```

Al igual que habíamos comentado anteriormente, si queremos ubicar los archivos de cabeceras durante el tiempo de compilación, debes usar el comando include_directories() e incluir la ruta de acceso a tus archivos de cabecera:

```
include_directories( imported-lib/include/ )
```

Nota: Si deseamos empaquetar una biblioteca compilada previamente que no sea una dependencia de tiempo de compilación (por ejemplo, al agregar una biblioteca compilada previamente que sea una dependencia de imported-lib), no es necesario que apliques las siguientes instrucciones para vincular la biblioteca.

Para vincular la biblioteca compilada previamente, debemos de incluirla en el comando target_link_libraries(), tal que así.

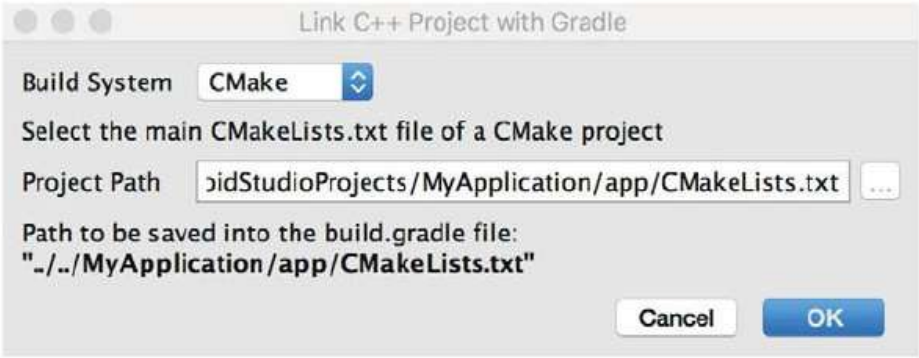
```
target_link_libraries( native-lib imported-lib app-glue ${log-lib} )
```

6.3.1.4. Fichero build.gradle

Los archivos build.gradle indican a Gradle la manera de compilar tu app, ya sea con CMake o ndk-build.

Para el caso de CMake, que es el estudiado en este punto, se puede establecer que Gradle se vincule con nuestra biblioteca nativa haciendo usando la IU de Android Studio, para ello:

1. Abrimos el subpanel Project del lado izquierdo de IDE y seleccionamos la vista de Android.
2. Hacemos clic con el botón secundario en el módulo que desees vincular con nuestra biblioteca nativa (por ejemplo, el módulo de app) y selecciona Link C++ Project with Gradle en el menú.
3. Seleccionamos CMake, en el campo junto a Project Path debemos indicar el archivo de secuencia de comandos CMakeLists.txt de tu proyecto de CMake externo.



También es posible realizar esta vinculación de forma manual. Pare ellos debemos de debemos agregar el bloque externalNativeBuild {} al archivo build.gradle y configurarlo con cmake {} e incluir el path del archivo CMakeLists.txt creado con anterioridad.

```
android {
  ...
  defaultConfig {...}
  buildTypes {...}

  // Encapsulates your external native build configurations.
  externalNativeBuild {

    // Encapsulates your CMake build configurations.
    cmake {

      // Provides a relative path to your CMake build script.
      path "CMakeLists.txt"
    }
  }
}
```

Especificar las ABI

De forma predeterminada, Gradle compila nuestra biblioteca nativa en archivos separados .so para las ABI que el NDK admite y las empaqueta en nuestra APK. Si deseamos que Gradle compile y empaquete solo determinadas configuraciones de ABI de nuestras bibliotecas nativas, puedemos especificarlas con el indicador ndk.abiFilters en nuestro archivo build.gradle, tal y como se muestra a continuación:

```
android {
  ...
  defaultConfig {
    ...
    externalNativeBuild {
      cmake {...}
      // or ndkBuild {...}
    }

    ndk {
      // Specifies the ABl configurations of your native
      // libraries Gradle should build and package with your APK.
      abiFilters 'x86', 'x86_64', 'armeabi', 'armeabi-v7a',
                  'arm64-v8a'
    }
  }
  buildTypes {...}
  externalNativeBuild {...}
}
```

También podemos especificar argumentos e indicadores opcionales para CMake configurando otro bloque externalNativeBuild {} dentro del bloque defaultConfig {} de nuestro build.gradle. Como en el caso de otras propiedades del bloque defaultConfig {}, puedes anular estas propiedades para cada clase de producto de tu configuración de compilación. Asi como incluir algunos argumentos del comando CMake (ver https://developer.android.com/ndk/guides/cmake.html).

Por ejemplo, si nuestro proyecto CMake define varias bibliotecas nativas, podemos utilizar la propiedad targets para compilar y empaquetar solo un subconjunto de esas bibliotecas para una clase de producto dado. En el siguiente ejemplo de código se describe alguna de las propiedades que se pueden configurar:

```
android {
  ...
  defaultConfig {
    ...
    // This block is different from the one you use to link Gradle
    // to your CMake or ndk-build script.
    externalNativeBuild {

      // For ndk-build, instead use ndkBuild {}
      cmake {

        // Passes optional arguments to CMake.
```

```
        arguments "-DANDROID_ARM_NEON=TRUE", "-DANDROID_TOOLCHAIN=clang"

        // Sets optional flags for the C compiler.
        cFlags "-D_EXAMPLE_C_FLAG1", "-D_EXAMPLE_C_FLAG2"

        // Sets a flag to enable format macro constants for the C++ com-
piler.
        cppFlags "-D_STDC_FORMAT_MACROS"
      }
    }
  }

  buildTypes {...}

  productFlavors {
    ...
    demo {
      ...
      externalNativeBuild {
        cmake {
          ...
          // Specifies which native libraries to build and package for
this
          // product flavor. If you don't configure this property, Gradle
          // builds and packages all shared object libraries that you de-
fine
          // in your CMake or ndk-build project.
          targets "native-lib-demo"
        }
      }
    }

    paid {
      ...
      externalNativeBuild {
        cmake {
          ...
          targets "native-lib-paid"
        }
      }
    }
  }

  // Use this block to link Gradle to your CMake or ndk-build script.
  externalNativeBuild {
    cmake {...}
    // or ndkBuild {...}
  }
}
```

6.3.1.5. Compilar y ejecutar una app con código nativa

En este apartado vamos a analizar el proceso que realiza Android Studio al compilar y ejecutar una aplicación con código nativo.

Cuando hacemos clic en Run, Android Studio crea y lanza una app que muestra el texto "Hello from C++" en tu dispositivo o emulador de Android. Los eventos que tienen lugar para compilar y ejecutar la app con código nativo son los siguientes:

1. Gradle llama a la secuencia de comandos de compilación externa, en este caso CMakeLists.txt.

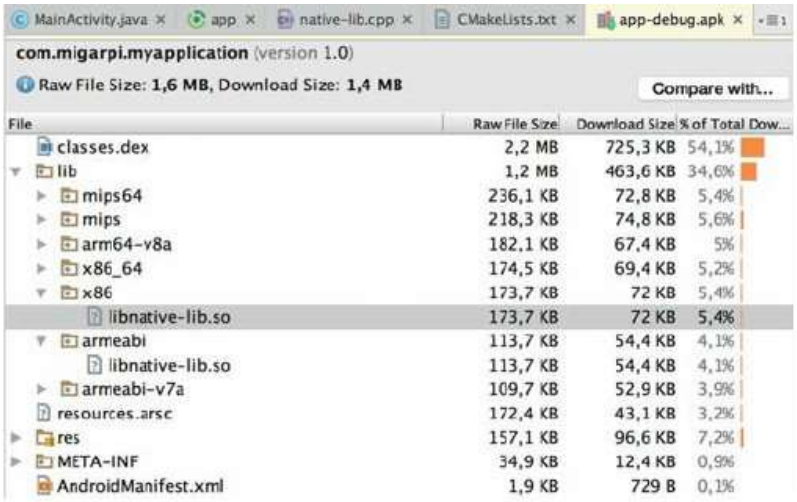
2. CMake sigue los comandos en la secuencia de comandos de compilación para compilar un archivo de origen de C++ denominado native-lib.cpp, en una biblioteca de objetos compartidos y la crea la biblioteca compartida libnative-lib.so. Luego, Gradle la empaqueta en el APK.

3. Durante la ejecución, la MainActivity de la app carga la biblioteca nativa usando System.loadLibrary(). La función nativa de la biblioteca, stringFromJNI(), quedará disponible para la app.

4. MainActivity.onCreate() llama a stringFromJNI(), que muestra "Hello from C++", y la usa para actualizar el TextView.

El Instant Run no es compatible con proyectos que usan código nativo, es por ello que Android Studio inhabilita la característica de manera automática.

Para comprobar si el Gradle ha empaquetado las bibliotecas nativas en el APK vamos a utilizar el analizador de APK. Para ello, seleccionamos Build ➔ Analyze APK y después seleccionamos la APK del directorio app/build/outputs/apk/ y hacemos clic en OK. Como podemos observar existe una biblioteca para cada una de las arquitecturas de los procesadores que soportan el desarrollo en NDK de Android.



6.3.2. Desarrollo práctico de Android NDK con ndk-build

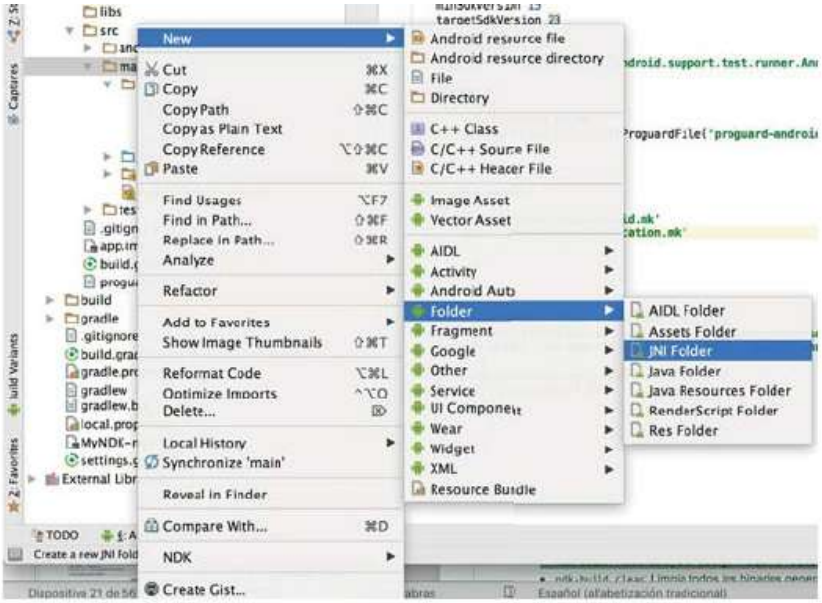
El desarrollo de una aplicación mediante Android NDK con ndk-build puede realizarse a través de cuatro pasos básicos:

1. Situar el código nativo.
2. Describir las fuentes para la construcción NDK (Android.mk).
3. Describir opciones acerca del compilar, tipo de CPU, etc. a través de Application.mk (opcional).
4. Revisar/Modificar el build.gradle para realizar la llamada a ndk-build y Android.mk y Application.mk.
5. Compilar y ejecutar la aplicación a través de los medios habituales para poder tener la aplicación final.

6.3.2.1. Situación del código fuente nativo

El código nativo escrito en C o en C++ y todo lo relacionado con el desarrollo de aplicaciones nativas deben estar situados en el directorio jni de nuestro proyecto. Este directorio no existe por defecto; por tanto, debe ser creado por el desarrollador.

Este directorio cuelga directamente del proyecto. Así, la situación será $PROYECTO/jni/ donde $PROYECTO y corresponde a la ruta del proyecto Android. Para crear el directorio JNI. Simplemente desde la vista "Project" y dentro del directorio "main", botón secundario New ➔ Folder ➔ JNI Folder.



Dentro del directorio jni se puede organizar la información como quiera el desarrollador, ya que los nombres de directorios y archivos no van a influir en el paquete final de la aplicación. Lo único a tener en cuenta es que no pueden utilizar pseudónimos como com.<miempresa>.<miproyecto> ya que dicho nombre sí interfiere en el APK.

6.3.2.2. La herramienta ndk-build

El Android NDK r4 introdujo un nuevo shell script, llamado ndk-build, para simplificar la construcción del código máquina. Este script se encuentra en el directorio de nivel superior del NDK, y deberá llamarse desde la línea de comandos en el directorio principal del proyecto de nuestra aplicación. Por ejemplo:

```
cd $PROYECTO
ndk-build
```

La herramienta ndk-build posee varias opciones que vamos a analizar a continuación:

- `ndk-build`: Reconstruye el código máquina requerido.
- `ndk-build clean`: Limpia todos los binarios generados.
- `ndk-build NDK_DEBUG=1`: Genera código nativo depurable, mediante la generación de binarios de depuración.
- `ndk-build NDK_DEBUG=0`: Genera código nativo depurable, mediante la liberación de binarios.
- `ndk-build V=1`: Lanza el build mostrando comandos integrados.
- `ndk-build -B`: Fuerza una reconstrucción completa.
- `ndk-build -B V=1`: Fuerza una reconstrucción completa y muestra los comandos.
- `ndk-build NDK_LOG=1`: Muestra un display interno para los mensajes de log de Android NDK.
- `ndk-build NDK_HOST_32BlT=1`: Utiliza siempre herramientas de 32 bits.
- `ndk-build NDK_APPLlCATlON_MK = <archivo>`: Reconstruye mediante una Application.mk específica apuntada por la variable `NDK_APPLlCATlON_MK`.
- `ndk-build -C <proyecto>`: Construye el código nativo para el proyecto situado en el path <proyecto>. Esta opción es útil si no se quiere utilizar el comando `cd` en elterminal.

6.3.2.3. Fichero Android.mk

El archivo Android.mk describe las fuentes de compilación al sistema. Más concretamente se trata de:

- Un pequeño fragmento del makefile de GNU que es analizado una o más veces por el sistema de compilación. Se debe intentar minimizar las variables declaradas en este fichero y dar por válida cualquier suposición que no haya sido definida durante el análisis.
- La sintaxis de los archivos se ha diseñado para permitir que su grupo de fuentes sean módulos. Un módulo es considerado como una:
    - Biblioteca estática.
    - Biblioteca compartida.

Solo las bibliotecas compartidas se instalarán/copiarán en el paquete de la aplicación (.psk). Las bibliotecas estáticas, sin embargo, se pueden utilizar para

generar bibliotecas compartidas. Podemos definir uno o más módulos en cada archivo Android.mk, y se puede utilizar el mismo archivo de origen en varios módulos. El sistema de construcción se encarga de todos los detalles por nosotros. Por ejemplo, no es necesario enumerar los archivos de cabecera o dependencias explícitas entre los archivos dentro del archivo Android.mk. El sistema de construcción de Android NDK es el que se encarga de realizar estas acciones automáticamente.

A continuación se muestra un ejemplo de un fichero Android.mk. Sobre el cual se explicarán las funcionalidades comando.

```
LOCAL_PATH = $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE = sanangeles

LOCAL_CFLAGS = -DANDROlD_NDK \
                -DDlSABLE_lMPORTGL

LOCAL_SRC_FlLES = \
    importgl.c \
    demo.c \
    app-android.c \

LOCAL_LDLlBS = -lGLESv1_CM -ldl -llog

include $(BUlLD_SHARED_LlBRARY)
```

Ahora, vamos a explicar estas líneas. La primera de ellas es:

```
    LOCAL_PATH = $(call my-dir)
```

Un archivo Android.mk debe comenzar con la definición de la variable `LOCAL_PATH`. Se utiliza para localizar los archivos de origen en el árbol de directorios de desarrollo. En este ejemplo, la función macro `my-dir`, proporcionada por el sistema de construcción, se utiliza para devolver la ruta del directorio actual (es decir, el directorio que contiene el propio archivo Android.mk).

La siguiente línea es:

```
    include $(CLEAR_VARS)
```

La variable `CLEAR_VARS` es proporcionada por el sistema de construcción y apunta a un makefine especial de GNU, que liberará distintas variables del tipo `LOCAL_XXX (LOCAL_MODULE, LOCAL_SRC_FlLES, LOCAL_STATlC_LlBRARlES, etc.)` con la excepción de la variable `LOCAL_PATH`. Esto es necesario porque todos los archivos de control de compilación se analizan en un solo contexto de ejecución make GNU donde todas las variables son globales.

Después encontramos:

```
    LOCAL_MODULE = sanangeles
```

La variable `LOCAL_MODULE` debe definirse para identificar cada módulo descrito en el archivo Android.mk. El nombre debe ser único y no contener ningún espacio.

Además, hay que tener en cuenta que el sistema de construcción agregará automáticamente el prefijo y el sufijo adecuado. Es decir, el módulo de biblioteca compartida denominada sanangeles generará «libsanangeles.so».

Si el nombre de nuestro módulo es libsanangeles, el sistema de construcción no añadirá otro prefijo «lib» y generará «libsanangeles.so».

A continuación tenemos:

```
    LOCAL_CFLAGS = -DANDROlD_NDK \
                   -DDlSABLE_lMPORTGL
```

La variable `LOCAL_CFLAGS` es un conjunto opcional de parámetros del compilador que se utilizará en la construcción de los archivos de código fuente C y/o C++. Esto puede ser útil para especificar las definiciones de macros adicionales u opciones de compilación. En nuestro caso utilizaremos el Android NDK para deshabilitar la importación de la biblioteca de gráficos.

La siguiente variable que presenta este ejemplo es:

```
    LOCAL_SRC_FlLES = \
        importgl.c \
        demo.c \
        app-android.c \
```

La variable `LOCAL_SRC_FlLES` debe contener una lista de los archivos de código C y/o C++ que van a ser compilados y montados en este módulo. Cabe tener en cuenta que aquí no debemos incluir los ficheros cabecera del módulo. El propio sistema de construcción será el encargado de calcular las dependencias automáticamente por nosotros; solo pasaremos una lista de los archivos de código fuente al compilador. Tenemos que tener en cuenta que la extensión por defecto para los archivos de código fuente de C++ es .cpp. Sin embargo, es posible especificar una extensión diferente si modificamos la variable `LOCAL_CPP_EXTENSlON`.

Luego tenemos la variable:

```
    LOCAL_LDLlBS = -lGLESv1_CM -ldl -llog
```

La variable `LOCAL_LDLlBS` indica una lista de enlazadores adicionales que se utilizarán en la construcción del módulo. Esto es útil para pasar el nombre de las bibliotecas del sistema específicas con el prefijo «-l».

Por último, tenemos:

```
    include $(BUlLD_SHARED_LlBRARY)
```

Esta variable es proporcionada por el sistema de construcción que apunta a un script makefile de GNU que es el encargado de recoger toda la información que ha definido en las variables `LOCAL_XXX` desde el último `include $(CLEAR_VARS)` y de determinar qué construir y cómo hacerlo exactamente. También hay la llamada `BUlLD_STATlC_LlBRARY` para generar una biblioteca estática.

Por último, debemos indicar que existen más variables que pueden ser definidas en el archivo Android.mk. Creemos que las expuestas en este punto son las más relevantes; pero si se quiere averiguar más sobre las variables que pueden ser incluidas en este archivo podemos ir a la documentación de Android NDK.

6.3.2.4. Fichero Application.mk (opcional)

El propósito del fichero Application.mk es describir los módulos nativos que son requeridos en nuestra aplicación. Este fichero está situado dentro del directorio jni de nuestro proyecto.

El archivo Application.mk es realmente un fragmento diminuto GNU makefile que debe definir algunas variables, como:

- `APP_PROJECT_PATH`: Esta variable debería dar la * ruta absoluta * a nuestro directorio raíz del proyecto de la aplicación. Esto se utiliza para copiar/instalar versiones livianas de las bibliotecas compartidas JNI para concretar la ubicación específica de las herramientas de generación de nuestra APK.
- `APP_MODULES`: Esta variable es opcional. Si es definida por el desarrollador, el compilador de NDK seleccionará por defecto todos los declarados en nuestro archivo Android.mk. Si no es definida por el desarrollador, el compilador de NDK seleccionará todos los módulos por defecto declarados en nuestro archivo Android.mk.
- `APP_OPTIM`: Esta variable opcional se puede definir como release o debug. Se utiliza para modificar el nivel de optimización cuando se están compilando los módulos de la aplicación. El modo release es el modo predeterminado; por tanto, si no está definido, este será el modo utilizado.
- `APP_CFLAGS`: Son un conjunto de indicadores del compilador C al compilar cualquier código fuente C o C++ de cualquiera de los módulos. Se puede utilizar para cambiar la construcción de un módulo dependiendo de la aplicación que lo necesite, en lugar de modificar el propio archivo Android.mk.
- `APP_CPPFLAGS`: Posee las mismas características que `APP_CFLAGS`; pero en este caso orientado a C++.
- `APP_BUILD_SCRIPT`: Por defecto, el sistema de construcción NDK buscará un archivo llamado Android.mk situado en `$(APP_PROJECT_PATH)/jni`. Si desea cambiar este comportamiento, podemos definir el `APP_BUILD_SCRIPT` para apuntar a un script de construcción alternativo.
- `APP_ABI`: Por defecto, el sistema de construcción NDK generará código máquina para el ABI 'armeabi'. Esto corresponde a una ARMv5TE basado en una CPU con soporte de operaciones en punto flotante. Podemos utilizar `APP_ABI` para seleccionar una arquitectura diferente. Por ejemplo, para el soporte de dispositivos basados en ARMv7, podríamos indicar: `APP_ABI = armeabi-v7a`. Si quisiéramos soporte para todas las arquitecturas indicaríamos `AAP_ABI = armeabi armeabi-v7a x86 mips` o `AAP_ABI = all`.
- `APP_PLATFORM`: Indica el nombre de la plataforma Android de destino. Por ejemplo, `android-3` correspondería a la API 3 de Android (v1.5).
- `APP_SHORT_COMMANDS`: Es el equivalente de `LOCAL_SHORT_COMMANDS` para todo el proyecto.

- `APP_PlE`: A través de esta opción se hace más difícil explotar los errores de corrupción de memoria en la ubicación aleatoria del código.

A continuación se muestra un ejemplo de un fichero Application.mk.

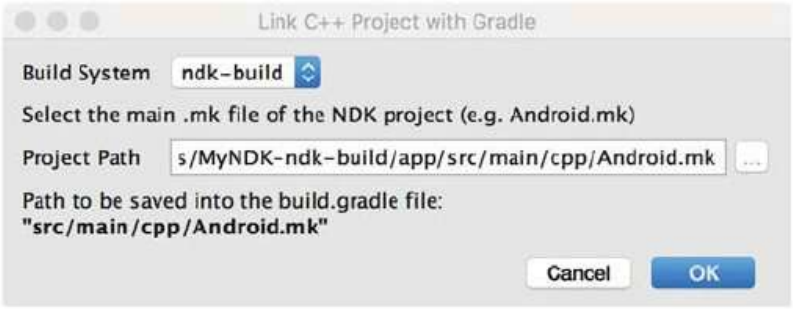
```
# The ARMv7 is significanly faster due to the use of the hardware FPU
APP_ABl = armeabi armeabi-v7a
APP_PLATFORM = android-8
```

Existen más variables que pueden ser definidas en el archivo Application.mk. Creemos que las expuestas en este punto son las más relevantes; pero si se quiere averiguar más sobre las variables que pueden ser incluidas en este archivo podemos ir a la documentación de Android NDK.

6.3.2.5. Fichero build.gradle

Los archivos build.gradle indican a Gradle la manera de compilar tu app, ya sea con CMake o ndk-build. Para el caso de ndk-build, se puede establecer que Gradle se vincule con nuestra biblioteca nativa haciendo usando la IU de Android Studio, para ello:

1. Abrimos el subpanel Project del lado izquierdo de IDE y seleccionamos la vista de Android.
2. Hacemos clic con el botón secundario en el módulo que desees vincular con nuestra biblioteca nativa (por ejemplo, el módulo de app) y selecciona Link C++ Project with Gradle en el menú.
3. Seleccionamos ndk-build, y utilizamos el campo junto a Project Path para especificar el archivo Android.mk del proyecto ndk-build externo. Android Studio también incluye el archivo Application.mk si se encuentra en el mismo directorio que tu archivo Android.mk.



También es posible realizar esta vinculación de forma manual. Pare ellos debemos de debemos agregar el bloque externalNativeBuild {} al archivo build.gradle y configurarlo con ndkBuild {} e incluir el path de los archivos Android.mk y Application.mk creados con anterioridad.

```
    defaultConfig {

    }
    buildTypes {

        }
```

```
    }
    externalNativeBuild {
        ndkBuild {
            path 'src/main/cpp/Android.mk'
        }
    }
```

Al igual que vimos con CMake, de forma predeterminada, Gradle compila nuestra biblioteca nativa en archivos separados .so para las ABI que el NDK admite y las empaqueta en nuestra APK. Si deseamos que Gradle compile y empaquete solo determinadas configuraciones de ABI de nuestras bibliotecas nativas, puedemos especificarlas con el indicador ndk.abiFilters en nuestro archivo build.gradle, tal y como se muestra a continuación:

```
android {
  ...
  defaultConfig {
    ...
    externalNativeBuild {
      ndkBuild {...}
    }
    ndk {
      // Specifies the ABl configurations of your native
      // libraries Gradle should build and package with your APK.
      abiFilters 'x86', 'x86_64', 'armeabi', 'armeabi-v7a',
                  'arm64-v8a'
    }
  }
}
```

6.3.2.6. Compilar y ejecutar una app con código nativa

En este apartado vamos a analizar el proceso que realiza Android Studio al compilar y ejecutar una aplicación con código nativo con la herramienta ndk-build.

Cuando hacemos clic en Run, Android Studio crea y lanza la app desarrollada. Los eventos que tienen lugar para compilar y ejecutar la app con código nativo son los siguientes:

1. Gradle llama a la secuencia de comandos de compilación externa, en este caso Android.mk y Application.mk.

2. ndk-build sigue los comandos en la secuencia de comandos para compilar un archivo de origen de C/C++, en una biblioteca de objetos compartidos y crea tantas bibliotecas como ABI hayamos incluido. Luego, Gradle la empaqueta en el APK.

3. Durante la ejecución, la MainActivity de la app carga la biblioteca nativa usando System.loadLibrary(). La función nativa de la biblioteca quedará disponible para la app y cuando se llame al método nativo se ejecutará la acción

El Instant Run no es compatible con proyectos que usan código nativo, es por ello que Android Studio inhabilita la característica de manera automática.

Para comprobar si el Gradle ha empaquetado las bibliotecas nativas en el APK vamos a utilizar el analizador de APK. Para ello, seleccionamos Build ➔ Analyze APK y después seleccionamos la APK del directorio app/build/outputs/apk/ y hacemos clic en OK. Como podemos observar existe una biblioteca para cada una de las arquitecturas de los procesadores que soportan el desarrollo en NDK de Android.

Ejercicio: Ejercicio paso a paso con CMake.

Vamos a desarrollar una aplicación muy sencilla denominada SumaNDK para ver el funcionamiento de la programación NDK con CMake.

1. Crea un proyecto con las siguientes definiciones:

```
Application name SumaNDK
Project name SumaNDK
Package name com.sumandk
Activity name MainActivity
Layout name activity_main
Resto de parámetros por defecto.
```

2. Modifica el fichero layout activity main.xml para que posea el siguiente aspecto.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns android="http //schemas.android.com/apk/res/android"
xmlns tools="http //schemas.android.com/tools" an-
droid layout_width="match_parent"
android layout_height="match_parent" an-
droid paddingLeft="@dimen/activity_horizontal_margin"
android orientation="vertical"
android paddingRight="@dimen/activity_horizontal_margin"
android paddingTop="@dimen/activity_vertical_margin"
android paddingBottom="@dimen/activity_vertical_margin"
tools context=".MainActivity">

<EditText
    android layout_width="300dp"
    android layout_height="wrap_content"
    android id="@+id/editText1"
    android layout_alignParentTop="true"
    android layout_alignParentLeft="true"
    android layout_alignParentStart="true" />

<EditText
    android layout_width="300dp"
    android layout_height="wrap_content"
    android id="@+id/editText2"
    android layout_below="@+id/editText1"
    android layout_centerHorizontal="true" />
```

```
<Button
    android layout_width="wrap_content"
    android layout_height="wrap_content"
    android text="SUMAR"
    android id="@+id/btnSumar" />

<TextView
    android layout_width="wrap_content"
    android layout_height="wrap_content"
    android textAppearance="?android attr/textAppearanceLarge"
    android text="Resultado es "
    android id="@+id/textViewResultado" />
</LinearLayout>
```

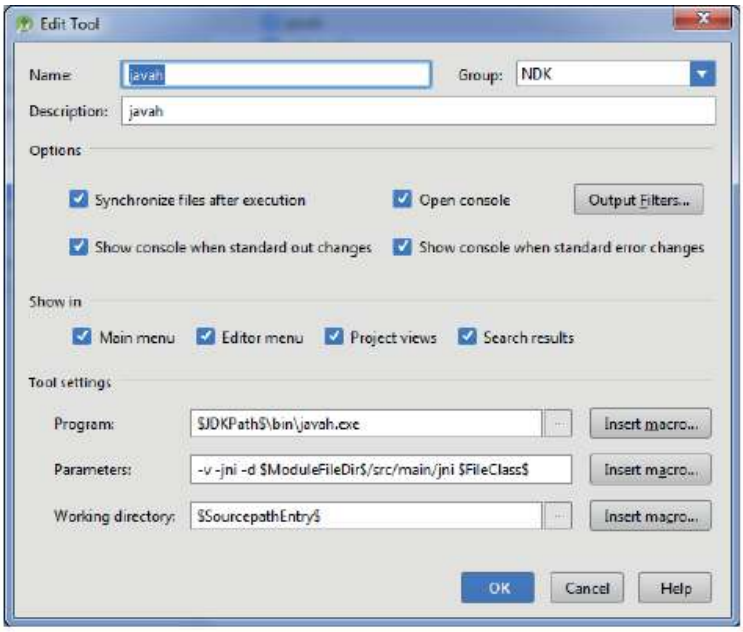
3. Crea el directorio cpp dentro del directorio app/src/main/ de la aplicación.

4. El siguiente paso será modificar el archivo MainActivity.java para introducir nuestro método nativo `calcularSuma()`, la carga de la biblioteca «libsuma.so» e implementar la actividad.

```
public class MainActivity extends AppCompatActivity {

    static {
        System.LoadLibrary("suma");
    }

    EditText etl, et2;

    @Override
    protected void onCreate(Bundle savedlnstanceState) {
        super.onCreate(savedlnstanceState);
        setContentView(R.layout.activity_main);
        Button btnSumar = (Button)findViewByld(R.id.btnSumar);
        btnSumar.setOnClickListener(Oklistener);
    }

    private OnClickListener Oklistener = new OnClickListener(){

        public void onClick(View view){
            etl = (EditText)findViewByld(R.id.editTextl);
            et2 = (EditText)findViewByld(R.id.editText2);
            int aux1 = lnteger.vaLueOf(etl.getText().toString());
            int aux2 = lnteger.vaLueOf(et2.getText().toString());
            int resultado = calcularSuma(aux1, aux2);
            final TextView tvResultado = (Tex-
tView)findViewByld(R.id.textViewResultado);
            tvResultado.setText(""+resultado);
        }
    };

    public native int calcularSuma(int aux1, int aux2);

}
```

5. Vamos a preparar la herramienta javah para crear las caberas de forma automática desde Android Studio a partir de los métodos Java. Para ello iremos a File ➔ Settings ➔ Tools ➔External Tools. Pusaremos sobre el +.

6. Se abrirá una ventana e introduciremos la siguiente información. Name: javah, Group: NDK, Description: javah, habilitaremos todas las opciones y en Tool settings indicaremos donde está la herramienta javah.exe en el caso de Windows (javah en el caso Linux/Unix), en Parameters: -v -jni -d $ModuleFileDir$/src/main/cpp $FileClass$ y en el Working directory: $SourcepathEntry$. Quedando como se muestra en la siguiente figura.



7. Para ejecutar esta herramienta nos situaremos sobre la carpet jni pulsaremos botón derecho ➔ NDK ➔ javah

8. Una vez creado el fichero .h en el directorio cpp. El siguiente paso será implementar el método nativo en código C. Para ello crearemos un fichero denominado com_sumandk_MainActivity.c. La implementación de la función nativa debe tener el mismo prototipo que en la cabecera. Un método nativo siempre tiene al menos dos parámetros `JNIEnv` y `jobject`. El parámetro `env` apunta a una tabla de punteros a funciones, que son las funciones que usaremos para acceder a los datos Java de JNI. El segundo argumento varía dependiendo de si es un método de instancia o un método de clase (estático). Si es un método de instancia se trata de un `jobject` que actúa como un puntero `this` al objeto Java. Si es un método de clase, se trata de una referencia `jclass` a un objeto que representa la clase en la cual están definidos los métodos estáticos.

```
#include "com_sumandk_MainActivity.h"

jint Java_com_sumandk_MainActivity_calcularSuma (JNIEnv * env, jobject
thiz, jint aux1, jint aux2){
    jint total= (aux1+aux2);
```

```
    return total;
}
```

9. Como nuestras fuentes nativas aún no tienen una secuencia de comandos de compilación de CMake, debemos crear una e incluir los comandos correspondientes de CMake.

    - Abrimos el subpanel Project del lado izquierdo de IDE y seleccionamos la vista Project del menú desplegable.
    - Hacemos clic con el botón secundario en el directorio raíz de nuestro módulo y seleccionamos New ➔File.
    - Ingresamos "CMakeLists.txt" como nombre de archivo y haz clic en OK.

10. Abrimos el fichero CMakeLists.txt e introduciremos los comandos cmake_minimum_required() y add_library(). Que son los comandos mínimos necesarios para que nuestra aplicación nativa funcione.

```
# Sets the minimum version of CMake required to build your native library.
# This ensures that a certain set of CMake features is available to
# your build.

cmake_minimum_required(VERSlON 3.4.1)

# Specifies a library name, specifies whether the library is STATlC or
# SHARED, and provides relative paths to the source code. You can
# define multiple libraries by adding multiple add.library() commands,
# and CMake builds them for you. When you build your app, Gradle
# automatically packages shared libraries with your APK.

add_library( # Specifies the name of the library.
             suma

             # Sets the library as a shared library.
             SHARED

             # Provides a relative path to your source file(s).
             src/main/cpp/com_sumandk_MainActivity.c )
```

11. El penúltimo paso será vincular el Gradle con nuestra biblioteca nativa. Para ello:

    - Abrimos el subpanel Project del lado izquierdo de IDE y seleccionamos la vista de Android.
    - Hacemos clic con el botón secundario en el módulo que desees vincular con nuestra biblioteca nativa (por ejemplo, el módulo de app) y selecciona Link C++ Project with Gradle en el menú.
    - Seleccionamos CMake, en el campo junto a Project Path debemos indicar el archivo de secuencia de comandos CMakeLists.txt que acabamos de crear.

12. Finalmente si queremos limitar la compilación a un tipo de ABI, por ejemplo a las arquitecturas de procesadores x86. Para realizar esto debemos de abrir el archivo build.gradle e introducir la siguiente información dentro del defaultConfig{}. Hay que tener en cuenta que si se limita la compilación a un tipo de procesador que no es el que dispone nuestro terminal físico o nuestro emulador, la aplicación no funcionará.

```
ndk{
    abiFilters("x86")
}
```

13. Por último, vamos a lanzar nuestra aplicación. El resultado deberá ser algo similar a la siguiente captura.

Preguntas de repaso: Funcionamiento y estructura de Android NDK

6.4. Interfaz entre JAVA y C/C++ (JNI)

JNI es un mecanismo que nos permite ejecutar código nativo desde Java y viceversa. El código nativo son funciones compiladas en código máquina del procesador para un sistema operativo donde se está ejecutando la máquina virtual. Las bibliotecas nativas se suelen escribir en C o C++ y se compilan dando lugar a códigos binarios que pueden ejecutar directamente el procesador.

JNI tiene una interfaz bidireccional que permite a las aplicaciones Java llamar a código nativo y viceversa, donde se permite llamar a funciones implementadas en código nativo desde Java y también se permite incrustar una máquina virtual en una aplicación nativa.

6.4.1. Bibliotecas de enlace estático y dinámico

Las bibliotecas de enlace estático son ficheros destinados a almacenar funciones, clases y variables globales y tradicionalmente se crean a partir de varios ficheros de código objeto .o (UNIX) o .obj (Windows). En UNIX las bibliotecas de enlace estático suelen tener la extensión .a y en Windows la extensión .lib.

Las bibliotecas de enlace dinámico son ficheros cuyas funciones no se incrustan el ejecutable durante la fase de enlazado, sino que se hace en tiempo de ejecución. El programa busca el fichero, carga su contenido en memoria y enlaza su contenido según va siendo necesario; es decir, según vamos llamando a las funciones. Esto tiene la ventaja de que varios programas pueden compartir las mismas bibliotecas, lo cual reduce el consumo de recursos en los dispositivos. La extensión de estas bibliotecas es .so para sistemas operativos UNIX y .dll para Windows.

6.4.2. Tipos fundamentales, referencias y arrays

En Java existen principalmente dos tipos de datos:

- Tipos fundamentales, como: `int`, `float` o `double`. Su correspondencia con tipos C es directa, ya que en el fichero jni.h encontramos definiciones de tipos C equivalentes.
- Referencias. Apuntan a arrays y objetos. JNI pasa estos datos a los métodos nativos como punteros a la posición de memoria donde se almacenan. Desde JNI no se informa de la estructura interna de estos datos. Por lo tanto, el programador ha de ser informado de cuál es esta estructura para poder manipular los objetos o arrays.

En la siguiente tabla resumen se exponen los tipos de datos que podemos utilizar en la programación nativa mediante la interfaz JNI:

tipo Java	tipo nativo	tipo array nativo	código tipo	código tipo array
boolean	jboolean	jbooleanArray	Z	[Z
Byte	jbyte	jbyteArray	B	[B
Char	jchar	jcharArray	C	[C
Doublé	jdouble	jdoubleArray	D	[D
Float	jfloat	jfloatArray	F	[F
Int	Jint	jintArray	I	[I
Long	jlong	jlongArray	J	[J
Short	jshort	jshortArray	S	[S
Object	jobject	jobjectArray	L	[L
String	jstring	No disponible	L	[L
Class	jclass	No disponible	L	[L
Throwable	jthrowable	No disponible	L	[L
Void	Void	No disponible	V	No disponible

Tabla 7. Tipos de datos en la interfaz JNI.

String es una clase que está representada por el tipo C `jstring`. Para acceder al contenido de este objeto existen funciones que convierten un String Java en cadenas C. Estas funciones nos permiten convertir tanto a Unicode como a UTF-8. Para obtener el texto UTF-8 correspondiente a un String Java tenemos la función:

```
const jbyte* GetStringUTFChars(JNlEnv* env, jstring string, jboolean*
                                                              isCopy);
```

Que nos devuelve un puntero a un array de caracteres UTF-8 del String. Luego para llamar a esta función haríamos:

```
const jbyte* str = (*env)->GetStringUTFChars(env,text,NULL);
if (str==NULL)
retun NULL;
```

Como veremos más tarde, el lanzamiento de excepciones en JNI es distinto al lanzamiento de excepciones en Java. Cuando se lanza una excepción en JNI, no se retrocede por la pila de llamadas hasta encontrar el `catch`, sino que la excepción queda pendiente, y en cuanto salimos del método nativo es cuando se lanza la excepción. El buffer obtenido por esta llamada no se libera hasta que lo liberamos explícitamente usando la función:

```
void ReleaseStringUTFChars(JNlEnv* env, jstring string, const char*
                                                          utf_buffer);
```

Además de las funciones `GetStringUTFChars()` y `ReleaseStringUTFChars()` tenemos las funciones:

```
const jchar* GetStringChars(JNlEnv* env, jstring string, jboolean*
                                                              isCopy);
void RealeaseStringChars(JNlEnv* env, jstring string, const jchar*
                                                         chars_buffer);
```

También podemos crear nuevas instancias de un `java.lang.String` desde un método nativo usando las funciones JNI:

```
jstring NewStringUTF(JNlEnv* env,const char* bytes);
```

Esta función recibe un array de caracteres UTF-8 y crea un objeto `jstring`:

```
jstring NewString(JNlEnv* env, const jchar* ubuffer, jsize length);
```

Esta otra función recibe un array de caracteres Unicode y crea un `jstring`. Ambas funciones, si fallan, producen una excepción `OutOfMemoryError`, en cuyo caso además retornan `NULL`, con lo que siempre hay que comprobar el retorno de la función.

JNI permite trabajar con dos tipos de arrays:

- Arrays de tipos de datos fundamentales.
- Arrays de referencias.

Para acceder a arrays de tipos fundamentales existen funciones JNI que nos devuelven el array en una variable de tipo jarray o derivada. Una vez tengamos una variable nativa de tipo jarray o derivada podemos obtener la longitud del array con:

```
jsize GetArrayLength(JNlEnv* env, jarray array);
```

Después podemos acceder a los datos del array usando funciones J NI. Para cada tipo fundamental existe su correspondiente función J NI. A continuación incluimos un ejemplo de la llamada, donde habría que sustituir `type` por el tipo de dato correspondiente, ya sea `boolean`, `int`, `double`, etc.

```
jtype* GetTypeArrayElements(JNlEnv* env, jtypeArray array, jtype*
                                                          isCopy);
```

Donde `isCopy` es un parámetro de salida que nos dice si el puntero devuelto es el array original, o si se ha hecho una copia de este. Al igual que pasaba con String, las funciones procuran enviar el array original si es posible, con lo que el código nativo no debe bloquearse o llamar a otras funciones J NI entre las llamadas a estas funciones. Por último; también existen funciones para crear arrays J ava desde el código nativo.

```
jtypeArray NewTypeArray(JNlEnv* env, jsize length);
```

6.4.3. Desarrollo paso a paso de un programa mediante J NI (I)

En este punto vamos a desarrollar un programa con código nativo desde cero utilizando la interfaz J NI.

Ejercicio: Desarrollo de la aplicación nativa HolaMundoNDKmediante J NI (I y ndk-build).

1. Crea un proyecto con con las siguientes definiciones:

   Application name HolaMundoNDK

   Project name HolaMundoNDK

   Package name com.holamundondk

   Activity name HolaMundoNDK

   Layout name activity_hola_mundo_ndk

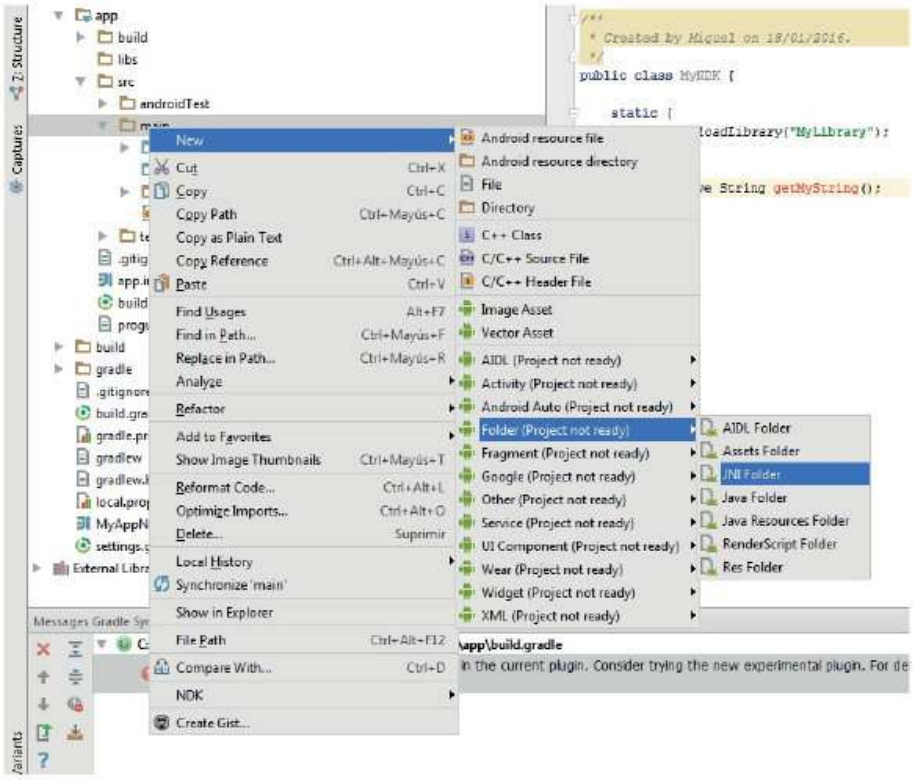
   Resto de parámetros por defecto.

2. Modifica el fichero layout activity hola mundo ndk.xml para que posea el siguiente aspecto. Modifica el string `hello_world` para que nuestra aplicación muestre el mensaje «`Hola Mundo NDK!!`»:

```
<RelativeLayout xmlns android="http //schemas.android.com/apk/res/android"
    xmlns tools="http //schemas.android.com/tools"
    android layout_width="match_parent"
    android layout_height="match_parent"
    android paddingBottom="@dimen/activity_vertical_margin"
    android paddingLeft="@dimen/activity_horizontal_margin"
    android paddingRight="@dimen/activity_horizontal_margin"
    android paddingTop="@dimen/activity_vertical_margin"
    tools context=".HolaMundoNDK" >
```

```
    <TextView
        android layout_width="wrap_content"
        android layout_height="wrap_content"
        android text="@string/hello_world" />
</RelativeLayout>
```

3. La siguiente tarea será crear la carpeta jni. Seleccionamos nuestro proyecto, pulsamos el botón derecho, New/Folder e indicamos el nombre jni.



6.4.3.1. Declaración del método nativo y creación del archivo Android.mk

Los métodos nativos llevan el modificador `native` y están sin implementar, ya que su implementación estará en una biblioteca nativa.

El método para realizar esta declaración es el siguiente:

```
static {
       System.LoadLibrary("biblioteca");
    }
public native String nombreMetodo();
```

Como podemos observar, primero recibe el nombre de una biblioteca de enlace dinámico y la carga. La biblioteca debe cargarse antes de llamar a cualquier método nativo.

Ejercicio: Desarrollo de la aplicación nativa HolaMundoNDK mediante JNI (I) - continuación.

1. El siguiente paso será modificar el archivo HolaMundoNDK.java para introducir nuestro método nativo `dameDatos()` y la carga de la biblioteca «libholamundondk.so». Para ello modificaremos el código y el resultado debe quedar así:

```
public class HolaMundoNDK extends Activity {
    static {
       System.LoadLibrary ("holamundondk");
    }
    public native String dameDatos();

    @Override protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_hola_mundo_ndk);
       setTitle(dameDatos());
    }
}
```

2. Dentro del directorio jni debemos crear el archivo Android.mk que contendrá los siguientes parámetros:

```
LOCAL_PATH = $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE = holamundondk
LOCAL_SRC_FILES = com_holamundondk_HolaMundoNDK.c
include $(BUILD_SHARED_LIBRARY)
```

6.4.3.2. Creación del fichero de cabecera nativo

Cuando trabajamos en C o C++ almacenaremos el código fuente en ficheros .c o .cpp. Pero además también se requieren ficheros .h donde se indican las cabeceras de los métodos implementados. Para ahorrarnos trabajo podemos utilizar la herramienta `javah` para generar de forma automática estos ficheros. A partir de ficheros Java anteriores esta herramienta generará los .h que serán utilizados en nuestro código nativo.

Ejercicio: Desarrollo de la aplicación nativa HolaMundoNDK mediante JNI (I) –continuación (Android Studio).

1. Vamos a preparar la herramienta javah para crear las caberas de forma automática desde Android Studio a partir de los métodos Java. Para ello iremos a File ➔ Settings ➔ Tools ➔ External Tools. Pusaremos sobre el +.

2. Se abrirá una ventana e introduciremos la siguiente información. Name: javah, Group: NDK, Description: javah, habilitaremos todas las opciones y en Tool settings indicaremos donde está la herramienta javah.exe en el caso de Windows (javah en el caso Linux/Unix), en Parameters: -v -jni -d $ModuleFileDir$/src/main/jni $FileClass$ y en el Working directory: $SourcepathEntry$. Quedando como se muestra en la siguiente figura.

3. Para ejecutar esta herramienta nos situaremos sobre la clase de la actividad principal, en nuestro caso HolaMundoNDK, en el árbol de directorios de la izquierda de nuestro IDE y pulsaremos botón derecho ➔ NDK ➔ javah

6.4.3.3. Implementación del método nativo

La implementación de la función nativa debe tener el mismo prototipo que en la cabecera. Un método nativo siempre tiene al menos dos parámetros `JNIEnv` y `jobject`. El parámetro `env` apunta a una tabla de punteros a funciones, que son las funciones que usaremos para acceder a los datos Java de JNI. El segundo argumento varía dependiendo de si es un método de instancia o un método de clase (estático). Si es un método de instancia se trata de un `jobject` que actúa como un puntero `this` al objeto Java. Si es un método de clase, se trata de una referencia `jclass` a un objeto que representa la clase en la cual están definidos los métodos estáticos.

Ejercicio: Desarrollo de la aplicación nativa HolaMundoNDK mediante JNI (I) – continuación.

1. Lo primero que debemos hacer es crear un archivo nuevo en el directorio jni de nuestro proyecto ayudándonos del fichero de cabeceras creado en el punto anterior. El nombre del fichero en C de este ejemplo será com holamundondk HolaMundoNDK.c.
2. El contenido será el siguiente:

```
#include "com_holamundondk_HolaMundoNDK.h"
JNIEXPORT jstring Java_com_holamundondk_HolaMundoNDK_dameDatos (JNIEnv *
env, jobject this) {
```

```
    return (*env)->NewStringUTF(env,"App nativa");
}
```

Donde llamamos a la biblioteca creada en el paso anterior y creamos un string App nativa en C, que será utilizado en nuestro proyecto al llamar al método nativo `dameDatos()`.

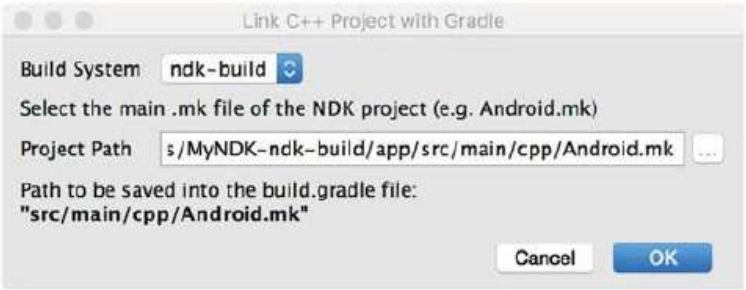
6.4.3.4. Compilación del fichero nativo

El siguiente paso es compilar todo lo realizado hasta ahora con la herramienta de construcción ndk-build.

Ejercicio: Desarrollo de la aplicación nativa HolaMundoNDK mediante J NI (I) –continuación (Android Studio).

Los archivos build.gradle indican a Gradle la manera de compilar nuestra app nativa, ya sea con CMake o ndk-build. Para el caso de ndk-build, se puede establecer que Gradle se vincule con nuestra biblioteca nativa haciendo usando la IU de Android Studio, para ello:

1. Abrimos el subpanel Project del lado izquierdo de IDE y seleccionamos la vista de Android.
2. Hacemos clic con el botón secundario en el módulo que desees vincular con nuestra biblioteca nativa (por ejemplo, el módulo de app) y selecciona Link C++ Project with Gradle en el menú.
3. Seleccionamos ndk-build, y utilizamos el campo junto a Project Path para especificar el archivo Android.mk del proyecto ndk-build externo. Android Studio también incluye el archivo Application.mk si se encuentra en el mismo directorio que tu archivo Android.mk.



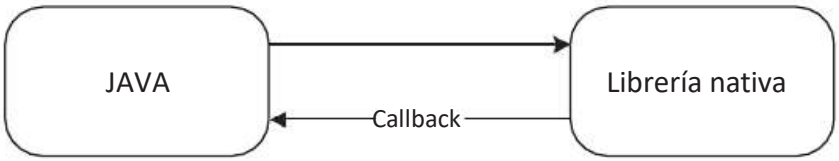
6.4.3.5. Ejecución de la aplicación

Por último, vamos a lanzar nuestra aplicación. Para ello simplemente:

1. Accedemos al menú Run ➔ Run.
2. Si no sale un cuadro de diálogo, le indicamos que queremos una aplicación Android y automáticamente se creará el .apk y este será lanzado en el emulador y/o dispositivo móvil. La apariencia será la siguiente:

6.4.4. Acceso a métodos Java desde código nativo (JNI callback)

Además de poder acceder a los atributos de un objeto Java desde un método nativo, también podemos acceder desde un método nativo a sus métodos Java. A este acceso muchas veces se le llama acceso callback, porque primero Java ejecutó un método nativo y luego el método nativo vuelve a ejecutar un método Java.



Las llamadas callback se hacen de forma distinta dependiendo del tipo del método Java: métodos de instancia, métodos de clase y constructores.

6.4.4.1. Métodos de instancia

Para ejecutar un método de instancia de un objeto tenemos que hacer dos cosas:

1. Obtener el method ID del método usando:

```
jmethodID GetMethodID(JNIEnv* env, jclass class, const char* name,
                                                  const char* signature);
```

2. Ejecutar el método usando una de las siguientes funciones:

```
void CallVoidMethod(JNIEnv* env, jobject object, jmethodID methodID, ...);
jboolean CallBooleanMethod(JNIEnv* env, jobject object,
                                                  jmethodID methodID, ...);
jbyte CallByteMethod(JNIEnv* env, jobject object, jmethodID methodID,
                                                                    ...);
jshort CallShortMethod(JNIEnv* env, jobject object, jmethodID methodID,
                                                                    ...);
jchar CallCharMethod(JNIEnv* env, jobject object, jmethodID methodID,
                                                                    ...);
jint CallIntMethod(JNIEnv* env, jobject object, jmethodID methodID,
                                                                    ...);
jlong CallLongMethod(JNIEnv* env, jobject object, jmethodID methodID,
                                                                    ...);
jfloat CallFloatMethod(JNIEnv* env, jobject object, jmethodID methodID,
                                                                    ...);
jdouble CallDoubleMethod(JNIEnv* env, jobject object, jmethodID methodID,
                                                                    ...);
jobject CallObjectMethod(JNIEnv* env, jobject object, jmethodID methodID,
                                                                    ...);
```

Debemos usar una función u otra según el tipo de retorno que tenga el método. Los parámetros se pasan en la lista de parámetros variables que tiene al

final la función (_). `CallObjectMethod()` se usa tanto para los métodos que devuelven objetos, como para los que devuelven arrays; los métodos que devuelven tipos fundamentales debemos ejecutarlos usando la función que corresponda de las que se enumeran arriba.

6.4.4.2. Métodos de clase

El proceso de ejecutar métodos de clase es similar al de ejecutar métodos de instancia, solo que ahora usamos otras funciones. En concreto los pasos son:

1. Para obtener el method ID usamos:

```
jmethodlD GetStaticMethodlD(JNlEnv* env, jclass class, const char* name,
                                                    const char* signature);
```

2. Para ejecutar los métodos usamos las funciones:

```
void CallStaticVoidMethod(JNlEnv* env, jclass class, jmethodlD methodlD,
                                                                    ...);
jbyte CallStaticByteMethod(JNlEnv* env, jclass class, jmethodlD  methodlD,
                                                                    ...);
jshort CallStaticShortMethod(JNlEnv* env, jclass class, jmethodlD
                                                        methodlD, ...);
jchar CallStaticCharMethod(JNlEnv* env, jclass class, jmethodlD methodlD,
                                                                    ...);
jint CallStaticlntMethod(JNlEnv* env, jclass class, jmethodlD methodlD,
                                                                    ...);
jlong CallStaticLongMethod(JNlEnv* env, jclass class, jmethodlD  methodlD,
                                                                    ...);
jfloat CallStaticFloatMethod(JNlEnv* env, jclass class, methodlD  methodlD,
                                                                    ...);
jdouble CallStaticDoubleMethod(JNlEnv* env, jclass class,  methodlD
                                                       methodlD, ...);
jobject CallStaticObjectMethod(JNlEnv* env, jclass class, methodlD
                                                        methodlD, ...);
```

Estas son idénticas a las de ejecutar métodos de instancia, solo que ahora reciben como parámetro un jclass en vez de un jobject.

6.4.4.3. Invocar constructores

Para llamar a un constructor se llama igual que cualquier otro método de instancia, solo que al ir a acceder al constructor usamos `<init>` como nombre del método. Una vez tengamos el `methodlD` podemos pasárselo a la función:

```
jobject NewObject(JNlEnv, jclass class, jmethodlD constructorlD,...)
```

Ejercicio: Desarrollo de la aplicación nativa HolaMundoNDK mediante J NI (II).

A partir del ejemplo anterior vamos a incluir unos botones para hacer una llamada desde el código J AVA a un método nativo, y viceversa; es decir, desde el código nativo llamar a un método desarrollado en J ava.

1. Lo primero será modificar el layout que teníamos anteriormente. Para ello modificaremos el fichero activity hola mundo ndk.xml introduciendo:

```
    <Button
        android id="@+id/buttonl"
        android layout_width="fiLL_parent"
        android layout_height="wrap_content"
        android layout_alignLeft="@+id/textViewl"
        android layout_below="@+id/textViewl"
        android layout_marginTop="BBdp"
        android  onClick="buttonl"
        android text="@string/buttonl" />
    <Button
        android id="@+id/buttone"
        android layout_width="fiLL_parent"
        android layout_height="wrap_content"
        android layout_alignBottom="@+id/buttonl"
        android layout_alignLeft="@+id/buttonl"
        android layout_marginBottom="42dp"
        android onClick="buttone"
        android text="@string/buttone" />
    <TextView
        android id="@+id/output"
        android layout_width="fiLL_parent"
        android layout_height="wrap_content"
        android layout_above="@+id/buttone"
        android layout_alignLeft="@+id/buttone"
        android gravity="center" />
```

2. El siguiente paso es modificar el archivo HolaMundoNDK.java (ver código siguiente). En él hemos incluido dos métodos nativos: `funcion1()` y `funcion2()`. También existe un método denominado `funcion3Callback()` que será llamado desde el código nativo, como veremos más adelante. Como podemos observar son métodos sencillos donde se les pasa una cadena string y devuelven otra cadena string:

```
public class MainActivity extends AppCompatActivity {

    private TextView salida;
    public native String dameDatos();
    public native String funcion1(String message);
```

```
    public native void funcion2();

    static {
        System.LoadLibrary ("holamundondk");
    }

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setTitle(dameDatos());
        salida = (TextView)super.findViewByld(R.id.output);
    }

    public void button0(View v){
        salida.setText(funcion1("testString"));
    }
    public void button1(View v){
        funcion2();
    }
    public void funcion3Callback(){
        String message = "funcion3Callback llamada por la funcion2 na-
tiva";
        salida.setText(message);
    }
    public void run(){}
}
```

3. El siguiente paso es modificar el archivo Android.mk. En él vamos a introducir las `LOCAL_LDLlBS = -llog` y `LOCAL_CFLAGS = -Werror`. Esto implica que vamos a cargar las bibliotecas de log y vamos a activar los flags de error en C. De esta forma podremos extraer información a través del log y detectar errores en la ejecución del código nativo:

```
LOCAL_PATH = $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE = holamundondk
LOCAL_SRC_FlLES = com_holamundondk_HolaMundoNDK.c
LOCAL_LDLlBS = -llog
LOCAL_CFLAGS = -Werror

include $(BUlLD_SHARED_LlBRARY)
```

4. El siguiente paso es actualizar el fichero de cabecera. Para ello volveremos a ejecutar nuestra herramienta externa javah. Al volver a ejecutar esta herramienta nuestro fichero com holamundondk HolaMundoNDK.h se actualizará y contendrá las dos nuevas declaraciones de los métodos nativos:

```
/*
 * Class      com_holamundondk_HolaMundoNDK
 * Method     funcion1
 * Signature (Ljava/lang/String;)Ljava/lang/String;
 */
```

```
JNlEXPORT jstring JNlCALL Java_com_holamundondk_HolaMundoNDK_funcionl
  (JNlEnv *, jobject, jstring);
/*
 * Class      com_holamundondk_HolaMundoNDK
 * Method     funcion2
 * Signature ()V
 */
JNlEXPORT void JNlCALL Java_com_holamundondk_HolaMundoNDK_funcion2
  (JNlEnv *, jobject);
```

5. Ahora vamos a modificar el fichero com holamundondk HolaMundoNDK.c para implementar los dos nuevos métodos. Primero hemos definido los parámetros del log, como son la etiqueta (`LOG_TAG`), el log de información (`LOGl`) y el log de error (`LOGE`).

   En la `funcion1()` hemos decido pasarle una cadena como entrada para tener la oportunidad de mostrar cómo el desarrollador debe proceder cuando tiene que manejar los tipos de variables que no son los primarios. En este caso, tenemos que convertir explícitamente la cadena de `jstring`a un elemento nativo `char *`. Si no lo hacemos, a veces, el error no aparece a simple vista, pero mientras la aplicación se ejecuta este error puede llevarnos a comportamientos no previstos, y la depuración se convertirá en una tarea muy difícil. Por lo tanto, hay que prestar especial atención en los datos. Hemos declarado una cadena nativa, lo que permitirá guardar el resultado de la función mediante `GetStringUTFChars` que obtiene los caracteres de la cadena representados en el formato Unicode. El tercer parámetro indica si queremos una copia o el puntero a la cadena real java; en este último caso, el desarrollador debe prestar atención a no modificar el contenido de la cadena devuelta. Este parámetro es típico tenerlo a`NULL`.

   El método `funcion2()` representa un ejemplo muy simple de una devolución de llamada (callback) J NI. Normalmente, cada vez que tenemos que devolver la llamada a un método J ava implementado en la clase que hizo la llamada, vamos a tener que realizar los siguientes tres pasos:

   a. Obtener la clase (`jclass`), empezando desde el objeto (`jobject`) que hizo la llamada, a través de la `GetObjectClass`.

   b. Obtener el identificador del método, a través de `GetMethodlD`, que lleva a cabo una búsqueda para el método en la clase dada. La búsqueda se basa en el nombre y el descriptor del tipo de método. Si el método no existe, `GetMethodlD` devuelve`NULL`.

   c. Llamar al método, en el objeto de llamadas, pasando el `methodlD`.

   El código dentro de la `funcion2()` es fácil de entender, a pesar del detalle que vamos a analizar en la función `GetMethodlD`, que toma como parámetros de entrada:

   a. `JNlEnv *`, el puntero al entorno J ava.

   b. `jobject`, el objeto que llama, que pasamos a la función nativa como parámetro de entrada.

   c. `const char *`, la cadena que identifica el nombre del método para devolver la llamada.

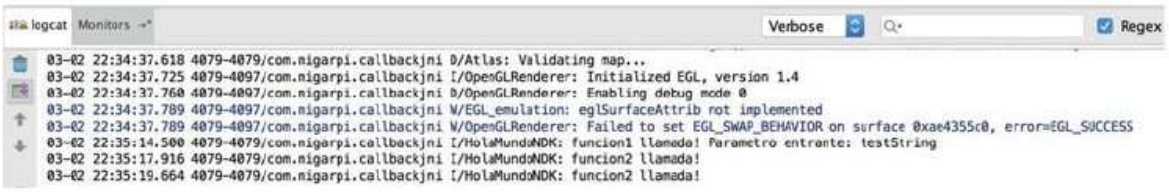
d. `const char *`, la firma del método, llamado descriptor de método: a pesar de que pueda parecer extraño, es muy fácil de entender. La primera parte, entre paréntesis, representa los parámetros de entrada, y sus tipos. El último identificador, después del paréntesis, representa el tipo de cambio. En nuestro caso, «`()V`» significa `void funcion3Callback()` Si nuestra función hubiera sido, `float funcion3Callback (int i)`, el descriptor sería «`(I)F`». Y si tuviéramos `float funcion3Callback (int i, float f, int j)` escribiríamos «`(IFI)F`».

6. A través de esa función llamamos a un método Java desde el código nativo:

```
#include "com_holamundondk_HolaMundoNDK.h"
#include <android/log.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define  LOG_TAG    "HolaMundoNDK"
#define LOGI(...)_android_log_print(ANDROID_LOG_INFO,LOG_TAG,_VA_ARGS_ )
#define LOGE(...)_android_log_print(ANDROID_LOG_ERROR,LOG_TAG,_VA_ARGS_ )

JNIEXPORT jstring Java_com_holamundondk_HolaMundoNDK_dameDatos (JNIEnv *
                                                    env, jobject thiz) {
    return (*env)->NewStringUTF(env,"App nativa");
}

jstring Java_com_holamundondk_HolaMundoNDK_funcion1(JNIEnv* env,
                                        jobject thiz, jstring message) {
    const char *nativeString = (*env)->GetStringUTFChars(env, message, 0);
    LOGI("funcion1 llamada! Parametro entrante %s", nativeString);
    (*env)->ReleaseStringUTFChars(env, message, nativeString);
    return (*env)->NewStringUTF(env, "Llamada nativa JNI realizada!");
}

void Java_com_holamundondk_HolaMundoNDK_funcion2(JNIEnv* env,
                                                    jobject thiz) {
    LOGI("funcion2 llamada!");
    jclass clazz = (*env)->GetObjectClass(env, thiz);
    if (!clazz) {
        LOGE("callback_handler FALLO object Class");
        goto failure;
    }
    jmethodID method = (*env)->GetMethodID(env, clazz, "funcion3Callback",
                                                                    "()V");
    if (!method) {
        LOGE("callback_hand ler FALLO metodo ID");
        goto failure;
    }
    (*env)->CallVoidMethod(env, thiz, method);

    failure return;
}
```

7. Ahora actualizaremos la conexión con los archivos nativos. Para ello, pulsamos sobre el menú Build ➔ Refresh Linked C++ Projects.
8. Si no existe ningún error ya podremos realizar el siguiente paso: la creación del .apk. En caso contrario deberemos analizar los errores y subsanarlos hasta que no exista ninguno de ellos.
9. Accedemos al menú Run/Run.
10. Si no sale un cuadro de diálogo, le indicamos que queremos una aplicación Android y automáticamente se creará el .apk y este será lanzado en el emulador y/o dispositivo móvil. La aplicación tendrá la siguiente apariencia:

11. Observa el Android Monitor. Podemos ver que cada vez que pulsamos sobre un botón, en nuestra aplicación aparece una entrada en el log, lo cual nos puede ayudar a la hora de comprobar el correcto funcionamiento de nuestra aplicación.



6.4.5. Excepciones

Cuando se produce una excepción (bien sea en un método JNI o bien en un método callback de Java) no se interrumpe el flujo normal de ejecución del programa (como pasa en Java), sino que en el hilo se activa un flag indicando que se ha lanzado la excepción. Existe un único flag por cada hilo y este está almacenado en la estructura env, con lo que el hecho de que se produzca una excepción en un hilo no afecta a los demás hilos. Podemos comprobar si en nuestro hilo se ha producido una excepción llamando a:

```
jboolean ExceptionCheck(JNIEnv* env);
```

O alternativamente podemos usar:

```
jthrowable ExceptionOccurred(JNIEnv* env);
```

Esta última función devuelve una referencia a la excepción ocurrida, o NULL si no ha habido excepción. Aunque parezca tedioso es necesario que una función nativa bien construida compruebe si se ha producido una excepción cada vez que

llama a una función JNI que puede producir una excepción o a un método callback de Java.

```
jfieldlD fieldlD = (*env)->GetFieldlD(env,clase,"atributo","l");
if (fieldlD==NULL) {
    // Se ha producido una excepcion y la tratamos
}
```

En cualquier caso, si nuestro método nativo no se molesta en comprobar las posibles excepciones de cada llamada pueden producirse resultados inesperados al seguir llamando a más funciones de JNI.

En realidad, cuando se produce una excepción hay una pequeña lista de funciones JNI que son las únicas que se pueden ejecutar de forma segura antes de tratar la excepción que es la siguiente. Estas funciones nos permiten tratar la excepción o liberar recursos antes de retornar a Java:

DeleteLocalRef()	PushLocalFrame()
DeleteGlobalRef()	PopLocalFrame()
ExceptionOccured()	ReleaseStringChars()
ExceptionDescribe()	ReleaseStringUTFChars()
ExceptionClear()	ReleaseStringCritical()
ExceptionCheck()	Release<Primitive>ArrayElements()
MonitorExit()	ReleasePrimitiveArrayCritical()

Una función nativa puede lanzar una excepción usando la función JNI:

```
jint ThrowNew(JNlEnv* env, jclass class, const char* message);
```

Esto activa el flag de la excepción y deja la excepción pendiente hasta que la tratemos. La función retorna 0 si la excepción se lanza o un valor distinto de 0 si no se pudo lanzar la excepción.

Preguntas de repaso: Interfaz entre Java y C/C++ (JNI).

6.5. Rendimiento de aplicaciones con código nativo

En este punto del tema vamos a comprobar el rendimiento de las aplicaciones con código nativo con respecto a las aplicaciones desarrolladas íntegramente con Java.

Para ello vamos a desarrollar un ejemplo que mostrará el valor que ocupa la posición X en la serie de Fibonacii; es decir, si la serie de Fibonacci es 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 … el valor correspondiente a la posición 6 será 8. Esta acción será implementada en dos métodos de programación y veremos cuál es más rápido.

Ejercicio: Desarrollo de una aplicación para comprobar el rendimiento del código nativo con ndk-build.

1. Crea un proyecto con con las siguientes definiciones:

```
Application name FibonacciNDK
Project name FibonacciNDK
Package name com.fibonaccindk
Resto de parámetros por defecto.
Activity name FibonacciNDK
Layout name main
```

2. Modifica el fichero main.xml para que posea el siguiente aspecto:

```
<?xml version="1.e" encoding="utf-B"?>
<LinearLayout xmlns android="http://schemas.android.com/apk/res/android"
    android orientation="verticaL"
    android layout_width="fiLL_parent"
    android layout_height="fiLL_parent"
    android gravity="center_horizontaL">
    <TextView
        android layout_width="fiLL_parent"
        android layout_height="wrap_content"
        android text="DaLvik vs. Nativo"
        android gravity="center"
        android  textSize="4esp"
        android layout_margin="ledp" />
    <LinearLayout
        android layout_width="wrap_content"
        android layout_height="wrap_content">
        <EditText
            android layout_width="wrap_content"
            android layout_height="wrap_content"
            android id="@+id/VaLorEntrante"
            android hint="VaLor"
            android textSize="3esp"
            android inputType="number"
            android layout_margin="ledp">
        </EditText>
        <Button
            android layout_width="wrap_content"
            android layout_height="wrap_content"
            android id="@+id/botonLanzar"
            android text="Lanzar"
            android textSize="3esp"
            android layout_margin="lesp">
```

```
        </Button>
    </LinearLayout>
    <TextView
        android layout_width="wrap_content"
        android layout_height="wrap_content"
        android id="@+id/ResuLtado"
        android text="RESULTADO"
        android textSize="2esp">
    </TextView>
</LinearLayout>
```

3. La siguiente tarea será crear la carpeta jni. Seleccionamos nuestro proyecto, pulsamos el botón derecho, New/Folder e indicamos el nombre jni.

4. Ahora añadimos el código Java en el archivo FibonacciNDK.java. Este fichero debe poseer dos métodos que ejecuten el objetivo del ejercicio tanto de forma recursiva como iterativa. En nuestro caso `fibonacciDalvikR()` y `fibonacciDalvikl()`. Después tenemos la llamada a la biblioteca «libfibonacci.so» y los métodos nativos `fibonacciNativoR()` y `fibonacciNativol()`.

5. A través del método `onClick()` extraeremos los tiempos utilizados tanto en las implementaciones nativas como las virtualizadas en la máquina Dalvik:

```
public class FibonacciNDK extends Activity implements OnClickListener {
  TextView Resultado;
  Button botonLanzar;
  EditText ValorEntrante;

  @Override public void onCreate(Bundle savedlnstanceState) {
    super.onCreate(savedlnstanceState);
    setContentView(R.layout.main);
    ValorEntrante = (EditText) findViewByld(R.id.VaLorEntrante);
    Resultado = (TextView) findViewByld(R.id.ResuLtado);
    botonLanzar = (Button) findViewByld(R.id.botonLanzar);
    botonLanzar.setOnClickListener(this);
  }

  public static long fibonacciDalvikR(long n) {
    if (n <= 0)
        return 0;
    if (n == 1)
        return 1;
    return fibonacciDaLvikR(n - 1) + fibonacciDaLvikR(n - 2);
  }
  public static long fibonacciDalvikl(long n) {
    long previous = -1;
    long result = 1;
    for (long i = 0; i <= n; i++) {
       long sum = result + previous;
       previous = result;
       result = sum;
    }
    return result;
```

```
  }


  static {
    System.LoadLibrary("fibonacci");
  }

  public static native long fibonacciNativoR(int n);

  public static native long fibonacciNativol(int n);

  public void onClick(View view) {
    int input = lnteger.parseInt(ValorEntrante.getText().toString());
    long start1, start2, stop1, stop2;
    long result;
    String out = "";
// Dalvik - Recursivo
    start1 = System.currentTimeMiLLis();
    result = fibonacciDaLvikR(input);
    stop1 = System.currentTimeMiLLis();
    out += String.format("Dalvik recursiva - Valor  %d Tiempo  (%d msec)",
                                                result, stop1 - start1);
// Dalvik - lterativo
    start2 = System.currentTimeMiLLis();
    result = fibonacciDaLvikI(input);
    stop2 = System.currentTimeMiLLis();
    out += String.format("\nDalvik iterativa-Valor  %d Tiempo  (%d msec)",
                                                result, stop2 - start2);
// Nativo - Recursivo
    start1 = System.currentTimeMiLLis();
    result = fibonacciNativoR(input);
    stop1 = System.currentTimeMiLLis();
    out += String.format("\nNativo recursivo-Valor  %d Tiempo  (%d msec)",
                                                result, stop1 - start1);
// Nativo - lterativo
    start2 = System.currentTimeMiLLis();
    result = fibonacciNativoI(input);
    stop2 = System.currentTimeMiLLis();
    out += String.format("\nNativo iterativo-Valor  %d Tiempo  (%d msec)",
                                                result, stop2 - start2);
    Resultado.setText(out);
  }
}
```

6. Dentro del directorio jni debemos crear el archivo Android.mk que contendrá.

```
LOCAL_PATH = $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE = fibonacci
LOCAL_SRC_FlLES = fibonacci.c

include $(BUlLD_SHARED_LlBRARY)
```

7. Crea del fichero com fibonaccidk fibonacciNDK.h. Para ello, nos situaremos sobre el directorio jni y pulsaremos botón secundario ➔ NDK ➔ javah y así ejecutaremos la herramienta javah. La información más importante de este archivo autogenerado son las siguientes llamadas:

```
/*
 * Class      com_fibonaccindk_FibonacciNDK
 * Method     fibonacciNativoR
 * Signature (l)J
 */
JNlEXPORT jlong JNlCALL Java_com_fibonaccindk_FibonacciNDK_fibonacciNativoR
                                                  (JNlEnv *, jclass, jint);

/*
 * Class      com_fibonaccindk_FibonacciNDK
 * Method     fibonacciNativol
 * Signature (l)J
 */
JNlEXPORT jlong JNlCALL Java_com_fibonaccindk_FibonacciNDK_fibonacciNativoI
                                                  (JNlEnv *, jclass, jint);

#ifdef_cplusplus
}
#endif
#endif
```

8. Lo siguiente que debemos hacer es crear un archivo nuevo en el directorio jni de nuestro proyecto. El nombre será fibonacci.c. En él tendremos la implementación de los métodos `fibonacciNativoR()` y `fibonacciNativol()` y sus respectivas llamadas:

```
#include "com_fibonaccindk_FibonacciNDK.h"
jint fibonacciNativoR(jint n) {
  if(n<=0) return 0;
  if(n==1) return 1;
  return fibonacciNativoR(n-1) + fibonacciNativoR(n-2);
}

jint fibonacciNativoI(jint n) {
  jint previous = -1;
  jint result = 1;
  jint i=0;
  jint sum=0;
  for (i = 0; i <= n; i++) {
    sum = result + previous;
    previous = result;
    result = sum;
  }
  return result;
}

JNlEXPORT jlong JNlCALL Java_com_fibonaccindk_FibonacciNDK_fibonacciNativoR
                                      (JNlEnv *env, jclass obj, jint n) {
```

```
    return fibonacciNativoR(n);
}
JNlEXPORT jlong JNlCALL Java_com_fibonaccindk_FibonacciNDK_fibonacciNativoI
                                        (JNlEnv *env, jclass obj, jint n) {
    return fibonacciNativol(n);
}
```

9. El siguiente paso es la vinculación de nuestro código nativo con el sistema de compilación Gradle. Para ello hacemos clic con el botón secundario en el módulo de app y selecciona Link C++ Project with Gradle en el menú.

10. Seleccionamos ndk-build, y utilizamos el campo junto a Project Path para especificar el archivo Android.mk del proyecto ndk-build externo.

11. Si no existe ningún error ya podremos realizar el siguiente paso que es la creación del .apk. En caso contrario deberemos analizar los errores y subsanarlos hasta que no exista ninguno de ellos.

12. Accedemos al menú Run/Run.

13. Si no sale un cuadro de diálogo le indicamos que queremos una aplicación Android y automáticamente se creará el .apk y este será lanzado en el emulador y/o dispositivo móvil. La aplicación tendrá la siguiente apariencia:

Práctica: Número primo en código nativo.

Para realizar otra comprobación del comportamiento del código nativo debes realizar un programa similar al anterior, pero que al introducir un número indique si es primo o no. Este algoritmo ya fue explicado en capítulos anteriores; por tanto la parte algorítmica ya está resuelta.

6.6. Procesado de imagen con código nativo

En este último punto del capítulo vamos a desarrollar una aplicación de procesado de imagen utilizando los conceptos vistos a lo largo de esta unidad.

Ejercicio: Desarrollo de una aplicación de procesado de imagen con código nativo con CMake.

1. Crea un proyecto con con las siguientes definiciones:

```
Application name lmgProcesadoNDK
Project name lmgProcesadoNDK
```

```
Package name com.imgprocesadondk
Resto de parámetros por defecto.
Activity name ImgProcesadoNDK
Layout name main
```

2. Modificar el layout, incluyendo el siguiente código XML:

```
<LinearLayout xmlns android="http //schemas.android.com/apk/res/android"
    android layout_width="fill_parent"
    android layout_height="fill_parent"
    android orientation="vertical" >
    <ImageView
        android  layout_width="wrap_content"
       android  layout_height="wrap_content"
       android scaleType="centerCrop"
       android layout_gravity="center_verticallcenter_horizontal"
       android id="@+id/ivDisplay"/>
    <LinearLayout
        android orientation="horizontal"
        android gravity="center"
        android layout_width="match_parent"
        android layout_height="wrap_content" >
        <Button
            android id="@+id/btnReset"
            style="?android attr/buttonStyleSmall"
            android layout_width="wrap_content"
            android layout_height="wrap_content"
            android text="Original"
            android onClick="onResetImagen" />
        <Button
            android id="@+id/btnConvert"
            style="?android attr/buttonStyleSmall"
            android layout_width="wrap_content"
            android layout_height="wrap_content"
            android text="Grises"
            android onClick="onConvertirGrises" />
    </LinearLayout>
</LinearLayout>
```

3. Crea la carpeta cpp. Para ello, selecciona en nuestro proyecto, pulsa con el botón derecho y selecciona New/Folder.

4. Modificaremos el fichero de la actividad:

```
public class ImgProcesadoNDK extends AppCompatActivity {

    private String tag = "ImgProcesadoNDK";
    private Bitmap bitmapOriginal = null;
    private Bitmap bitmapGrises = null;
    private ImageView ivDisplay = null;

    static {
        System.LoadLibrary("imgprocesadondk");
    }
    public native void convertirGrises(Bitmap bitmapIn, Bitmap bitmapOut);
```

```
    @Override
    public void onCreate(Bundle savedlnstanceState) {
        super.onCreate(savedlnstanceState);
        setContentView(R.layout.activity_img_procesado_ndk);
        Log.i(tag, "lmagen antes de modificar");
        ivDisplay = (lmageView) findViewByld(R.id.ivDispLay);
        BitmapFactory.Options options = new BitmapFactory.Options();
        // Asegurar que La imagen tiene 24 bits de coLor
        options.inPreferredConfig = Bitmap.Config.ARGB_BBBB;
        bitmapOriginal = BitmapFactory.decodeResource(this.getResources(),
                R.drawable.sampLeimage, options);
        if (bitmapOriginal != null)
            ivDisplay.setlmageBitmap(bitmapOriginal);
    }

    public void onResetlmagen(View v) {
        Log.i(tag, "Resetear lmagen");
        ivDisplay.setlmageBitmap(bitmapOriginal);
    }

    public void onConvertirGrises(View v) {
        Log.i(tag, "Conversion a escala de grises");
        bitmapGrises = Bitmap.createBitmap(bitmapOriginal.getWidth(),
                bitmapOriginal.getHeight(), Bitmap.Config.ARGB_BBBB);
        convertirGrises(bitmapOriginal, bitmapGrises);
        ivDisplay.setlmageBitmap(bitmapGrises);
    }
}
```

5. Crea el fichero CMakeLists.txt:

```
# Sets the minimum version of CMake required to build the native
# library. You should either keep the default value or only pass a
# value of 3.4.0 or lower.

cmake_minimum_required(VERSlON 3.4.1)

# Creates and names a library, sets it as either STATlC
# or SHARED, and provides the relative paths to its source code.
# You can define multiple libraries, and CMake builds it for you.
# Gradle automatically packages shared libraries with your APK.

add_library( # Sets the name of the library.
             imgprocesadondk
             # Sets the library as a shared library.
             SHARED
             # Provides a relative path to your source file(s).
             # Associated headers in the same location as their source
             # file are automatically included.
             src/main/cpp/com_imgprocesadondk_lmgProcesadoNDK.c )
# Searches for a specified prebuilt library and stores the path as a
# variable. Because system libraries are included in the search path by
# default, you only need to specify the name of the public NDK library
# you want to add. CMake verifies that the library exists before
# completing its build.
find_library( # Sets the name of the path variable.
```

```
              log-lib
              # Specifies the name of the NDK library that
              # you want CMake to locate.
              log )
find_library( # Sets the name of the path variable.
              jnigraphics-lib
              # Specifies the name of the NDK library that
              # you want CMake to locate.
              jnigraphics )
# Specifies libraries CMake should link to your target library. You
# can link multiple libraries, such as libraries you define in the
# build script, prebuilt third-party libraries, or system libraries.

target_link_libraries( # Specifies the target library.imgprocesadondk
                       # Links the target library to the log library
                       # included in the NDK.
                       ${log-lib} ${jnigraphics-lib})
```

6. Crea el fichero com imgprocesadondk ImgProcesadoNDK.h con la herramienta javah. Para ello modifica los parámetros para incluir la clase que da soporte a Bitmap. Desde el menú de External Tools, selecciona javah y modificala introduciendo como parámetros: -v -jni -d $ModuleFileDir$/src/main/cpp -classpath $ModuleSdkPath$/platforms/android-23/android.jar: $FileClass$. Si estamos trabajando con Windows los ":" deberán cambiar por ";" y las "/" por "\".

7. Implementa el siguiente código nativo en el fichero imgprocesadondk.c:

```
#include "com_imgprocesadondk_lmgProcesadoNDK.h"
#include <android/log.h>
#include <android/bitmap.h>
#define LOG_TAG "libimgprocesadondk"
#define LOGl(...)_android_log_print(ANDROlD_LOG_lNFO,LOG_TAG,_VA_ARGS_ )
#define LOGE(...)_android_log_print(ANDROlD_LOG_ERROR,LOG_TAG,_VA_ARGS_ )
typedef struct {
    uint8_t red;
    uint8_t green;
    uint8_t blue;
    uint8_t alpha;
} rgba;
/*Conversion a grises por pixeL*/
JNlEXPORT void JNlCALL Java_com_imgprocesadondk_lmgProcesadoNDK_convertirGrises
    (JNlEnv *env, jobject obj, jobject bitmapcolor, jobject bitmapgris) {
    AndroidBitmaplnfo infocolor;
    void *pixelscolor;
    AndroidBitmaplnfo infogris;
    void *pixelsgris;
    int ret;
    int y;
    int x;
    LOGl("convertirGrises");
    if ((ret = AndroidBitmap_getlnfo(env, bitmapcolor, &infocolor)) < 0) {
        LOGE("AndroidBitmap_getlnfo() failed ! error=%d", ret);
        return;
```

```
    }

    if ((ret = AndroidBitmap_getlnfo(env, bitmapgris, &infogris)) < 0) {
        LOGE("AndroidBitmap_getlnfo() failed ! error=%d", ret);
        return;
    }

    LOGl("imagen color ancho %d;alto %d;avance %d;formato %d;flags %d",
         infocolor.width, infocolor.height, infocolor.stride,
         infocolor.format, infocolor.flags);
    if (infocolor.format != ANDROID_BITMAP_FORMAT_RGBA_BBBB) {
        LOGE("Bitmap no es formato RGBA_8888 !");
        return;
    }

    LOGl("imagen color ancho %d;alto %d;avance %d;formato %d;flags %d",
         infogris.width, infogris.height, infogris.stride,
         infogris.format, infogris.flags);
    if (infogris.format != ANDROID_BITMAP_FORMAT_RGBA_BBBB) {
        LOGE("Bitmap no es formato RGBA_8888 !");
        return;
    }

    if ((ret = AndroidBitmap_lockPixels(env, bitmapcolor, &pixelscolor))
        < 0) {
        LOGE("AndroidBitmap_lockPixels() failed ! error=%d", ret);
    }

    if ((ret = AndroidBitmap_lockPixels(env,bitmapgris,&pixelsgris)) <0){
        LOGE("AndroidBitmap_lockPixels() fallo ! error=%d", ret);
    }
    // modificacion pixeLes en eL aLgoritmo de escaLa grises
    for (y = 0; y < infocolor.height; y++) {
        rgba *line = (rgba *) pixelscolor;
        rgba *grisline = (rgba *) pixelsgris;
        for (x = 0; x < infocolor.width; x++) {
            float output = (line[x].red + line[x].green + line[x].blue)/3;
            if (output > 255) output = 255;
            grisline[x].red = grisline[x].green = grisline[x].blue =
                    (uint8_t) output;
            grisline[x].alpha = line[x].alpha;
        }
        pixelscolor = (char *) pixelscolor + infocolor.stride;
        pixelsgris = (char *) pixelsgris + infogris.stride;
    }
    LOGl("unlocking pixels");
    AndroidBitmap_unlockPixels(env, bitmapcolor);
    AndroidBitmap_unlockPixels(env, bitmapgris);
}
```

8. El siguiente paso es la vinculación de nuestro código nativo con el sistema de compilación Gradle. Hacemos clic con el botón secundario en el módulo de app y selecciona Link C++ Project with Gradle en el menú. Seleccionamos CMake, en el campo junto a Project Path debemos indicar el archivo de secuencia de comandos CMakeLists.txt de tu proyecto de CMake externo.

9. Si no existe ningún error, ya podrás realizar el siguiente paso que es la creación del .apk. En caso contrario, deberemos analizar los errores y subsanarlos.

10. Accedemos al menú Run/Run.

11. Si no sale un cuadro de diálogo le indicamos que queremos una aplicación Android y automáticamente se creará el .apk y este será lanzado. La aplicación tendrá la siguiente apariencia:

Práctica: Mi InstagramNDK.

Partiendo del ejercicio anterior, trata de realizar una aplicación que tome una imagen a partir de la cámara del dispositivo móvil (mediante el SDK de Android, es decir, mediante código Java). Modifica el programa realizado anteriormente para que la imagen sea la tomada por la cámara. Añade otro procesado de imagen en código nativo para que la imagen pase a tono sepia. A continuación, se indica el algoritmo que has de utilizar para cada píxel; recordad que debemos mantener la componente alpha de la imagen:

```
outputRed = (inputRed * .393) + (inputGreen *.769) + (inputBlue * .189)
outputGreen = (inputRed * .349) + (inputGreen *.686) + (inputBlue * .168)
outputBlue = (inputRed * .272) + (inputGreen *.534) + (inputBlue * .131)
```

El resultado será:

CAPÍTULO 7.

Ingeniería inversa en Android

Por JESÚS TOMÁS

La ingeniería inversa consiste en descubrir cómo se ha realizado un diseño a partir del producto disponible al público. En el mundo del software podemos matizar esta definición diciendo que consiste en tomar como partida una aplicación y tratar de descubrir cómo funciona u obtener el código que se ha utilizado en su desarrollo.

Las empresas usan la ingeniería inversa para estudiar a la competencia y de esta manera mejorar sus propios productos. Por desgracia, en el mundo del software la mayoría de las veces que se utiliza la ingeniería inversa es para tratar de evitar las comprobaciones de licencia que se incluyen en las aplicaciones comerciales y, de esta manera, poder utilizarlas de forma fraudulenta, sin tener que pagar por ellas. Por supuesto, el objetivo de este capítulo no es este. Vamos a explicar varias técnicas de ingeniería inversa con el fin de que el lector conozca las vulnerabilidades de una aplicación y, así, pueda introducir técnicas que dificulten la ingeniería inversa en sus desarrollos. Existen otros ámbitos donde la ingeniería inversa estaría justificada, como aprender cómo funcionan internamente las cosas o descubrir comportamientos maliciosos de ciertas aplicaciones.

Este capítulo se puede dividir en dos partes: cómo realizar ingeniería inversa en Android y cómo tratar de evitarla.

En la primera parte comenzaremos describiendo el formato APK, utilizado para distribuir aplicaciones Android. Veremos cómo obtener estos ficheros, cómo analizarlos, cuál es su contenido y cómo se firman. Luego aprenderemos a decompilar el código de un APK. Veremos cómo obtener el código ensamblador de la máquina Dalvik y estudiaremos su diferencia con los byte codes de la máquina Java. También veremos que es posible obtener un código en Java similar al de la aplicación original. Esta parte termina describiendo cómo podemos modificar partes

de la aplicación comercial. Veremos cómo modificar recursos binarios, recursos XML y el código.

En la segunda parte trataremos de evitar la ingeniería inversa. Primero aprenderemos a ofuscar el código y, así, evitar que sea fácilmente decompilado. Luego aprenderemos a utilizar el servicio de licencias de Google Play y, así, evitar usos no autorizados de nuestras aplicaciones. Aunque desde nuestro código se verifique la licencia para poder ejecutar la aplicación, veremos como es posible localizar las líneas donde se hace la verificación y eliminarlas. Terminaremos el capítulo enumerando varias contramedidas para tratar de evitarlo.

Objetivos:

- Repasar el concepto de ingeniería inversa y emplearlo en aplicaciones Android.
- Mostrar varios mecanismos para impedir el uso de ingeniería inversa en nuestras aplicaciones.
- Describir cómo se distribuyen las aplicaciones en ficheros APK y cuál es su contenido.
- Comparar la máquina virtual Dalvik con la máquina virtual Java.
- Ilustrar cómo podemos decompilar una aplicación Android, tanto a ensamblador como a Java.
- Mostrar cómo podemos modificar aplicaciones Android, tanto sus recursos binarios, recursos XML como su código.
- Enumerar los pasos a seguir para ofuscar nuestras aplicaciones.
- Describir el uso de licencias de Google Play para impedir usos fraudulentos de nuestras aplicaciones.

7.1. El formato APK

Las aplicaciones Android se distribuyen en ficheros APK. Estos ficheros suelen distribuirse por cualquier medio, aunque la opción más habitual es obtenerlos a través de Google Play.

Un fichero APK es una variante del formato JAR de Java. No es más que un fichero en formato comprimido ZIP donde se han empaquetado cuatro tipos de información: el código, los recursos, la firma digital y el fichero de manifiesto. Vamos a realizar algunos ejercicios para aprender más sobre los ficheros APK.

Ejercicio: Descargar un APK usando AirDroid.

Nota: Aunque actualmente está disponible la versión de Apalabrados 3.2.4, en este apartado vamos a utilizar una versión antigua, la 3.1.1. No utilizaremos la versión actual dado que su código ha sido ofuscado. En el siguiente apartado se explica en que consiste la ofuscación. Aunque los ejercicios propuestos funcionan con las dos versiones, hemos considerado conveniente utilizar el código no ofuscado. Además, ambas versiones son operativas y permiten jugar online con cualquiera de ellas.

1. Utilizando un terminal real abre Google Play e instala la aplicación de la que quieres obtener su fichero .apk. En este capítulo vamos a utilizar como ejemplo la aplicación Apalabrados (en concreto, hemos descargado la versión 3.1.1).
2. Una de las formas más sencillas de obtener un fichero .apk es por medio de la aplicación AirDroid. Utiliza de nuevo Google Play para descargarla. Además, AirDroid es una espléndida herramienta que te permitirá gestionar tu terminal desde un ordenador personal.
3. Abre la dirección http://web.airdroid.com desde un navegador web de tu ordenador. En el centro de la pantalla aparecerá el siguiente cuadro de diálogo:

4. Ejecuta en el terminal móvil la aplicación AirDroid que acabas de instalar. Pulsa en el botón que se muestra. Se activará la cámara. Acércala al monitor hasta que el código QR que se muestra en la página web sea reconocido. Esto permitirá establecer una conexión entre ambos dispositivos a través de WiFi. Es necesario que ambos dispositivos estén en la misma red o iniciar sesión en AirDroid para conectarlos.

5. Ya podemos realizar gran número de acciones del teléfono desde nuestro ordenador personal. Por ejemplo: podemos gestionar SMS, música, contactos,

ficheros, tonos de llamada, portapapeles… Dedica unos minutos a explorar las diferentes opciones que nos permite esta gran aplicación.

6. En este ejercicio nos interesa la gestión de aplicaciones, para lo que has de pulsar el botón «Apps»:

7. Se mostrará una lista con todas las aplicaciones instaladas en el terminal. Busca la aplicación Apalabrados y pulsa sobre el botón «Download».

El fichero Apalabrados 3.1.1.apk será descargado a tu ordenador.

Vídeo[Tutorial]: AirDroid: qué es, cómo descargarlo y usarlo.

Disponemos de otras opciones para obtener el apk de una aplicación

1. Existen muchas páginas web que proporcionan los apk en crudo
   - https://www.apkmirror.com/ (apk subidos por los usuarios)
   - https://www.uptodown.com/android (apk desde Play Store, solo para apps gratuitas)
   - https://apps.evozi.com/apk-downloader/ (genera el enlace de descarga para un apk actual de la Play Store)
2. Mediante la utilidad adb, mediante la lína de comandos, es posible descargarse al ordenador el apk de una aplicación actualmente instalada:

Obtenemos la lista de aplicaciones instaladas (eliminar el -3 para ver también las del sistema)

```
adb shell pm list packages -f -3
```

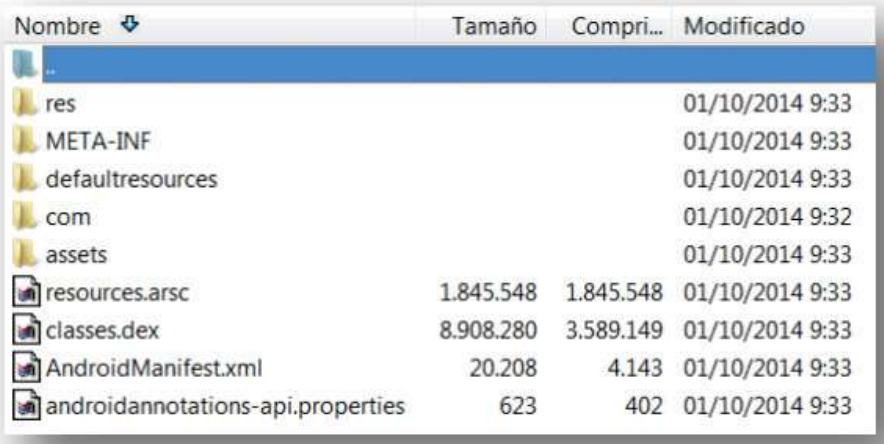
Una vez localizamos la que nos interesa, la descargamos a nuestro ordenador

```
adb pull /data/app/com.etermax.apalabrados-1/base.apk
```

3. Otra opción es utilizar ApkExtractor. Una vez instalado, seleccionar la aplicación deseada y extraer el apk automáticamente a la memoria del teléfono.

Ejercicio: Análisis del contenido de un APK.

1. Arranca un programa que te permita trabajar con ficheros comprimidos. En este ejercicio hemos utilizado WinRAR.
2. Con este programa abre el fichero Apalabrados.apk. Por ejemplo, arrastrando el fichero sobre WinRAR. El resultado ha de ser similar al siguiente:



Nombre	Tamaño	Compri...	Modificado
..			
res			01/10/2014 9:33
META-INF			01/10/2014 9:33
defaultresources			01/10/2014 9:33
com			01/10/2014 9:32
assets			01/10/2014 9:33
resources.arsc	1.845.548	1.845.548	01/10/2014 9:33
classes.dex	8.908.280	3.589.149	01/10/2014 9:33
AndroidManifest.xml	20.208	4.143	01/10/2014 9:33
androidannotations-api.properties	623	402	01/10/2014 9:33

Esta estructura de ficheros y carpetas es similar a la que encontramos en un proyecto Android, aunque tiene sus diferencias. En todo fichero APK siempre se incluyen los siguientes elementos:

assets y **res:** contiene los ficheros de recursos de la aplicación. Conviene recordar que los ficheros XML dentro de la carpeta res no están en formato de texto, sino que se encuentran comprimidos.

META-INF: certificado digital que se ha utilizado para firmar la aplicación. Por supuesto, solo se incluye la clave pública. La clave privada ha de ser custodiada por la empresa propietaria de la aplicación.

clases.dex: contiene el código de las clases de la aplicación. Los diferentes ficheros Java han sido compilados a byte-codes de la máquina virtual Dalvik/ART y agrupados en un fichero DEX.

AdroidManifest.xml: versión comprimida del fichero de manifiesto.

3. Navega por las carpetas res y assets para descubrir los diferentes recursos incluidos en esta aplicación.
4. Trata de abrir un fichero XML dentro de la carpeta res o el AdroidManifest.xml. Verifica que no se trata de un fichero de texto.

Antes de realizar el siguiente paso conviene repasar el concepto de firma digital y cómo las aplicaciones han de ser firmadas antes de su publicación. Si no dominas estos conceptos puedes ver los siguientes vídeos:

Vídeo[Tutorial]: La firma digital.

Vídeo[Tutorial]: Firmar una aplicación Android.

Ejercicio: Estudio de la firma digital de una aplicación.

1. Abre con un visor de ficheros comprimidos la aplicación Apalabrados.apk.
2. Entra dentro de la carpeta META-INF. Encontrarás los siguientes tres ficheros:

   Los ficheros de esta carpeta corresponden a la firma digital de una aplicación Android. De arriba abajo podemos ver el archivo de manifiesto (.MF), el archivo de firma (.SF) y el certificado digital (.RSA).
3. Con un editor de texto abre el archivo de manifiesto: MANIFEST.MF. Verás una lista de todos los archivos incluidos en el paquete, junto con su huella digital en formato resumen SHA1:

```
Manifest-Version: 1.0
Created-By: 1.6.0_29 (Sun Microsystems Inc.)

Name: res/drawable/facebook_icon.png
SHA1-Digest: 9jTbeG4b8nJZlZss16UPfagqlUs=

Name: res/drawable/com_facebook_picker_list_longpressed.9.png
SHA1-Digest: /uU9+qqC9MjC0cwR3L+DskCy9xM=

Name: res/drawable/square_d.xml
SHA1-Digest: kl+o4e30jU9gil/l+A/AUGBcI8U=

Name: res/drawable-ldpi/warply_notifications.png
SHA1-Digest: qaM+xgxBdXq0EfBbTdLffAJ0D70=

...
```

4. Abre el archivo de firma: APAL.SF. Otra vez verás una lista de todos los archivos del paquete, junto con un resumen SHA1. Pero esta vez, cada resumen SHA1 se ha calculado a partir de las dos líneas que describe el fichero en MANIFEST.MF, en lugar de utilizar el fichero en sí. El archivo de firma también contiene un valor de resumen SHA1 para todo el fichero de manifiesto. Este valor se indica en el encabezado `SHA1-Digest-Manifest` .

```
Signature-Version: 1.0
SHA1-Digest-Manifest-Main-Attributes: ngANqza5/whaYZFfNQhzhT/iRBA=
Created-By: 1.6.0_29 (Sun Microsystems Inc.)
SHA1-Digest-Manifest: cbci83Z5qhSYsrmSxoegFDp5I3c=

Name: res/drawable/facebook_icon.png
SHA1-Digest: PDgDHoip/6HBjLGwRLz06l35sy4=

Name: res/drawable/com_facebook_picker_list_longpressed.9.png
SHA1-Digest: 2CGPL/jp5oPTQ2ltrIIWxbJ07UM=

Name: res/drawable/square_d.xml
SHA1-Digest: 5UTm9fULZG3sQPFpSsTpofL7Yro=

Name: res/drawable-ldpi/warply_notifications.png
SHA1-Digest: 4KFSug0Gx0l271gbgNaqEUvi8Fk=

...
```

5. Finalmente, el archivo APAL.RSA contiene un certificado digital. Dentro de este fichero se incluyen dos elementos esenciales para la verificación:

    - La firma del fichero .SF.
    - La clave pública que permite verificar la firma.

6. Vamos a utilizar el comando `keytool` para ver el contenido del certificado digital. Para ello, descomprime el fichero APAL.RSA en una carpeta y desde la línea de comando ejecuta algo similar a:

```
C:\>"C:\Program Files (x86)\Java\jre7\bin\keytool" -printcert -file
APAL.RSA
Propietario: 0=Etermax
Emisor: 0=Etermax
Número de serie: 4e861907
Válido desde: Fri Sep 30 21:31:19 CEST 2011 hasta: Sun Sep 06 21:31:19
CEST 2111
Huellas digitales del Certificado:
         MD5: 05:0A:16:21:5C:61:4C:76:9D:3E:7C:38:65:5F:61:ED
         SHA1: 13:A3:A4:55:F0:C1:26:26:42:2C:52:C4:54:4C:DC:B5:A2:27:52:25
         SHA256: 6D:56:0E:33:C6:C6:9C:A4:6F:37:98:8B:DF:42:CC:8D:E2:BC:BB:E8:
                                        1D:87:36:D6:42:44:18:02:67:D1:F3:EE
         Nombre del Algoritmo de Firma: SHA1withRSA
         Versión: 3
```

Recuerda que este certificado es un certificado autofirmado; es decir, no es verificado por ninguna autoridad de certificación. Esto no nos permite asegurar que el autor de la aplicación sea la empresa Etermax. No obstante, solo el que disponga de la clave privada de este certificado podrá volver a firmar la

aplicación. Por lo tanto, si cambiamos un solo bit del fichero APK se romperá la firma digital, lo que impedirá instalar la aplicación. En los siguientes apartados vamos a modificar parte de la aplicación. Para poder instalarla en nuestro dispositivo vamos a tener que firmarla de nuevo con nuestro propio certificado.

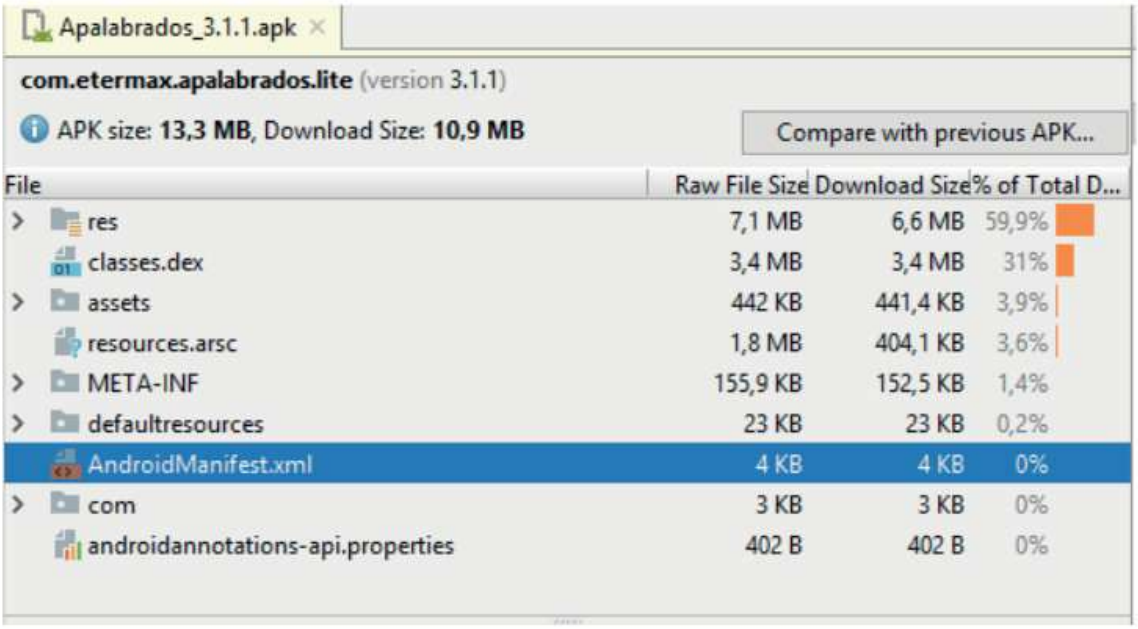
Preguntas de repaso: El formato APK.

7.2. Decompilando aplicaciones Android

7.2.1. Un primer vistazo al contenido de un fichero .apk

Ejercicio: Observar el contenido de un apk desde Android Studio.

1. Desde Android Studio (no importa que proyecto tengas abierto), ve a Build/Analize Apk
2. Selecciona el apk de Apalabrados
3. Observa en la pestaña que se ha abierto el contenido del apk



4. Selecciona el fichero AndroidManifest.xml y encuentra la actividad inicial de la aplicación
5. Observa el contenido de la carpeta de recursos (res). Podrás observar que faltan las carpetas values, entre otras

6. Selecciona el fichero clases.dex y navega por su contenido. Intenta encontrar la clase que define la actividad principal que has encontrado antes en el manifest. Observa como a pesar de poder expandir las clases, solo podemos ver la lista de métodos, pero no la implementación de ellos.

7.2.2. La máquina virtual Dalvik/ART

Dentro de todo APK encontraremos el fichero classes.dex con el código de la aplicación. Los ficheros DEX contienen código ejecutable Android preparado para la máquina virtual Dalvik/ART. Este código es muy similar a los byte codes utilizados en la máquina virtual Java de los ficheros .class. Varios ficheros .class suelen distribuirse de forma conjunta en ficheros JAR.

La gran diferencia entre estas dos máquinas virtuales es que la máquina Java tiene una arquitectura basada en pila mientras que la máquina Dalvik tiene una arquitectura basada en registros. Además, se han introducido otras optimizaciones para que el código de la máquina Dalvik ocupe menos memoria y se han cambiado los códigos de operación para que se ejecuten más rápido. La mayor diferencia es que un byte code se codifican en 8 bits, mientras que un código Dalvik se codifica en 16 bits. Este aumento del tamaño del código permite indicar directamente en la instrucción los operandos con los que ha de trabajar, para lo que se utiliza un campo de 4 bits, también conocido como registro virtual. Esto reduce el número de instrucciones que necesita un programa y aumenta su velocidad de interpretación.

Aunque existe gran diferencia entre los bytes codes y el código Dalvik/ART, resulta sencillo convertir un código en el otro. Dentro del SDK de Android disponemos de la herramienta `dx`, que nos permite convertir ficheros CLASS en ficheros DEX. Además, en este apartado usaremos una herramienta para convertir ficheros DEX en su equivalente JAR.

Para profundizar algo más en la diferencia entre los byte codes y el código Dalvik, vamos a estudiar cómo sería el resultado de compilar un pequeño ejemplo en Java para estas dos máquinas virtuales[1].

```
public static long sumArray(int[] arr) {
    long sum = 0;
    for (int i : arr) {
        sum += i;
    }
    return sum;
}
```

Figura 1. Código fuente en Java (.java).

```
0000: lconst_0       // pila ← 0
0001: lstore_1       // v1 ← pila
```

[1] BORNSTEIN, DAN (2008-05-29). Presentation of Dalvik VM Internals (PDF). Google. p. 22. Retrieved 2010-08-16.

```
0002: aload_0        // pila ← v0 (parámetro arr)
0003: astore_3       // v3 ← pila
0004: aload_3        // pila ← v3
0005: arraylength    // pila ← long. del array en la pila
0006: istore 04      // v4 ← pila
0008: iconst_0       // pila ← 0
0009: istore 05      // v5 ← pila
000b: iload 05       // pila ← v5             (lect.local, escr.pila)
000d: iload 04       // pila ← v4             (lect.local, escr.pila
000f: if_icmpge 0024 // si pila >= pila a fin (lect.pila, lect.pila)
0012: aload_3        // pila ← v3             (lect.local, escr.pila)
0013: iload 05       // pila ← v5             (lect.local, escr.pila
0015: iaload         // pila ← pila[pila]     (2xlect.pila, escr.pila)
0016: istore 06      // v6 ← pila             (lect.pila, escr.pila)
0018: lload_1        // pila ← v1             (2xlect.local, 2xesc.pila)
0019: iload 06       // pila ← v6             (lect.local, escr.pila)
001b: i2l            // pila ← (long)pila     (lect.pila, 2xescr.pila)
001c: ladd           // pila ← pila + pila    (4xlect.pila, 2xescr.pila)
001d: lstore_1       // v1 ← pila             (2xlect.pila, 2xescr.pila)
001e: iinc 05, #+01  // v5++                  (lect.local, escr.local)
0021: goto 000b      // salta a principio del bucle
0024: lload_1        // pila ← v1
0025: lreturn        // retorna el valor en pila
```

Figura 2. Código generado para la máquina virtual Java (.class).
Estadísticas del bucle: 25 bytes, 14 instrucciones, 45 lecturas, 16 escrituras.

```
0000: const-wide/16 v0, #long 0    // v0 ← 0
0002: array-length v2, v8          // v2 ← long del array
0003: const/4 v3, #int 0           // v3 ← 0
0004: move v7, v3                  // v7 ← v3
0005: move-wide v3, v0             // v3 ← v0
0006: move v0, v7                  // v0 ← v7
0007: if-ge v0, v2, 0010           // si v0 >= v2 salta fin     (2xlect.)
0009: aget v1, v8, v0              // v1 ← v8[v0]       (2xlect. escr.)
000b: int-to-long v5, v1           // v5 ← (long)v1     (lect. 2xescr.)
000c: add-long/2addr v3, v5        // v3 ← v3 + v5      (3xlect. 2xescr.)
000d: add-int/lit8 v0, v0, #int 1  // v0 ← v0 + 1       (lect. escr.)
000f: goto 0007                    // salta a principio bucle
0010: return-wide v3               // retorna v3
```

Figura 3. Código generado para la máquina virtual Dalvik (.dex).
Estadísticas del bucle: 18 bytes, 6 instrucciones, 19 lecturas, 6 escrituras.

Vídeo[Tutorial]: Google I/O 2OO8 - Dalvik Virtual Machine Internals.

El proceso de compilación de una aplicación Android no se realiza de forma directa desde código en Java a código Dalvik. Por el contrario, primero se obtienen

los byte codes con un compilador de Java convencional y luego se convierten a código Dalvik por medio de la herramienta `dx`.

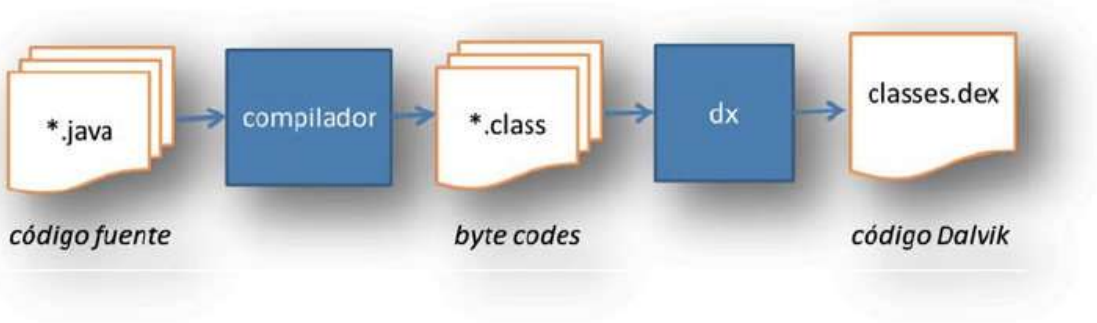


Figura 4. Proceso de compilación de una aplicación Android.

7.2.3. Decompilando aplicaciones Android

En el punto anterior hemos descrito como un compilador Java genera el código para que pueda ser ejecutado por la máquina virtual. En este apartado vamos a describir el proceso inverso: cómo a partir del código ejecutable podemos obtener una versión bastante aproximada del código Java a partir del que se creó. Esto se conoce como decompilar y constituye el proceso más importante en la ingeniería inversa del software. Una de las herramientas más interesantes para decompilar los byte codes es `jd-gui`. Sin embargo, esta herramienta toma como entrada un fichero JAR y el código de las aplicaciones Android se almacena en ficheros DEX. Para salvar este problema vamos a utilizar la herramienta `dex2jar`, que permite transformar el formato. La siguiente figura muestra el proceso a realizar:

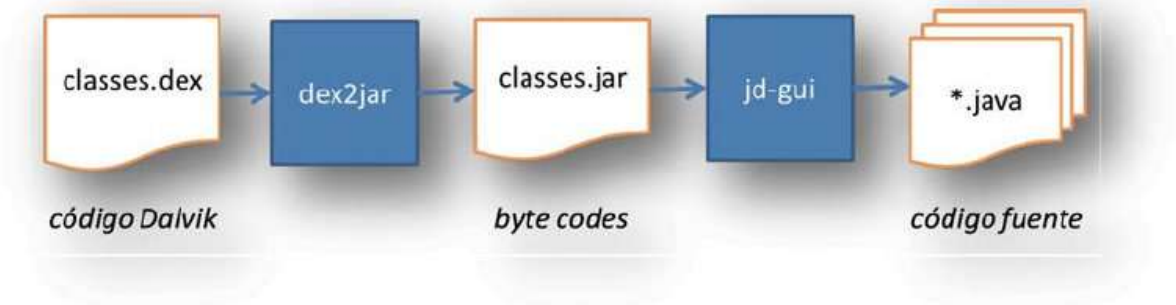


Figura 5. Proceso de decompilación de una aplicación Android.

Ejercicio: Obtención del código Java de una aplicación.

1. Crea una nueva carpeta (por ejemplo C:\C9) y descomprime en ella el fichero clases.dex que encontrarás dentro de Apalabrados.apk.
2. Accede al URL: https://sourceforge.net/projects/dex2jar/files/.
3. En la sección de Downloads descarga el fichero con la última versión (en nuestro caso dex2jar-2.0.zip).

4. Descomprime este zip en la carpeta anterior.
5. Todos los ficheros de la herramienta serán almacenados en la carpeta dex2jar-2.O (o similar). Renombra esta carpeta para que se llame dex2jar. De esta forma será más sencillo acceder a ella.
6. Abre un intérprete de comandos y sitúate en la carpeta creada en este ejercicio.
7. Para transformar el fichero DEX en JAR, escribe el siguiente comando:

```
C:\C9>dex2jar\d2j-dex2jar classes.dex
dex2jar classes.dex -> classes-dex2jar.jar
```

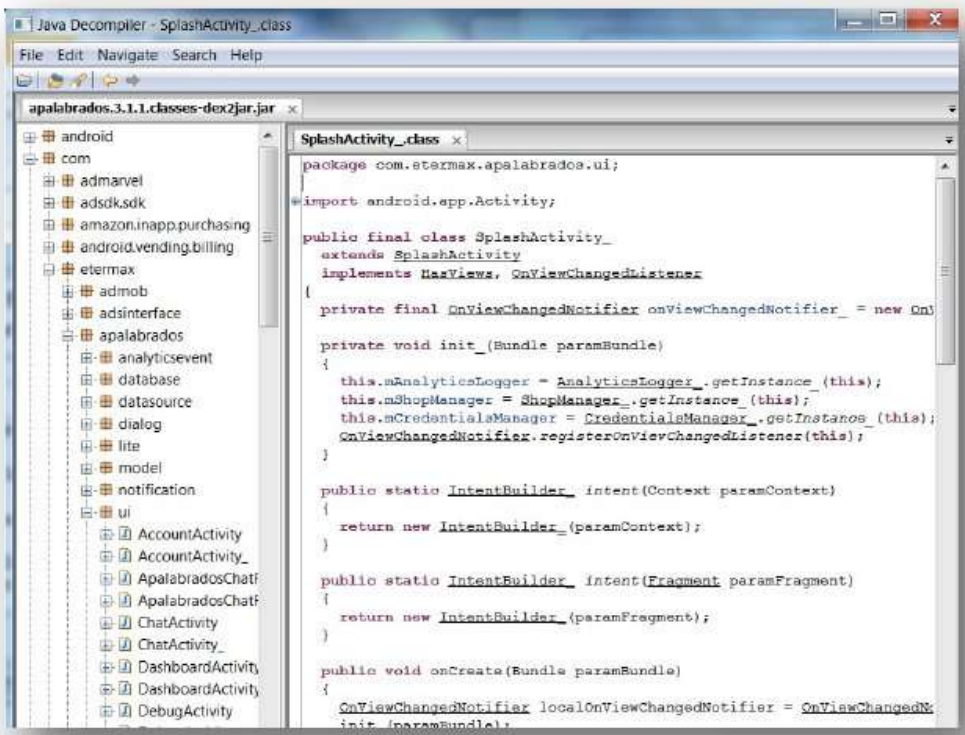
   Verifica que se ha creado el fichero classes-dex2jar.jar.

8. Vamos a descargar el decompilador en la web http://jd.benow.ca/. Observa como existe una versión en modo comando y un plug-in para Eclipse e IntelliJ IDEA (IDE usado en Android Studio). La versión mas recomable es JD-GUI

   Puedes encontrar otros decompiladores en:

   http://sourceforge.net/projects/jadclipse/ y http://www.neshkov.com/dj.html.

9. En este tutorial vamos a usar la versión de JD con interfaz gráfica. Accede a la opción JD-GUI y descarga la versión adecuada para tu sistema operativo. Está disponible para Windows, Linux y OS X.
10. La instalación en Windows es muy sencilla. No tienes más que copiar el fichero jd-gui.exe que encontrarás en el fichero comprimido, dentro de la carpeta creada en este ejercicio. Ejecútalo.
11. Selecciona la opción File/Open File e indica el fichero classes-dex2jar.jar. Si necesidad de realizar más acciones, esta herramienta habrá decompilado todas las clases y te permitirá visualizar el código obtenido.
12. Las clases son agrupadas según el paquete al que pertenecen. Explora alguna de estas clases:



En la imagen se muestra la primera actividad que ejecutará la aplicación, `SplashActivity_` . Veremos más adelante cómo se ha obtenido esta información.

13. Observa como esta clase extiende `SplashActivity` . Si pulsas sobre el nombre de la clase la abrirá.

14. Repite este proceso hasta que no te permita continuar. La jerarquía de clases se muestra a continuación:

```
SplashActivity_
       |
SplashActivity
       |
BaseSplashActivity
       |
FragmentActivity
       |
Activity
```

Más adelante utilizaremos esta información en otro ejercicio.

Preguntas de repaso: Decompilando aplicaciones Android.

7.3. Modificando aplicaciones Android

En este apartado vamos a estudiar cómo podemos modificar una aplicación a partir de su fichero APK. Veremos que es posible modificar alguno de sus recursos o incluso su código y luego volver a generar un nuevo APK. Hay que recordar que el nuevo APK estará firmado por un certificado digital distinto al utilizado en la aplicación original. Este proceso puede ser algo laborioso. Afortunadamente, existen herramientas para automatizarlo, como APK Studio o APK Multi-Tools. Se trata de aplicaciones que combinan varias herramientas por medio de una interfaz de usuario muy sencilla de usar. Dentro de las herramientas que se integran, la piedra angular es apktool. Es una herramienta desarrollada por Google, que facilita el trabajo de ingeniería inversa sobre ficheros APK (http://code.google.com/p/android-apktool).

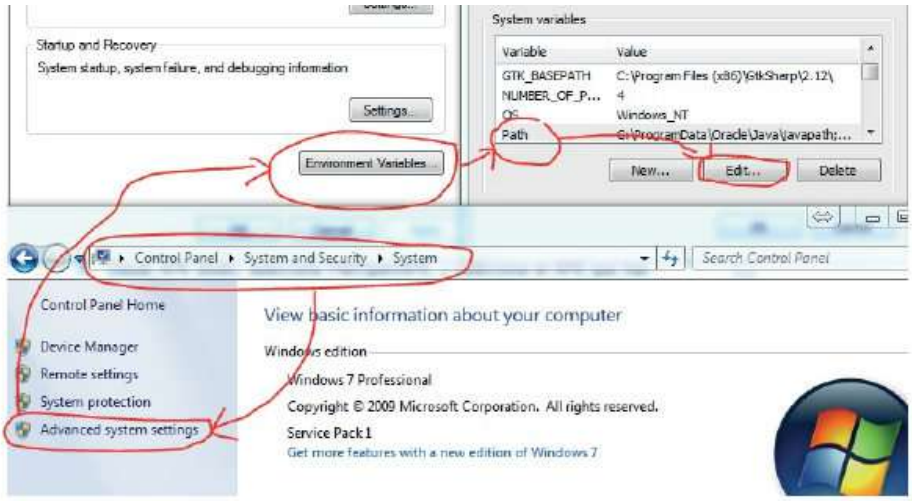
7.3.1. Modificando recursos binarios de una aplicación

La opción más sencilla para modificar una aplicación Android va a ser modificar alguno de sus recursos binarios. Vamos a poder reemplazar cualquier fichero de recurso de imágenes, audio, fuentes… siempre que respetemos el nombre de fichero y su formato.

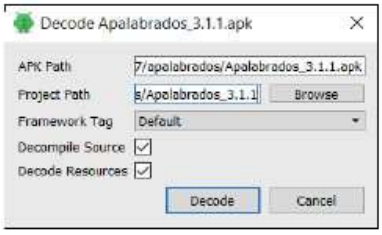
Ejercicio: Modificar recursos de una aplicación Android.

1. Accede a https://bintray.com/vaibhavpandeyvpz/generic/apkstudio/view y en la sección de descargas descarga la versión para tu sistema operativo de APK Studio. Está disponible para Windows y para Linux.
2. Instala la aplicación.
3. Para que funcione la herramienta será necesario que añadas al PATH varias carpetas. En particular, la carpeta platform-tool dentro del sdk de android, la carpeta build-tools/25.0.2 (o la versión más reciente que tengas) del SDK de Android y la carpeta bin de la instalación de JAVA (asegúrate de que sea una instalación de JDK, y no de JRE).

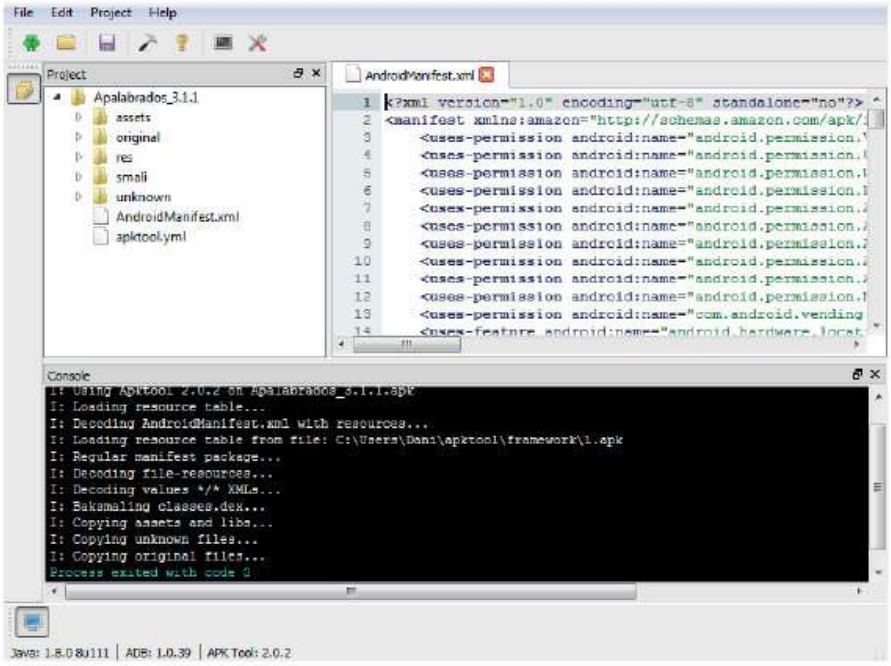
   Nota: Para añadir una carpeta al PATH en Windows, desde el Panel de Control / Sistema y Seguridad / Sistema, haz clic en Configuración Avanzada del Sistema. En la nueva ventana, haz clic en Variables del Entorno y en Variables del Sistema selecciona Path. Añade la ruta que se acaba de indicar. En versiones antiguas de Windows, tendrás que añadir un punto y coma al final del valor actual del Path y pegar la ruta de la nueva carpeta.



4. Copia el APK a modificar a una nueva carpeta, dentro de la misma carpeta utilizada en el apartado anterior.
5. Desde APK Studio, selecciona File/Open/APK, y selecciona el APK anterior.
6. APK Studio preguntará donde ha de crear los ficheros decompilados. Por defecto, se creará una carpeta, en la ubicación actual, con el nombre del APK.



7. Una vez acabe el proceso, se mostrará una ventana como la siguiente. Aquí puedes observar la estructura de ficheros, abrirlos e incluso modificarlos.



8. Sin embargo, será más sencillo abrir los archivos creados con el explorador de archivos y editarlos directamente. Verifica que la carpeta que se acaba de crear tiene el mismo contenido que el que muestra APK Studio.

9. Abre la carpeta res\drawable-hdpi (u otra según densidad gráfica de tu dispositivo) y modifica alguno de los dos ficheros PNG. Por ejemplo, haz doble clic sobre el fichero country es.png (en versiones más antiguas flag es.png) y modifícalo. También puedes modificar los iconos de la aplicación.

10. De vuelta en APK Studio, selecciona Project/Build para crear el APK modificado.

Nota: Es posible que de un error a la hora de hacer este paso. Este error es el siguiente:

```
Exception in thread "main" java.lang.0ut0fMemoryError: Java heap space
    at java.util.Arrays.copy0f(Unknown Source)
    at java.util.ArrayList.grow(Unknown Source)
```

Este error se da porque la aplicación se está quedando sin memoria para realizar las operaciones. Para solucionarlo, en APK Studio, ve a Edit/Settings y modifica la configuración Maximum JAVA Heap a un valor más alto (512 es suficiente para este proyecto). Vuelve a realizar el punto anterior. Verás cómo el error desaparece.

11. A continuación, será necesario firmar la aplicación. Selecciona Project y a continuación Sign/Export. Selecciona el fichero de claves que quieres utilizar para firmar la aplicación e introduce las contraseñas que utilizaste al crearlo.

12. Al acabar el proceso, si no ha habido ningún problema, te dará la opción de abrir la carpeta donde se ha guardado el apk modificado. Este se ha guardado sobrescribiendo el mismo fichero original.

13. Instala este fichero en un dispositivo. Si ya tienes instalada la aplicación original en el dispositivo, aparecerá el siguiente error «Aplicación no instalada. Ya se ha instalado un paquete con el mismo nombre con una firma en conflicto». Dado que el nombre de paquete coincide, tratará de instalarla como una actualización. Sin embargo, la actualización dará error dado que la firma digital no coincide con la original. La solución en este caso consiste en desinstalar primero la aplicación original. Luego ya te permitirá instalar la nueva aplicación.

14. Verifica los cambios realizados en los recursos gráficos:

7.3.2. Modificando recursos XML de una aplicación

Recuerda que dentro de los APK los ficheros XML se almacenan comprimidos en un formato binario. Por lo tanto, no podremos utilizar un editor de texto para editarlos. En este apartado aprenderemos a decodificar estos ficheros y, una vez modificados, volver a codificarlos.

Ejercicio: Modificar recursos XML de una aplicación.

1. Abre de nuevo el fichero AndroidManifest.xml en la misma carpeta del punto anterior.
2. Una de las primeras tareas para analizar una aplicación Android es descubrir la clase encargada de ejecutar la primera actividad. Esta información la encontrarás en el siguiente fragmento del manifiesto:

```
...
<activity android:theme="@*android:style/Theme.NoTitleBar.Fullscreen"
   android:label="@string/app_name"
   android:name="com.etermax.apalabrados.ui.SplashActivity_"
   android:screen0rientation="portrait">
   <intent-filter>
       <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
...
```

Más adelante analizaremos el fichero SplashActivity .java.

3. Edita el fichero res/values-es/strings.xml. Busca la siguiente línea:

```
...
<string name="new_game">Nueva partida</string>
...
```

Reemplaza este texto por otro como «Echar otra partida».

4. Recuerda guardar el fichero y cerrar todos los ficheros abiertos.
5. Vuelve a hacer el Build y a firmar la aplicación como se ha hecho anteriormente para volver a crear un nuevo apk.
6. Instala este fichero en un dispositivo.
7. Verifica los cambios realizados ejecutando la aplicación:

7.3.3. Modificando el código de una aplicación

También resulta posible modificar el código de una aplicación para cambiar su funcionamiento. Por supuesto, nunca hay que distribuir una aplicación manipulada sin la autorización del autor. Nosotros aprenderemos a hacerlo exclusivamente con fines didácticos. Si queremos evitar que manipulen nuestras aplicaciones, primero tenemos que aprender a hacerlo.

Existen dos alternativas para modificar el código. La primera consiste en modificar los códigos Dalvik (o su representación equivalente en ensamblador). La segunda consiste en modificar el código Java decompilado. Aunque la segunda opción resulte la más interesante por tratarse de un lenguaje que conocemos, en la práctica no suele resultar viable.

El código ensamblador es una representación en formato texto equivalente al código máquina. De esta forma va a resultar mucho más fácil entender o modificar un programa. En el caso del código Dalvik se suele representar utilizando el código ensamblador smali (https://code.google.com/p/smali).

Ejercicio: Modificar el código ensamblador de una aplicación.

En primer lugar, debemos tener claro qué parte del código queremos modificar. Hemos visto como la primera actividad de Apalabrados corresponde a la clase SplashActivity . En esta actividad se muestra una vista durante un par de segundos antes de entrar en el juego. Vamos a plantearnos modificar este tiempo para que esté 10 segundos.

1. Repite el ejercicio Obtención del código Java de una aplicación tratando de localizar la clase y la variable donde se programa este tiempo.

   La solución al punto anterior se encuentra en el siguiente fragmento de código:

```
package com.etermax.gamescommon.login.ui;
...

public abstract class BaseSplashActivity extends FragmentActivity
{
  public static final String LAUNCH_DATA = "Launch_data";
  protected static final int LOGIN_REQUEST;
  protected static int SPLASH_DURATION = 2000;
...
```

2. En la carpeta del punto anterior, abre la carpeta Apalabrados 3.1.1\smali. En esta carpeta se ha almacenado el código desensamblado de la aplicación en formato smali. Observa cómo se ha creado una estructura de directorios correspondientes a los nombres de paquete de cada clase.

3. Para acceder a la clase que queremos cambiar abre la carpeta com\etermax\gamescommon\login\ui.

4. Con un editor de texto abre el fichero BaseSplashActivity.smali. Parte de su contenido se muestra a continuación:

```
.class public abstract(1)
Lcom/etermax/gamescommon/login/ui/(2)BaseSplashActivity(3);
.super Landroid/support/v4/app/FragmentActivity;(4)
.source "BaseSplashActivity.java"

# static fields
.field public static final LAUNCH_DATA:Ljava/lang/String; = "launch_data"
.field protected static final L0GIN_REQUEST:I(5)
.field protected static SPLASH_DURATI0N:I(6)

# direct methods
.method static constructor <clinit>()V(7)
    .locals 1
    .prologue
    .line 35
    const/16 v0, 0x7d0
    sput v0, Lcom/etermax/gamescommon/login/ui/BaseSplashActivity;>
                                                    SPLASH_DURATI0N:I
    return-void
.end method
```

Compara la clase original en J ava con su equivalente ensamblador smali. Para ayudarte en la comparación hemos marcado las equivalencias.

```
package com.etermax.gamescommon.login.ui;(2)

import android.content.Intent;

public abstract class(1) BaseSplashActivity(3) extends FragmentActivity(4)
{
  protected static final int LOGIN_REQUEST;(5)
  protected static int SPLASH_DURATION(6) = 2000;
```

(1) en este fichero se define una clase publica y abstracta. (2) se indica el paquete de la clase. Observa como en smali no existen las palabras reservadas `package` ni `import`. Siempre que nos refiramos a una clase es obligatorio indicar el paquete al que pertenece. (3) nombre de la clase. (4) clase de la que heredamos. (5) y (6) declaración de atributos (o campos). En smali `I` es equivalente a `int` en Java. (7) observa como en la clase Java no se declara ningún constructor. Sin embargo en el código smali se declaran dos constructores de forma implícita. El primero inicializa el atributo `SPLASH_DURATION`, asignándole el valor 2OOO (7dO hexadecimal). Analicemos con más detalle el código de este constructor:`.locals` indica a la máquina virtual en número de registros locales que vamos a utilizar en este método. En este caso 1, concretamente `v0`. `.line21` indica la línea del código Java a partir de la cual se ha generado las siguientes instrucciones. La línea 21 en Java corresponde a (6). `const/16 v0, 0x7d0` Almacena en el registro `v0` el valor 2OOO (7dO en hexadecimal). Finalmente, el valor del registro `v0` es transferido al atributo `SPLASH_DURATION`. Observa como al nombre del atributo se antepone su clase con su identificador de paquete. El segundo constructor se limita a llamar al constructor del padre. Los dos constructores que acabamos de ver han de ser llamados cada vez que se crea un nuevo objeto de esta clase.

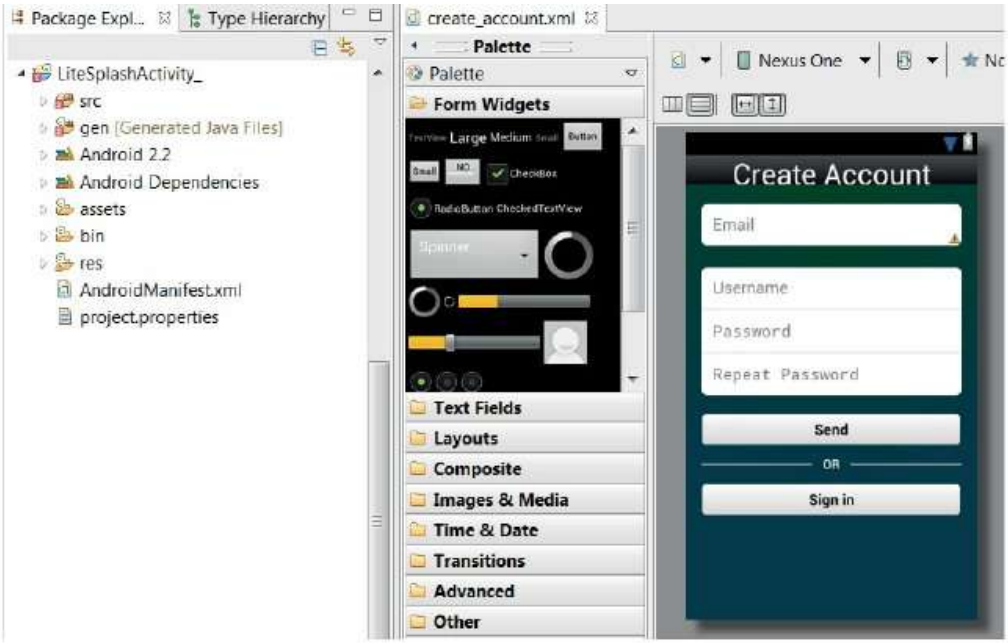
5. Reemplaza el valor `0x7d0` por `0x2710` que equivale a `10000` en decimal. De esta forma alargaremos el tiempo de la actividad splash.
6. En APK Studio, vuelve a generar el apk modificado como en los puntos anteriores.
7. Instala este fichero en un dispositivo móvil.
8. Verifica los cambios realizados ejecutando la aplicación.

Ejercicio: Modificar el código Java de una aplicación Android.

También podemos tratar de crear un proyecto en Eclipse correspondiente a la aplicación Apalabrados; pero ya adelantamos que el resultado no va a ser tan satisfactorio como en el ejercicio anterior. Aunque luego no podamos compilarlo, nos va a ser muy útil disponer del código y los recursos en la herramienta con la que trabajamos habitualmente.

1. Crea una nueva carpeta con nombre Proyecto. Copia dentro las carpetas assets, res y el fichero AndroidManifest.xml que encontrarás en la carpeta creada en los ejercicios anteriores.
2. Ejecuta la aplicación jd-gui y abre el fichero classes-dex2jar.jar que encontrarás dentro de Apalabrados.apk, tal y como se hizo en el ejercicio Obtención del código Java de una aplicación.
3. Selecciona la opción File/Save All Source para guardar en código Java en un fichero ZIP.

4. Crea la carpeta src dentro de la carpeta Proyecto y descomprime en ella el contenido de este ZIP.
5. Ejecuta Android Studio y selecciona File/New/Import Project. Selecciona la carpeta Proyecto, haz clic en «Next» y, a continuación, en «Finish». Se creará el proyecto Proyecto. El nombre se ha tomado del nombre de la carpeta.



6. Observa que el número de errores es importante. Será muy complicado llegar a eliminarlos todos, por lo que no podremos crear la aplicación a partir de este proyecto. No obstante, aunque el proyecto no compila, disponemos del código y los recursos de esta aplicación dentro de nuestro entorno de programación.

Práctica: Modificar otro comportamiento de la aplicación.

¿Te animarías a realizar otros cambios en la aplicación? Por ejemplo, elimina las consultas para obtener la posición del dispositivo.

Preguntas de repaso: Modificando aplicaciones Android.

Desafío: Decompilar y modificar una aplicación.

Es hora de aplicar lo aprendido, pero esta vez sobre una aplicación de tu elección. No es imprescindible que realices todos los puntos indicados.

1. Instala en tu móvil una aplicación diferente a Apalabrados y descarga su fichero apk.
2. Decompila esta aplicación y estudia su código.
3. Trata de modificar alguna imagen, cambiar algún texto, color o tamaño de fuente.
4. Plantéate como reto modificar su comportamiento. Por ejemplo, que al seleccionar una acción del ActionBar esta no se realice.
5. Trata de crear un proyecto para Eclipse o Android Estudio que contenga tanto el código como los recursos de esta aplicación. Si aparecen errores y crees que es viable, trata de eliminarlos.
6. Redacta un breve informe de forma esquemática donde se indique:
   - Aplicación, versión, nombre del paquete, si está ofuscada y primera actividad que se ejecuta
   - Alguna curiosidad encontrada en él código o los recursos.
   - Lista de recursos modificados y en qué actividad se aprecia el cambio.
   - Comportamiento que ha sido alterado y fichero smali modificado indicando las líneas que han sido editadas.
   - ¿El proyecto creado presentaba errores? ¿Ha sido posible corregirlos?
7. Si estás matriculado en un curso, crea un nuevo hilo en el foro de la asignatura con nombre "Aplicación___ decompilada y modificada". Copia el informe anterior y adjunta el fichero apk modificado y un zip con el proyecto creado.

7.4. Ofuscación del código

La ofuscación de código consiste en reordenar o alterar las instrucciones de un programa para que, aunque realice la misma función, sea más difícil su comprensión. En consecuencia, la ofuscación es la herramienta más importante para evitar la ingeniería inversa.

Como hemos podido comprobar en los apartados anteriores, cuando se compila Java se incluye información para depurar código en los byte codes (o código Dalvik). En esta información se incluyen los números de líneas a partir de las que se ha generado el código o los nombres de las variables. Por ejemplo, en la clase BaseSplashActivity de la aplicación Apalabrados, dentro del fichero DEX se ha incluido información como que existe una variable `SPLASH_DURATION` con nombre y que esta variable era inicializada en la línea 21 del fichero Java.

```
...
.field protected static SPLASH_DURATION:I

# direct methods
.method static constructor <clinit>()V
```

```
    .locals 1
    .prologue
    .line 35
...
```

Incluir esta información en una aplicación a la hora de distribuirla no parece muy razonable. No solamente estamos publicando los algoritmos que hemos desarrollado, sino que además estamos aumentando el tamaño del código con información que no es necesaria en tiempo de ejecución.

Una de las herramientas más utilizadas para ofuscar código Java es ProGuard. Además de ofuscar el código, lo optimiza, reduciendo su tamaño. En concreto esta herramienta realiza las siguientes acciones:

- elimina variables, clases, métodos y atributos no utilizados;
- elimina instrucciones innecesarias;
- elimina la información de depuración;
- renombra las clases, campos y métodos con nombres poco legibles.

ProGuard es parte del plugin de Android para Eclipse, por lo que no tienes que instalarlo ni invocarlo de forma manual. Solo tienes que activarlo en las propiedades del proyecto. En concreto, hay que establecer la propiedad `proguard.config` en el archivo project.properties.

En algunas situaciones ProGuard podría equivocarse y eliminar código que supuestamente no es utilizado, pero que en realidad es necesario para la aplicación. Algunos ejemplos incluyen:

- una clase que se hace referencia solo desde AndroidManifest.xml;
- un método llamado desde JNI;
- campos y métodos referenciados dinámicamente.

Ejercicio: Uso de Proguard para ofuscar una aplicación Android.

En este primer ejercicio vamos crear una aplicación sin activar las opciones de ofuscación y comprobaremos que resulta muy sencillo realizar la ingeniería inversa. Luego configuraremos Proguard para que ofusque el código y veremos que ya no es tan fácil realizar la ingeniería inversa.

1. Crea un nuevo proyecto con nombre Hola Mundo y paquete `com.example.holamundo`.
2. Añade el siguiente código a la actividad principal:

```
    private String cadena = "Hola Mundo";

    public String bienvenida (){
        return cadena;
    }
```

```
    public void noUsada() {}

    protected void onCreate(Bundle paramBundle) {

        Toast.makeText(this, bienvenida(),Toast.LENGTH_L0NG).show();
    }
```

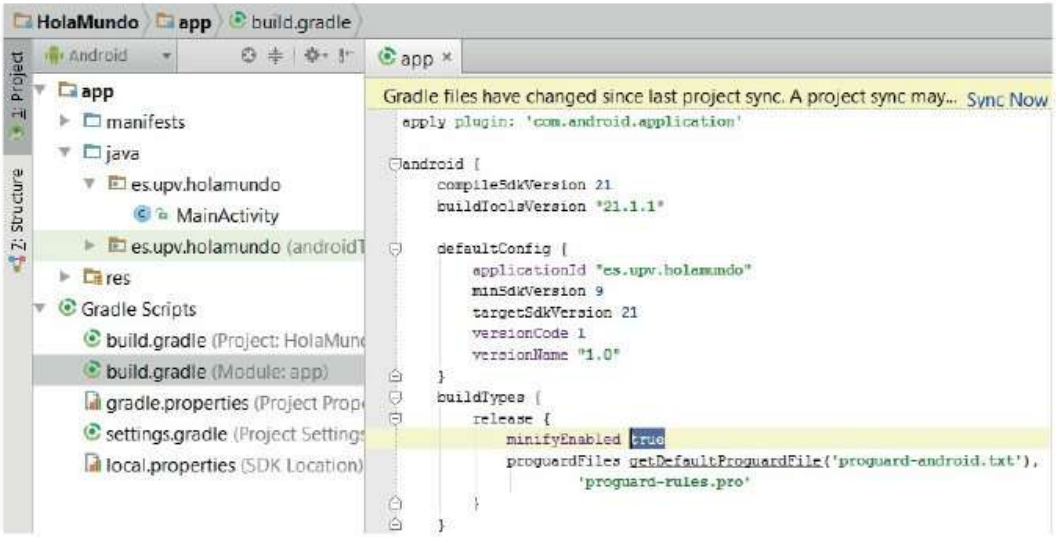
3. Vamos a crear el APK para distribución sin ofuscación. En **Android Studio**: Build > Build APK.
4. Finamente guarda el APK resultante en la carpeta de trabajo creada para este capítulo (C9). Renombra el fichero creado a no ofuscado.apk.
5. Abre el fichero con un lector de ZIP y extrae el fichero classes.dex.
6. Renombra este fichero para que se llame classes no ofusc.dex.
7. Para transformar el fichero DEX en JAR, escribe el siguiente comando:

```
C:\C9>dex2jar\d2j-dex2jar classes no ofusc.dex
dex2jar classes_no_ofusc.dex -> classes_no_ofusc-dex2jar.jar
```

Verifica que se ha creado el fichero classes no ofusc-dex2jar.jar.

8. Ejecuta jd-gui, selecciona la opción File/Open File. e indica el fichero classes no ofusc-dex2jar.jar.
9. Observa como el código de las clases Java que componen la aplicación puede leerse perfectamente.
10. Ahora vamos a crear de nuevo la aplicación ofuscando el código:

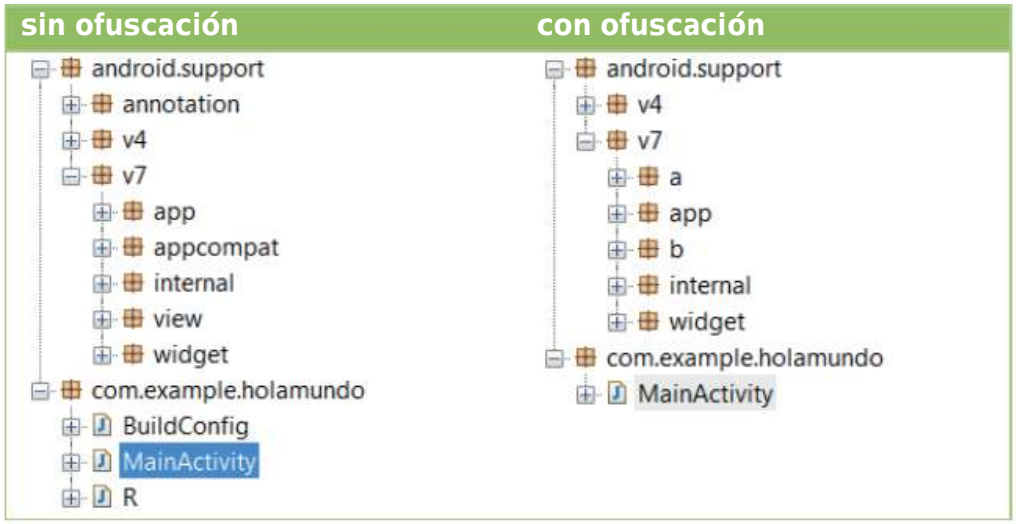
En **Android Studio** es muy sencillo. Desde la vista de proyecto en modo Android, abre el fichero Gradle Scripts/build.gradle (Module: app). Busca la propiedad `minifyEnable` y cambiala a **`true`**. De esta forma se activa la minimización de código utilizando los ficheros Proguard que se indican en la siguiente línea:



Tras el cambio guarda el fichero y pulsa en Sync Now (esquina superior derecha) para que este cambio tenga efecto. No es necesario que configures un fichero de Proguar, dado que ya incorpora uno Android Studio.

11. Repite los pasos 3 al 8; pero esta vez renombra los ficheros APK y DEX a ofuscado.apk y classes ofuscado.dex.

12. Compara las clases generadas para ambos proyectos.



Observa como tras la ofuscación las clases de nuestro paquete `R` y `BuidConfig` han desaparecido. Pambien ha desaparecido todo el paquete `android.support.annotation`. Todas estas clases contenían constantes. Como veremos más adelante, Proguard ha reemplazado estas constantes por el valor correspondiente, por lo que ya no es necesario declararlas.

Observa como algunas las clases `appcompat` e `internal` han sido renombradas como `a` y `b`. Cuando una clase es usada desde el sistema Android su nombre no puede ser cambiado, dado que si no no sería encontrada. Ocurre lo mismo cuando una clase es referenciada desde código no Java, por ejemplo desde Androidmanifest.xml. Por el contrario, cuando una clase es usada exclusivamente dentro de nuestra aplicación Proguard va a poder renombrarla.

13. Compara las dos clases `MainActivity` que han sido decompiladas. El siguiente fragmento de código muestra el código eliminado y por que ha sido reemplazado:

```
public class MainActivity extends ActionBarActivity → e {

    private String cadena → n = "Hola Mundo";

    public String bienvenida → i(){
        return cadena → n;
    }

    public void noUsada() {}

    protected void onCreate(Bundle paramBundle) {
        super.onCreate(paramBundle);
```

```
        setContentView(2130903063);
        Toast.makeText(this, bienvenida ➔ i (), 1).show();;
    }
```

A modo de resumen resaltamos: Los identificadores (de clase, método o variable) que solo son usados en nuestra aplicación son renombrados. El código no utilizado es eliminado. Las cadenas de caracteres no son ofuscadas.

Tras este experimento puede parecer que el código sigue siendo muy fácil de interpretar. Es debido a que se llama mayoritariamente a código del sistema, que no han podido ser ofuscados. Cuando se aplique estas técnicas a código escrito por nosotros, el resultado será muy diferente.

14. Utilizar Proguard para ofuscar el código tiene la ventaja adicional de que el código generado es más pequeño. Compara el tamaño de los dos ficheros DEX generados.

Nombre	Tamaño
classes no ofusc.dex	1634 KB
classes ofuscado.dex	564 KB

Esta espectacular reducción del tamaño es debida a: 1. La información de debug ha sido eliminada (nombres de ficheros fuente, número de línea de la que procede el código,). 2. Parte del código puede haber sido eliminada (clases o métodos no utilizados). 3. Los nombres de los identificadores ahora son mucho más cortos (~~ActionBarActivity~~ → e).

15. Vamos a comparar las dos aplicaciones creadas con APK Studio. Para ello, copia los dos ficheros APK a una nueva carpeta.

16. Ejecuta APK Studio y decompila ambas aplicaciones.

17. Abre los ficheros: */smali/com/example/holamundo/MainActivity.smali y compara el código generado en ambos casos.

Preguntas de repaso: Ofuscación en Android.

7.5. Obtención de licencias con Google Play

Google Play ofrece un servicio de concesión de licencias que te permite consultar en tiempo de ejecución si el usuario actual dispone de licencia para usar tu aplicación. Su uso más frecuente es asegurarnos que en las aplicaciones de pago el usuario realmente ha pagado. No obstante, también puede ser muy útil en cualquier tipo de aplicación, para asegurarnos de que se está utilizando la aplicación original.

Con este servicio se puede aplicar una política de licencias flexibles. Una aplicación puede solicitar restricciones personalizadas basadas en el estado de la licencia

obtenida de Google Play. Por ejemplo, una aplicación puede comprobar el estado de la licencia y luego aplicar restricciones personalizadas que permiten que el usuario lo ejecute sin licencia por un período limitado o permitiendo un uso parcial de la aplicación.

Para ayudar a verificar licencias, el SDK de Android proporciona un conjunto descargable de fuentes de biblioteca. La biblioteca de verificación de licencias (LVL) se puede agregar a tu aplicación de manera que solo tienes que llamar a un método para conocer el estado de la licencia.

Para usar el servicio de licencias tu aplicación ha de tener un nivel mínimo de API de 3 (v1.5). Otra restricción es que en el terminal ha de estar preinstalada la aplicación cliente de Google Play, dado que es realmente este componente el que se comunicará con el servidor de licencias. En los emuladores no se incluye el cliente de Google Play con plena funcionalidad, lo que dificulta la comprobación de los ejercicios de este capítulo. Si no puedes utilizar un terminal real, tendrás que realizar algún ajuste para conseguir que funcione sobre un emulador[2].

7.5.1. Cómo funciona el servicio de licencias

El servicio de licencias de Google Play es un mecanismo basado en red que permite consultar a la aplicación si el usuario del dispositivo actual dispone de licencia. El siguiente diagrama ilustra el funcionamiento de este servicio:

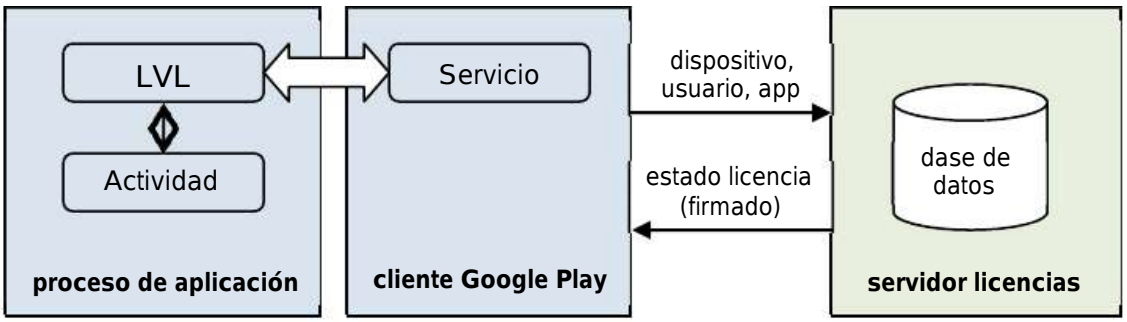


Figura 6. Elementos involucrados en la verificación de una licencia.

La solicitud se inicia cuando la aplicación realiza una solicitud de comprobación de licencia. Para ello utilizará la librería LVL, en concreto el método `LicenseChecker.checkAccess()`. Desde esta librería se invoca un servicio ofrecido por la aplicación cliente de Google Play. A continuación, la aplicación de Google Play envía una petición al servidor de licencias y recibe el resultado. La aplicación Google Play envía el resultado a la librería LVL, desde donde se invoca un método callback diferente según el resultado. Estos métodos callback (`allow()`, `dontAllow()`, `applicationError()`) han de estar definidos en tu aplicación.

[2] http://developer.android.com/google/play/licensing/setting-up.html

Para hacer la consulta de licencia hay que identificar la aplicación, el usuario y el dispositivo: la aplicación se identifica mediante nombre del paquete, el usuario mediante la cuenta Google configurada en el dispositivo y el dispositivo mediante el IMSI y otra información. El servidor de Google Play evalúa la solicitud utilizando esta información. Comprueba la identidad del usuario con los registros de compra o instalación y devuelve una respuesta.

Para asegurar la integridad de la respuesta, el servidor la firma antes de enviarla. Para cada aplicación que demos de alta en el servicio de licencias se van a crear un par de claves RSA de 2048 bits. La clave privada solo es conocida por el servidor de licencias, mientras que la pública ha de ser conocida por nuestra aplicación. Como es habitual, la clave privada se utilizará para firmar las respuestas del servidor y la pública para verificar esta firma. Esta forma de trabajar impide que se pueda engañar al sistema con respuestas falsificadas.

7.5.2. Como añadir una licencia a nuestra aplicación

En este apartado vamos a describir los pasos a seguir para usar licencias de Google Play en una de tus aplicaciones. Luego haremos un ejercicio paso a paso para ver estos pasos en la práctica.

A continuación, detallamos estos pasos. No es necesario seguir este orden:

1. **Dar de alta la aplicación en Google Play.** Para esto es imprescindible disponer de una cuenta en Google Play Developer Console[3]. Dar de alta una cuenta cuesta 25 $ para todas las aplicaciones que queramos publicar y por tiempo indefinido.
2. **Obtener la clave de licencia de esta aplicación.** Como se ha explicado para cada aplicación registrada en Google Play se van a generar un par de claves RSA. Nosotros solo tendremos acceso a la clave pública.
3. **Publicar la aplicación, aunque sea en modo beta cerrada.**
4. **Añadir la librería LVL a nuestro proyecto.** La forma recomendada es copiar las clases directamente al proyecto. Así facilitamos su ofuscación.
5. **Pedir el permiso de licencia.** Para ello editaremos el manifiesto de nuestra aplicación y usaremos el permiso correspondiente.
6. **Definir una política de licencias.** Hay que definir cosas como lo estricto que vamos a hacer cuando el usuario no disponga de conexión a Internet. Le denegamos la licencia, permitimos una caché, cuánto tiempo . Para implementar esta política podemos crear un descendiente de la clase `Policy`. Aunque la librería ya incorpora un par de políticas que pueden ser adecuadas para la mayoría de los casos.
7. **Crear una instancia de la clase `LicenseChecker`.** Será nuestro manejador de licencia. Al crearlo hay que indicarle nuestro contexto, la política de licencias y la clave pública de la licencia. El objeto creado establecerá una

[3] https://play.google.com/apps/publish

conexión con un servicio (bind) de la aplicación Google Play cliente de nuestro dispositivo.

8. **Verificar si disponemos de licencia.** No tenemos más que llamar al método `checkAccess()` del objeto creado en el punto anterior. Se trata de un método asíncrono, por lo que para analizar su respuesta tendremos que realizar el siguiente paso.
9. **Implementar la interfaz `LicenseCheckerCallback`** . Esta interfaz define tres métodos callbacks: `allow()`, `dontAllow()` y `applicationError()`. Cada uno de estos métodos será invocado según el resultado de la consulta realizada en el punto anterior.
10. **Llamar al método onDestroy() del `LicenseChecker`** . Será necesario cerrar la comunicación entre procesos (IPC) establecida al crear la instancia de esta clase (ver punto 6) para evitar dejar consultas a medias. El sitio ideal para realizar esta llamada es el método `onDestroy()` de nuestra actividad.

Veamos cómo realizar estos pasos en el siguiente ejercicio:

Ejercicio: Uso de la librería LVL.

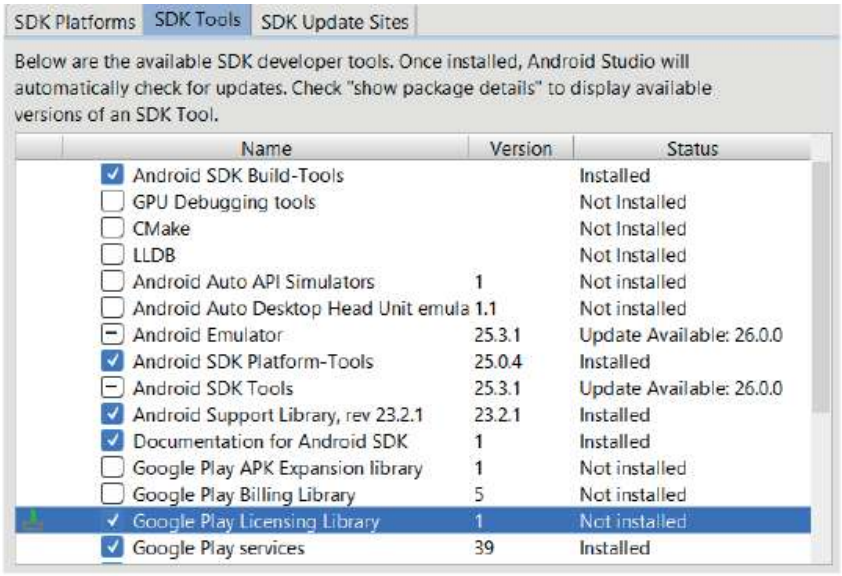
1. Crea un nuevo proyecto con los siguientes datos. Reemplaza garcia.moreno.pedro por tus apellidos seguido de tu nombre:

```
Application Name: ObtencionLicencia
Package name: org.garcia.moreno.pedro.obtencionlicencia
Minimum SDK: API 9 Android 2.3 (Gingerbread)
```

   Utiliza los valores por defecto para el resto de los campos.

   Nota: En este ejercicio no se puede utilizar el paquete com.example dado que hemos de publicar la aplicación en Google Play y este prefijo de paquete no está permitido. Además, vamos a utilizar nuestro nombre para asegurarnos de que el nombre del paquete no coincide con el de otros compañeros.

2. El primer paso será descargar la librería LVL. Para ello, entra en Android SDK Manager  y descarga el paquete Google Play Licensing Library.



3. Abre la carpeta <sdk>\extras\google\market licensing\library. <sdk> corresponde al directorio donde tienes instalado Android SDK. Para localizarlo, abre el SDK manager y tendrás la ruta en la parte superior de la ventana.

4. Para utilizar la librería será necesario importarla a Android Studio. Para ello, haz clic en File/New/Import Module y selecciona la carpeta anterior. Nombra la nueva librería como lvl.

5. Para importar la librería será también necesario añadir la siguiente línea en el build.gradle del módulo app, dentro de dependencies. Esta línea le indica que el módulo app utilizará el código de la librería.

```
compile project(path: ':lvl')
```

6. Dado que esta librería se compila por separado, será necesario activar en su propio build.gradle el parámetro `minifyEnable true` Resulta importante ofuscar nombres de los métodos y variables de la librería.

7. Para hacer las pruebas, vamos a introducir dos botones en el layout. Uno que comprobará si la licencia es válida y otro para entrar en la aplicación. Sustituye el contenido de activity main.xml por el siguiente código:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="comprobarLicencia"
        android:text="Comprobar Licencia" />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:onClick="entrar"
        android:text="Entrar" />
</LinearLayout>
```

8. A continuación, reemplaza el código de MainActivity por el siguiente. No te olvides de incluir los `imports` necesarios:

```
public class MainActivity extends Activity implements LicenseCheckerCallback {
    private static final String CLAVE_PUBLICA_LICENCIA = "Tu clave pública";
    // Genera 20 bytes aleatorios, y reemplazalos por los siguientes.
    private static final byte[] SALT = new byte[] {-46, 65, 30,-128,-103,
            -57, 74,-64, 51, 88,-95,-45, 77,-117,-36,-113,-11, 32,-64, 89};
    LicenseChecker comprobarLicencia;
    boolean permitir = false;
    ProgressDialog dialogo;

    @0verride protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        String idDispositivo = Secure.getString(getContentResolver(),
                                                Secure.ANDROID_ID);
        ServerManagedPolicy politica = new ServerManagedPolicy(this,
                new AES0bfuscator(SALT, getPackageName(), idDispositivo));
        comprobarLicencia = new LicenseChecker(this, politica,
                                               CLAVE_PUBLICA_LICENCIA);
        dialogo = new ProgressDialog(this);
        dialogo.setTitle("comprobando licencia");
        dialogo.setIndeterminate(true);
    }

    public void comprobarLicencia(View view) {
        dialogo.show();
        comprobarLicencia.checkAccess(this);
    }

    public void entrar(View view) {
        if (permitir) {
            Toast.makeText(this,"Entrando en aplicación",
                                            Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(this, "Licencia no válida",
                                            Toast.LENGTH_LONG).show();
        }
    }

    @0verride public void allow(int reason) {
        permitir = true;
        Toast.makeText(this,"Licencia correcta: "+reason,
                                            Toast.LENGTH_LONG).show();
        dialogo.dismiss();
    }

    @0verride public void dontAllow(int reason) {
```

```
        permitir = false;
        Toast.makeText(this,"Licencia no válida: "+reason,
                                                Toast.LENGTH_LONG).show();
        dialogo.dismiss();
    }

    @0verride public void applicationError(int errorCode) {
        Toast.makeText(this, "Error al comprobar licencia: " + errorCode,
                                                Toast.LENGTH_LONG).show();
        dialogo.dismiss();
    }

    @0verride protected void onDestroy() {
        super.onDestroy();
        comprobarLicencia.onDestroy();
    }
}
```

En esta actividad comenzamos declarando la constante `CLAVE_PUBLICA_LICENCIA`. Una vez registrada la aplicación en Google Play se creará una clave que tendremos que introducir en esta constante.

Dentro del onCreate(), después de establecer la vista a mostrar, averiguamos el `ANDROID_ID` del dispositivo. Se trata de un número de 64 bits (expresado en una cadena hexadecimal) que se genera aleatoriamente en el primer arranque del dispositivo y debe permanecer constante durante toda la vida del dispositivo. Luego definimos la política de licencias. Se ha seleccionado una de las predefinidas en la librería (`ServerManagedPolicy`). Al definir la política también definimos un ofuscador. Típicamente la política de licencias tiene que guardar los datos de respuesta de licencia en un almacenamiento persistente. Por ejemplo, una política podría mantener la fecha de la última comprobación correcta, el número de reintentos, el período de validez de la licencia, etc. Para evitar la manipulación de estos datos resulta conveniente almacenarlos de forma encriptada. En la librería LVL se incluye el encriptador `AESObfuscator`[4]. Para parametrizarlo hay que indicar tres semillas: una aleatoria (`SALT`), una que dependa de la aplicación (el nombre del paquete) y una que dependa del dispositivo (`idDispositivo`).

A continuación creamos un `LicenseChecker`, que será el encargado de comprobar la licencia. Como argumentos para la creación, le pasamos un contexto, una política y nuestra clave de licencia. Este método termina creando un cuadro de diálogo que será mostrado cuando estemos compro- bando la licencia.

Los siguientes dos métodos están asociados a la pulsación de los dos botones. Si pulsamos sobre el botón de comprobar licencia, el código llama a `checkAccess(this)`. El argumento que le pasamos ha de ser un objeto que implemente la interfaz `LicenseCheckerCallback`; en este ejemplo, nosotros

[4] http://es.wikipedia.org/wiki/Advanced_Encryption_Standard

mismos. Si pulsamos en botón «Entrar», simplemente se muestra el valor de la variable `permitir`.

Los siguientes tres métodos corresponden a la interfaz `LicenseCheckerCallback`. Uno de estos métodos será llamado tras invocar a `checkAccess()`. Finalmente, en el método `onDestroy()` tenemos que asegurarnos de que se cierra la comunicación con la aplicación Google Play.

9. En el manifiesto añade el siguiente permiso:

```
<uses-permission
        android:name="com.android.vending.CHECK_LICENSE" />
```

No hace falta pedir permiso de Internet, ya que es la aplicación Google Play la que se conectará a Internet para verificar la licencia, y no la nuestra.

10. En el siguiente paso vamos a dar de alta nuestra aplicación en Google Play Developer Console. Entra en el URL https://play.google.com/apps/publish.
11. Si no dispones de una cuenta puedes crear una nueva pagando 25 $. Durante el curso puedes pedir una autorización para utilizar la cuenta «Android Curso» del usuario androidcurso2013. Tienes que indicar una cuenta de Google al profesor para que la autorice.
12. Pulsa el botón «+ Crear una aplicación». Introduce en el campo Nombre tu apellido seguido de una coma y de tu nombre. Pulsa el botón «Preparar entrada de Play Store». Tampoco es necesario que introduzcas otros datos de distribución.
13. El siguiente paso será conseguir la clave pública de la licencia de nuestra aplicación. Podrás ver la clave de licencia dentro de la información de tu aplicación, en la pestaña de Servicios y API. Busca el campo «Tu clave de licencia para esta aplicación», cópialo y pégalo como valor para la constante `CLAVE_PUBLICA_LICENCIA`, definida al principio de `MainActivity`.

LICENCIAS Y FACTURACIÓN INTEGRADAS EN APLICACIONES

Nota: *La licencia permite evitar la distribución no autorizada de la aplicación. También puedes utilizarla para verificar las compras de facturación integrada en la aplicación. Más información sobre licencias.*

TU CLAVE DE LICENCIA PARA ESTA APLICACIÓN

Clave pública RSA codificada en base 64 para incluirla en tu código binario. Elimina todos los espacios.

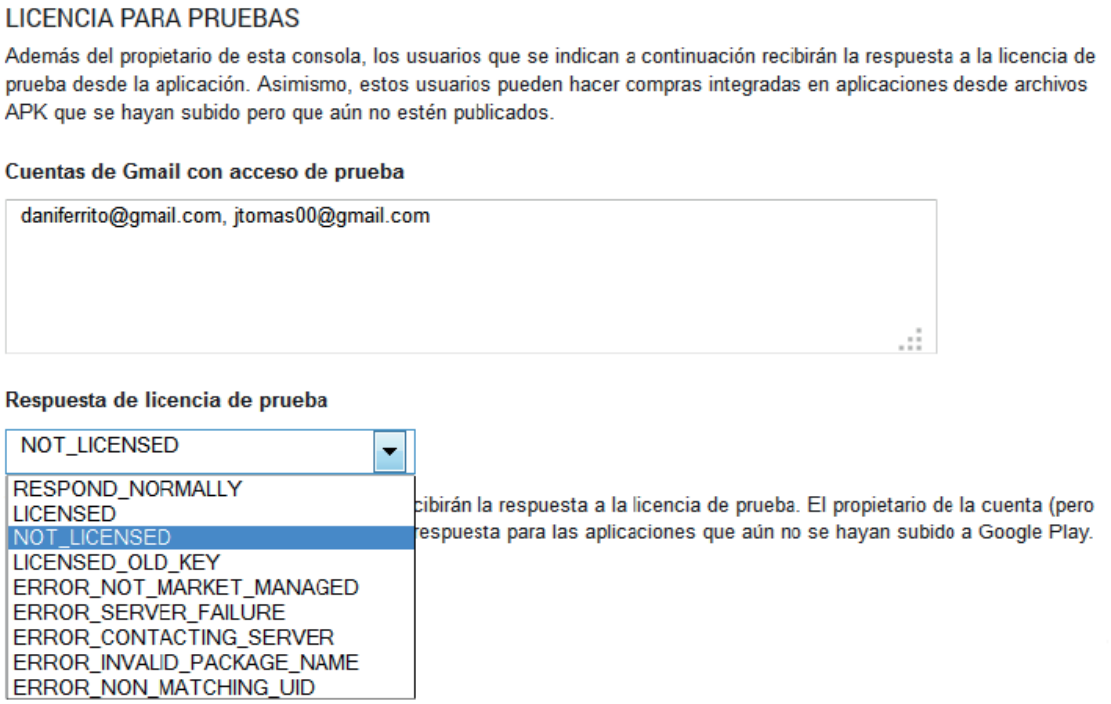
```
MIIBIjAN-
BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkCO7fHuf5p0k1KZ87LftnjRd7EQ7
HYHii5BthZ/IdROBS3Wt2PHkxSKRm0dTWgjfytyNNpylzkdZdA8CMq4eTM7e43Tn
n+5xs+rN3ZTNPxHNrCfcNajC8X7D4b54cp1/eOb4Iq+yVWf6Zfwk99FwkGsvZtLU
8G0WvyquGFF2U+SqbRPrgK4XJjfX9+X8llvw/6pkkalR8yxE+bxZ9VGgvLucLRYi
lHj6EpPNaGBkSVxOj+7PYw60KN15I0IPYM3U8zNKjTFJrQVmlQcAWHwLwD2bhDwj
WS8R+9AFj59dPJFrK2yKwIPIoSCPuN4rwcstKw78FaLWFHXAKwUwLa/0SQIDAQAB
```

14. Firma la aplicación con un certificado definitivo y crea el APK. Para ello puedes utilizar la opción Build / Generate Signed APK... Para más información, puedes ver el Vídeo[Tutorial]: Firmar una aplicación Android.

15. Ahora, tendrás que subir el APK ya firmado a Google Play. Selecciona la pestaña APK y luego la BETA TESTING. Luego, Subir nuevo APK.

16. Será necesario publicar la aplicación, aunque sea en modo de beta cerrada. Para ello es necesario que rellenes los datos como si fueses a publicar la aplicación, aunque no sean los datos finales.

17. Entra dentro de Google Play Developer Console y, en la izquierda, selecciona la pestaña de Configuración ( ). Si no eres el propietario de esta consola, añade tu cuenta de Gmail en la lista de licencias para pruebas:



18. Justo debajo puedes configurar la respuesta que dará el servidor para todas las aplicaciones con licencia de prueba. Si otros compañeros están haciendo pruebas es posible que se modifique este valor.

19. Ejecuta la aplicación sobre un terminal real. Puedes instalar el APK firmado o el que genera Android Studio en modo debug. Es indiferente dado que para verificar la licencia solo se utiliza el nombre del paquete, no se tiene en cuenta si está firmada.

Nota: La verificación de licencia se hace con la cuenta primaria del dispositivo.

20. Después de pulsar sobre el botón de comprobar licencia, se mostrará en un toast la respuesta del servidor. Si aparece el «Error al comprobar licencia: 3» quiere decir que Google Play desconoce el paquete (véase tabla de errores al final). Tendrás que esperar alrededor de media hora hasta que se actualice la información en los servidores de Google Play.

21. Pasado este tiempo, pon el valor de Respuesta de licencia de prueba a LICENSED y verifica que obtienes la licencia. El botón «Entrar» te permitiría entrar en la aplicación.

22. Pon el valor de Respuesta de licencia de prueba a NOT_LICENSED y verifica que ya no obtienes la licencia desde tu aplicación.

    Nota: Esta configuración es única para todas las aplicaciones de la cuenta. Si utilizas la cuenta compartida del curso, es posible que otro alumno esté cambiando este valor a la vez que lo haces tú.

23. Introduce otros valores en Respuesta de licencia de prueba y verifica el resultado en tu aplicación. La lista de posibles respuestas y errores se muestra a continuación:

valor	Identificador	Descripción
256	`LICENSED`	La licencia es válida.
561	`NOT_LICENSED`	La licencia no es válida.
291	`RETRY`	No se ha podido conectar con el servidor. Se debería reintentar.

Tabla 8. Valores posibles del parámetro en los métodos allow(int) y dontAllow(int) de la interfaz LicenseCheckerCallback.

valor	Identificador	Descripción
1	`INVALID_PACKAGE_NAME`	Paquete no instalado.
2	`NON_MATCHING_UID`	Se pide un paquete distinto al de la app actual.
3	`NOT_MARKET_MANAGED`	Google Play desconoce el paquete.
4	`CHECK_IN_PROGRESS`	Una comprobación se está realizando. Solo se permite una comprobación a la vez.
5	`INVALID_PUBLIC_KEY`	La clave púbica es incorrecta.
6	`MISSING_PERMISSION`	Falta el permiso `com.android.vending.CHECK_LICENSE` .

Tabla 9. Valores posibles del parámetro en el método applicationError(int) de la interfaz LicenseCheckerCallback.

7.5.3. ¿Qué es una política de licencia?

El servicio de licencias de Google Play no determina directamente si un usuario dado con una licencia otorgada debe conceder acceso a la aplicación. Esta decisión se deja en manos de la política de licencia que queramos usar en nuestra aplicación.

Una política se implementa declarando en LVL un descendiente de la clase abstracta `Policy`. Esta clase está pensada para diseñar la lógica de la aplicación a la hora de permitir o no permitir el acceso del usuario, basándose en el resultado de una verificación de licencia.

La librería LVL incluye dos implementaciones de política de licencia que puedes utilizar directamente o adaptarlas a tus necesidades:

- **ServerManagedPolicy:** implementa una política flexible que almacena los resultados en una caché para usarlos en caso de no disponer de acceso a la red.
- **StrictPolicy:** no almacena en caché los datos de respuestas y permite el acceso solo si el servidor devuelve una respuesta con licencia.

Para la mayoría de las aplicaciones, se recomienda el uso de `ServerManagedPolicy` (es el valor predeterminado). Los datos de la última consulta se almacenan localmente en un fichero `SharedPreferences` ofuscado. Esta caché es implementada para permitir seguir usando la aplicación aunque perdamos el acceso a Internet[5].

Preguntas de repaso: El servicio de licencias Google Play.

7.6. Cómo evitar que se elimine la verificación de licencia en nuestras aplicaciones

Aunque usemos el servicio de licencias no estamos protegidos automáticamente contra usos no autorizados. Alguien podría decodificar nuestro código, quitar las líneas de comprobación de licencia y volverlo a compilar en una versión craqueada.

Algunos métodos que podemos utilizar se describen a continuación[6]:

Ofuscar el código: de esta forma sería más complejo comprender cómo se realiza la verificación de licencia.

No usar la librería LVL estándar: no utilices la implementación estándar de esta librería si no quieres que sea fácilmente descubierto como se realiza la verificación. Simplemente puedes cambiar el perfil de los métodos y reordenarlos para que sean más difíciles de localizar. También puedes poner el código entremezclado con el tuyo. Otra idea es no hacerlo siempre igual. En cada aplicación que publiquemos lo enrevesamos de una forma diferente. Haz actualizaciones a menudo de la aplicación y con cada actualización modifica los métodos de comprobación de licencia. De esta forma, el que quiera piratear tu aplicación tendrá que hacerlo con cada actualización.

Verificar que la aplicación no se ha modificado: verifica en la aplicación si se ha modificado el código. Puedes verificar la firma u obtener un código CRC del APK y compararlo con uno previamente obtenido.

Validar la licencia desde nuestro servidor: si la aplicación descarga contenidos en línea de nuestro servidor, podemos exigir que con la petición se adjunte

---

[5] Más información para implementar tu propia política en: http://developer.android.com/google/play/licensing/adding-licensing.html#impl-Policy

[6] http://android-developers.blogspot.com.es/2010/09/securing-android-lvl-applications.html

el identificador de usuario y de dispositivo. Con esta información podemos verificar desde nuestro servidor en el servidor de Google Play que el usuario tiene una licencia válida. En caso negativo no daremos los contenidos.

En los siguientes apartados veremos cómo aplicar las tres primeras técnicas con detenimiento.

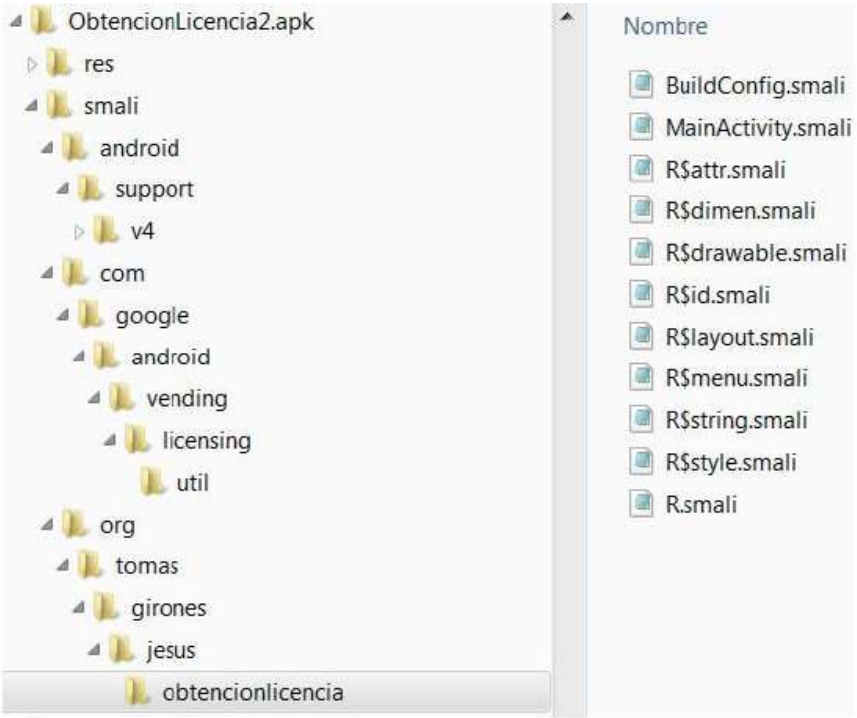
7.6.1. Ingeniería inversa en una aplicación con licencia

Veamos en el siguiente ejercicio cómo realizar la ingeniería inversa en la aplicación desarrollada en el punto anterior y luego la modificaremos para que no verifique la licencia. Vamos a trabajar con código Dalvik en notación smali. En el siguiente ejercicio aprenderemos más sobre este código.

Ejercicio: Ingeniería inversa en una aplicación con licencia.

En este ejercicio vamos a tratar de eliminar la comprobación de licencia de la aplicación desarrollada en el ejercicio anterior. Al no estar ofuscada podríamos decompilar el código a Java para estudiar mejor su contenido. Este paso nos lo vamos a saltar, dado que nosotros mismos hemos escrito este código.

1. Copia el fichero APK generado por el proyecto ObtencionLicencia dentro de una nueva carpeta. Ejecuta APK Studio y decompila la aplicación como se ha hecho en ejercicios anteriores.
2. Observa cómo se ha generado una carpeta con el nombre del apk. En ella puedes encontrar un fichero smali por cada una de las clases originales. Cada uno de estos ficheros estará en unas carpetas según el paquete al que pertenezcan. La estructura de carpetas creadas y los ficheros del paquete `org.tomas.girones.jesus.obtencionlicencia` se muestran a continuación:



3. Abre el fichero MainActivity.smali y compáralo con MainActivity.java del ejercicio anterior:

```
.class public Lorg/tomas/girones/jesus/obtencionlicencia/MainActivity;
.super Landroid/app/Activity;
.source "MainActivity.java"

# interfaces
.implements Lcom/google/android/vending/licensing/LicenseCheckerCallback;

# static fields
.field private static final CLAVE_PUBLICA_LICENCIA:Ljava/lang/String; =
.field private static final SALT:[B

# instance fields
.field comprobarLicencia:Lcom/google/android/vending/licensing/
                                                    LicenseChecker;
.field dialogo:Landroid/app/ProgressDialog;
.field permitir:Z

# direct methods
.method static constructor <clinit>()V
    .locals 1
    .prologue
    .line 18
    const/16 v0, 0x14
    new-array v0, v0, [B
    fill-array-data v0, :array_0
    sput-object v0, Lorg/ /obtencionlicencia/MainActivity;->SALT:[B
    .line 19
    return-void
    ...
```

Nota: Las líneas subrayadas desaparecerán en la versión ofuscada que veremos más adelante.

4. Busca el método `allow()` y compáralo con su equivalente en Java:

```
@0verride public void allow(int reason) {
   permitir = true;


# virtual methods
.method public allow(I)V
    .locals 3
    .parameter "reason"
    .prologue
    const/4 v2, 0x1
    .line 54
    iput-boolean v2, p0, Lorg/tomas/girones/jesus/obtencionlicencia/
                                              MainActivity; ->permitir:Z
```

Aunque no conozcamos en profundidad la notación smali, resulta fácil de interpretar. Las líneas que empiecen por `#` son comentarios. Luego se define el método público `allow()` que toma como parámetro un `int` (`I`) y devuelve `void` (`V`). La siguiente línea indica que va a utilizar tres variables locales. Se identificarán como los registros `v0`, `v1` y `v2`. Estos registros son siempre de 32 bits[7] y pueden almacenar un tipo simple o un puntero. Si el método recibe parámetros, estos se almacenan en los últimos registros. Todo método que no sea estático recibe como primer parámetro un puntero al objeto llamado (`this`). Este parámetro está implícito, no se indica en la lista de parámetros. En el método estudiado tendremos dos parámetros: `this`, que se almacenará en `v1`, y un entero, que se encuentra en `v2`. Para facilitar la lectura a estos registros se les asigna un nombre alternativo: `v1` = `p0` (primer parámetro) y `v2` = `p1` (segundo parámetro)[8][9].

Luego se indica que el primer parámetro en Java se llamaba `reason`. Esta es información de debug no relevante para ejecutar el código. Por fin llegamos a las instrucciones Dalvik[10]. La primera almacena la constante de 4 bits `0x1` en el registro `v2`. Los valores constantes se suelen expresar en hexadecimal, en este caso equivale a 1. La siguiente instrucción almacena el valor booleano que hay en `v2` en el campo `permitir` del objeto indicado en `p0`.

5. Analicemos la siguiente línea de Java:

---

[7] Un `bouble` de 64 bits se almacena en dos registros.

[8] Si en este método se hubieran pedido cuatro registros (`.locals 4`), el primer parámetro estaría en `v2=p0` y el segundo en `v3=p1`.

[9] Información sobre registros y parámetros: https://code.google.com/p/smali/wiki/Registers.

[10] Puedes encontrar una tabla con el significado de todas las instrucciones Dalvik en http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html.

```
    Toast.makeText(this,"Licencia correcta: "+reason, Toast.LENGTH_LONG)
                                                                .show();
```

Que ha sido compilada a:

```
.line 55
new-instance v0, Ljava/lang/StringBuilder;
const-string v1, ,"Licencia correcta: "
invoke-direct {v0, v1}, Ljava/lang/StringBuilder;-><init>
                                              (Ljava/lang/String;)V
invoke-virtual {v0, p1}, Ljava/lang/StringBuilder;->
                               append(I)Ljava/lang/StringBuilder;
move-result-object v0
invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()
                                                 Ljava/lang/String;
move-result-object v0
invoke-static {p0, v0, v2}, Landroid/widget/Toast;->
      makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
                                              Landroid/widget/Toast;
move-result-object v0
.line 56
invoke-virtual {v0}, Landroid/widget/Toast;->show()V
```

Creamos una nueva instancia de la clase `StringBuilder` que será apuntada por `v0`. Luego hacemos que `v1` apunte a la constante de cadena indicada. Invocamos al constructor (`<init>`) de `StringBuilder` pasándole dos parámetros: en `v0` el puntero al objeto a crear y en `v1` el objeto `String` para inicializarlo. Esta llamada no devuelve ningún resultado (`V`). Luego invocamos al método `StringBuilder.append(int)` pasándole dos parámetros: en `v0` el puntero al objeto y en `p1` el entero a concatenar, en este caso el parámetro con el que nos han llamado (`reason`). El resultado es un nuevo `StringBuilder` con la concatenación de este entero. En la siguiente instrucción movemos el resultado de la última invocación al registro `v0`. A continuación llamamos al método estático[11] `Toast.makeText(Context, CharSequence, int)` ,

[11] Observa como en este ejemplo aparecen tres tipos de invocaciones:

`invoke-virtual:` se utiliza para llamar a métodos de forma dinámica. En este tipo de llamada se indica al objeto el método y es él quien decide el método al que realmente se llama. Este tipo de llamada permite el polimorfismo de Java. Es decir, si en la llamada a `StringBuilder.append()` el objeto usado fuera un descendiente de esta clase y hubiera sobrescrito este método, se llamaría a un método diferente a `StringBuilder.append()`.

`invoke-direct:` se llama directamente al método, sin realizar una resolución dinámica. Se utiliza con constructores o cuando se indica `private` o `final` en el método.

`invoke-static:` se llama cuando se indica `static` en el método. Se hace una llamada directa, pero a diferencia de los casos anteriores no se indica el objeto involucrado.

pasándole tres parámetros[12]: `p0` con nuestra referencia (`this`), `v0` el `StringBuilder` antes creado (esta clase es descendiente de `CharSequence`) y `v2` con el valor 1 (`LENGTH_LONG`) . Este método devuelve un objeto `Toast` que es almacenado en la siguiente instrucción en `v0`. La última instrucción mostrada corresponde a la línea 56 de Java. En ella se llama al método virtual `Toast.show()` del objeto apuntado por `v0`.

6. Analicemos el final del método:

```
    dialogo.dismiss();
}
```

Que ha sido compilada a:

```
    .line 57
    iget-object v0, p0, Lorg/tomas/girones/jesus/obtencionlicencia/
                    MainActivity;-> dialogo:Landroid/app/ProgressDialog;
    invoke-virtual {v0}, Landroid/app/ProgressDialog;->dismiss()V
    .line 58
    return-void
.end method
```

La primera instrucción almacena en `v0` la referencia al objeto almacenado en el campo `dialog` del objeto apuntado por `p0`. También se indican las clases de los dos objetos apuntados. En la siguiente instrucción se llama al método `ProgressDialog.dismiss()` del objeto apuntado por `v0`. La última instrucción retorna del método sin devolver ningún parámetro.

7. Piensa las modificaciones que tendríamos que introducir en el código para que pudiéramos ejecutar la aplicación sin disponer de licencia. Una de las opciones más sencillas sería modificar el método `dontAllow()` para que ejecutara `permitir = true;` en lugar de `permitir = false;`. Veamos cómo se modificaría esta instrucción en los siguientes puntos.

8. Busca el método `dontAllow()` y compáralo con su equivalente en Java:

```
@0verride public void dontAllow(int reason) {
   permitir = false;
```

```
# virtual methods
.method public dontAllow(I)V
    .locals 2
    .parameter "reason"
    .prologue
    const/4 v0, 0x0
    .line 62
```

[12] Recuerda que en llamada a un método estático no se añade de forma implícita una referencia al objeto.

```
iput-boolean v0, p0, Lorg/tomas/girones/jesus/obtencionlicencia/
                                        MainActivity; ->permitir:Z
```

9. El valor subrayado corresponde a `false`. Para cambiarlo a `true` reemplázalo por el valor `0x1`. Guarda el fichero MainActivity.smali.
10. En APK Studio, recompila y firma la aplicación como se ha hecho en los puntos anteriores para generar un apk.
11. Instala este fichero en un dispositivo real y ejecútalo.
12. Entra dentro de Google Play Developer Console y en la izquierda selecciona la pestaña de Configuración ( ).
13. Modifica el valor de Respuesta de licencia de prueba a NOT_LICENSEC.
14. En la aplicación pulsa el botón «Comprobar Licencia». Verifica que no se obtiene licencia.
15. Pulsa el botón «Entrar». Verifica que se permite la entrada.

Práctica: Modificar otros aspectos de la aplicación de licencia.

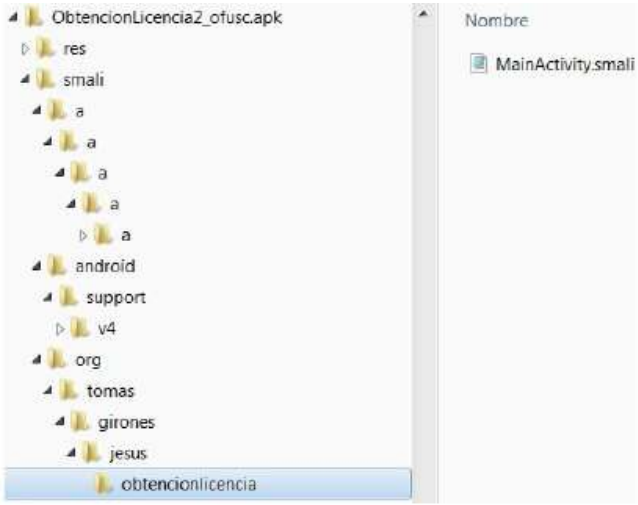
Trata de realizar otros cambios en la aplicación. Por ejemplo, que te permita entrar en la aplicación sin comprobar la licencia. También puedes eliminar el `Toast` que muestra el resultado negativo de la verificación de licencia.

7.6.2. Primera contramedida: ofuscar el código

Acabamos de ver lo sencillo que ha resultado eliminar el código en la aplicación de verificación de licencia. La primera acción a realizar para dificultar la ingeniería inversa de nuestra aplicación será ofuscar el código. Es muy fácil de hacer y evitamos que el código pueda transcribirse a Java, por lo que comprender el código será mucho más difícil.

Ejercicio: Ofuscación de la aplicación con licencia.

1. Ofusca la aplicación ObtencionLicencia. Para ello cambia a true el valor de `minifyEnabled` como se ha explicado en la sección de ofuscación de código. Luego exporta el APK firmado y ofuscado. Repite los puntos 1 al 4 del ejercicio anterior con este nuevo APK.
2. Observa como ahora, dentro de Projects, se ha creado una estructura de carpetas diferente:



La clase `org.tomas.girones.jesus.obtencionlicencia.MainAtivity` es una de las pocas que conserva su nombre. Esta clase está referenciada desde un AndroidManifest.xml por lo que no puede cambiar de nombre. El resto de las clases que antes pertenecían a este mismo paquete ahora están en `a.a.a.a.a` y se llaman `a`, `b`, `c`, `d`, `_` ,`u`.

3. Abre MainActivity.smali y compáralo con el obtenido en el ejercicio anterior:

```
.class public Lorg/tomas/girones/jesus/obtencionlicencia/MainActivity;
.super Landroid/app/Activity;

# interfaces
.implements La/a/a/a/a/m;

# static fields
.field private static final d:[B

# instance fields
.field a:La/a/a/a/a/i;
.field b:Z
.field c:Landroid/app/ProgressDialog;

# direct methods
.method static constructor <clinit>()V
    .locals 1
    const/16 v0, 0x14
    new-array v0, v0, [B
    fill-array-data v0, :array_0
    sput-object v0, Lorg/ /obtencionlicencia/MainActivity;->d:[B
    return-void
    ...
```

Observarás como han desaparecido algunas líneas (han sido subrayadas en el listado anterior). Observa cómo ha desaparecido el campo `CLAVE_PUBLICA_LICENCIA` . Esta constante se utiliza solo una vez, por lo que el ofuscador Proguard ha decidido usar directamente su valor, en vez de declararla. Puedes comprobarlo si miras dentro de `onCreate()` en la línea 256. De esta forma se ahorra código y disminuye la legibilidad. En la línea 256

también se aprecia como las cadenas de caracteres no son ofuscadas por Proguard. Otros ofuscadores más avanzados encriptan estas cadenas. Es interesante que observes como algunas referencias a clases externas han cambiado (los cambios se marcan en negrita y subrayado). Por ejemplo, `com/google/android/vending/licensing/LicenseCheckerCallback → a/a/a/a/a/m`. Sin embargo, otras clases no han cambiado, por ejemplo: `android/app/Activity` o `android/app/ProgressDialog`. No pueden cambiar cuando están definidas fuera de nuestra aplicación. Algo parecido ha pasado con el nombre de los métodos. Han cambiado `allow()→a()`, `applicationError()→b()` y `dontAllow()→c()`. Sin embargo, no han cambiado `comprobarLicencia()`, `entrar()` y `onCreate()`. Los dos primeros corresponden a eventos `onClick` que han sido definidos desde XML. Podrían ser ofuscados si se cambiara el nombre en los dos sitios; pero Proguard no realiza esta tarea. El método `onCreate()` no puede ser ofuscado al ser de la clase `Activity`, definida fuera de nuestra aplicación.

Práctica: Ingeniería inversa en una aplicación ofuscada.

Si tratas de obtener el código Java de un APK ofuscado, la herramienta jd-gui dará un resultado satisfactorio. Por lo tanto, tenemos que analizar el código a partir de los ficheros smali. El hecho de no disponer de los nombres de muchas de las clases, métodos y variables va a dificultar la tarea. ¿Qué estrategia seguirías para descubrir dónde se hace la comprobación de licencia?

Solución: Algunas posibilidades podrían ser:

- Buscar cadenas de caracteres relacionadas con la obtención de licencia.
- Buscar una interfaz que tenga un perfil similar a `LicenseCheckerCallback`. En concreto, visualiza el fichero a/a/a/a/a/m.smali:

```
.class public interface abstract La/a/a/a/a/m;
.super Ljava/lang/0bject;

# virtual methods
.method public abstract a(I)V
.end method

.method public abstract b(I)V
.end method

.method public abstract c(I)V
.end method
```

Está claro que una interfaz con tres métodos como los que se muestran tiene todo la pinta de ser la buscada. Lo que de momento no podemos saber es cuál de los métodos corresponde a `allow()`, `dontAllow()` o

`applicationError()`. Una vez descubierto, podrías localizar qué clase implementa esta interfaz y copiar el mismo código que se ha puesto en `allow()`, dentro de `dontAllow()`.

Práctica: Nuevas contramedidas para impedir la ingeniería inversa.

¿Qué contramedidas podrías implementar para impedir las acciones comentadas en la práctica anterior?

Solución:

Algunas posibilidades se describen en el siguiente apartado.

7.6.3. Segunda contramedida: no usar la librería LVL estándar

Ejercicio: Modificando la librería LVL.

En este ejercicio vamos a tratar de eliminar la comprobación de licencia de la aplicación desarrollada en el ejercicio anterior. Al no estar ofuscada podríamos decompilar el código a Java para estudiar mejor su contenido. Este paso nos lo vamos a saltar, dado que nosotros mismos hemos escrito este código.

1. Abre el proyecto ObtenciónLicencia.
2. Accede al paquete `com.google.android.vending.licensing` y edita la clase `LicenseCheckerCallback`.
3. Reemplaza la siguiente línea:

```
public void dontAllow(int reason);
```

Por:

```
public String dontAllow(String s, short reason);
```

No es necesario que cambies los identificadores; esto ya lo hará el ofuscador. Sería interesante cambiar también los otros métodos.

4. Pon este método como el primero de la interfaz. Proguard no reordena los métodos automáticamente. Si no lo hacemos nosotros siempre se realizará el mismo cambio de nombre: el primer método ➔ a(), el segundo ➔ b(), etc.

5. Añade otros métodos a la interfaz para despistar. Por ejemplo:

```
    public String noHaceNada(String s, short reason);
```

6. También es interesante cambiar los valores asignados a las constantes:

```
    public static final int ERROR_INVALID_PACKAGE_NAME = 1;
    public static final int ERROR_NON_MATCHING_UID = 2;
    public static final int ERROR_NOT_MARKET_MANAGED = 3;
    public static final int ERROR_CHECK_IN_PROGRESS = 4;
    public static final int ERROR_INVALID_PUBLIC_KEY = 5;
    public static final int ERROR_MISSING_PERMISSION = 6;
```

7. Guarda esta clase. Como es de suponer aparecerán errores en otras clases.
8. Edita la clase `LicenseChecker` y reemplaza la siguiente línea:

```
    validator.getCallback().dontAllow(Policy.RETRY);
```

Por:

```
    validator.getCallback().dontAllow("", (short) Policy.RETRY);
```

9. Edita la clase `LicenseValidator`; encontrarás dos errores. Igual que antes añade `"", (short)` al principio de los parámetros.
10. Edita la clase `MainActivity`; encontrarás dos errores. Para solucionar el segundo error reemplaza:

```
    @0verride public void dontAllow(int reason) {
```

Por:

```
    @0verride public String dontAllow(String s, short reason) {
```

Y añade al final de este método:

```
    return s+reason;
```

Para solucionar el primer error, pon el cursor del ratón encima del nombre de la clase; se desplegará un menú. Selecciona Add unimplemented methods.

11. Ejecuta la aplicación y comprueba que estos cambios no afectan en la verificación de licencia.
12. Ofusca y firma la aplicación utilizando la opción File/Export./Export Android Application. Se creará un fichero APK que has de copiar dentro de la carpeta place-apk-here-for-modding de APK Multi-Tool.
13. Con APK Studio decompila el apk.
14. Verifica que el nuevo fichero m.smali ha cambiado:

```
.class public interface abstract La/a/a/a/a/m;
.super Ljava/lang/0bject;

# virtual methods
.method public abstract a(Ljava/lang/String;)Ljava/lang/String;
```

```
.end method

.method public abstract a(Ljava/lang/String;S)Ljava/lang/String;
.end method

.method public abstract a(I)V
.end method

.method public abstract b(I)V
.end method
```

En el ejercicio anterior se han realizado cambios bastante básicos. Para aumentar la protección se podrían plantear cambios más profundos. Por ejemplo, reemplazar el patrón de diseño Observador por otro mecanismo de comunicación. Es decir, en lugar de definir estos métodos callback, resolver la comunicación de otra manera. Por ejemplo, podrías hacer la variable `permitir` de tipo estático y modificarla directamente en la clase `LicenseChecker` , en lugar de llamar a los callback. Otra posibilidad podría ser lanzar un anuncio broadcast cuando se verifique la licencia.

Almacenar si tenemos licencia en una variable booleana parece algo sencillo de descubrir. Sería más interesante si cuando verificamos la licencia cambiamos algún aspecto de un objeto, de forma que sin este cambio la aplicación no funcionará correctamente.

Las medidas realmente efectivas serían cambiar el funcionamiento interno de la librería LVL. Por ejemplo en [13] se propone una implementación alternativa del método `verify()` de esta librería.

Práctica: Modificando más aspectos de LVL.

Trata de implementar alguna de las medidas que se han comentado en la explicación anterior. También puedes proponer alternativas en los foros de este capítulo.

7.6.4. Tercera contramedida: verificar que no ha modificado nuestra APK

Parece que todo lo realizado hasta el momento no es suficiente. Como veremos en el siguiente ejercicio, las medidas implementadas hasta ahora no evitan que nos craqueen la aplicación con una herramienta automática.

[13] http://android-developers.blogspot.com.es/2010/09/securing-android-lvl-applications.html

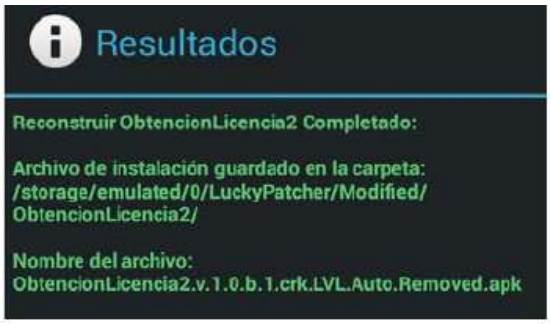
Ejercicio: Evitar la comprobación de licencia con Lucky Patcher.

Lucky Patcher es una herramienta para Android que permite evitar los anuncios, modificar permisos y saltarse la verificación de licencia de las aplicaciones instaladas. Aunque se trata de un proceso automático suele dar resultados satisfactorios con la mayoría de las aplicaciones.

1. Abre la web http://lucky-patcher.netbew.com desde un dispositivo móvil.
2. Ve a la sección Download y descarga la última versión de la aplicación. En este tutorial hemos usado la v3.2.6.
3. Instala la aplicación[14].
4. Al ejecutar esta herramienta te mostrará una lista con las aplicaciones encontradas que puede craquear.

5. Selecciona ObtencionLicencia. Te mostrará información detallada sobre la aplicación.
6. Selecciona el botón «Abrir menú de parches», luego «Crear archivo Apk modificado» y finalmente «Apk con verificación de licencia removida».
7. Aparecerá un menú con los tipos de parches disponibles. En este caso es suficiente con seleccionar «Modo Automático». Pulsa en «Reconstruir aplicación». Aparecerá la siguiente información:

[14] Esta acción puede resultar peligrosa. La hemos descargado de una fuente desconocida y además nos pide los permisos: editar o borrar contenido de USB, acceso completo a red y ejecutarse al inicio. Los datos almacenados en la memoria USB podrían estar comprometidos, por lo que sería interesante borrar la información comprometida que tengas en la memoria USB antes de instalarla. También es recomendable desinstalarla tras realizar el ejercicio.



8. Desinstala la aplicación ObtenciónLicencia e instala el nueva APK que acabas de crear.
9. Entra dentro de Google Play Developer Console y en la izquierda selecciona la pestaña de Configuración ( ).
10. Modifica el valor de Respuesta de licencia de prueba a NOT_LICENSEC.
11. En nuestra aplicación pulsa el botón «Comprobar Licencia». Verifica que se obtiene la licencia.

Las medidas implementadas en el punto anterior se han centrado en alterar la comunicación entre nuestra aplicación y la librería LVL. Seguramente es Lucky Patcher el que trate de localizar y modificar la comunicación entre LVL y el cliente Google Play. En este apartado vamos a utilizar un planteamiento diferente: vamos a tratar de verificar si la aplicación ha sido modificada.

Una de las alternativas para resolver esta cuestión es verificar si ha sido firmada con un certificado diferente al nuestro. Si alguien modifica nuestra aplicación no podrá usar nuestro certificado (solo nosotros tenemos la clave privada), por lo que tendrá que usar uno diferente.

Ejercicio: Verificar la firma de nuestra aplicación.

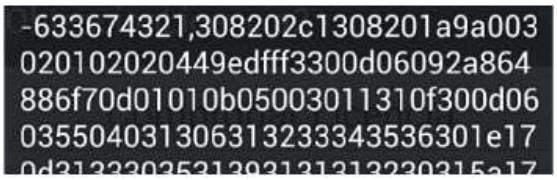
1. En el proyecto ObtencionLicencia añade el siguiente método a la clase `MainActivity`[15].

```
public boolean verificarFirma() {
    try {
        Signature[] firmas = getPackageManager().getPackageInfo(
            getPackageName(), PackageManager.GET_SIGNATURES).signatures;
        int i = firmas[0].hashCode();
        Toast.makeText(this, i+","+firmas[0].toCharsString(),
                                                Toast.LENGTH_LONG).show();
        return i == -633674321; //Reemplaza este valor
```

[15] Fuente: http://stackoverflow.com/questions/13445598/lucky-patcher-how-can-i-protect-from-it.

```
    } catch (NameNotFoundException e) {
        return false;
    }
}
```

Este método devolverá `true` en caso de comprobar que la firma es la original y `false` en caso contrario. Comenzamos utilizando el `PackageManager` para obtener información sobre el paquete de nuestra aplicación. En concreto le pedimos un array con las firmas de nuestra aplicación. La del certificado de la aplicación se almacena en la primera posición. La firma es muy larga, por lo que para manipularla de forma más cómoda vamos a obtener un código hash en un entero de 32 bits. En la siguiente línea se muestra en un `Toast` el código hash de la firma, seguido de una coma y de la firma en hexadecimal. Este `Toast` mostrará algo similar a:



A continuación, comprobamos que el código hash obtenido coincide con el nuestro y devolvemos el valor correspondiente. Cuando la aplicación esté firmada, tendrás que ejecutarla y poner el valor que aparezca en el `Toast` en sustitución del código hash que aparece en el código. El método termina capturando una excepción: si el nombre de nuestro paquete no es encontrado devolvemos que no podemos verificar la firma.

2. Modifica el método `entrar()` para que llame a la función anterior:

```
public void entrar(View view) {
   if (permitir) {
      if (verificarFirma()) {
         Toast.makeText(this, "Entrando en aplicación",
                                          Toast.LENGTH_LONG).show();
      } else {
         Toast.makeText(this, "Firma cambiada",Toast.LENGTH_LONG).show();
      }
   } else {
      Toast.makeText(this, "Licencia no válida",Toast.LENGTH_LONG).show();
   }
}
```

3. Firma la aplicación con un certificado y ejecuta en APK obtenido. Si obtienes la licencia y luego pulsas el botón «Entrar» te mostrará Firma cambiada.
4. Reemplaza el código hash de la firma como se ha explicado en el primer punto.
5. Firma la aplicación con el mismo certificado y ejecuta el APK obtenido. Ahora te ha de mostrar Entrando en aplicación.

6. Repite el ejercicio Evitar la comprobación de licencia con Lucky Patcher. Verifica que aunque sigue eludiendo la comprobación de licencia, esta nueva contramedida evita que se pueda entrar en la aplicación.

Otra alternativa para verificar que nuestra aplicación no ha sido modificada podría consistir en usar una función hash, como CRC32, para calcular el código hash del fichero APK de la aplicación. Puedes encontrar la ruta de los archivos de la aplicación llamando `context.GetApplicationInfo()` . Compara el resultado con el valor esperado. Por supuesto este código hash no puede almacenarse en el mismo código que estás usando para calcularlo. Puedes guardar este código en un archivo que no se utilice en este cálculo o en un servidor.

Preguntas de repaso: Evitar que pirateen nuestra aplicación.

Anexo A
El paquete camera2 de Android

Por ANTONIO ALBIOL

1.1. Introducción

El paquete android.hardware.camera2 proporciona un interfaz para acceder a las cámaras conectadas a un dispositivo Android. Está disponible a partir del nivel de API 21. Para niveles inferiores de API está disponible el paquete android.hardware.camera En este capítulo veremos cómo usar dicho paquete con el fin de:

Objetivos:

- Ser capaz de enumerar las cámaras presentes en un dispositivo Android.
- Ser capaz de conocer las características de las cámaras un dispositivo Android:
    - Modos de Enfoque
    - Modos de medida de exposición
    - Resoluciones de captura permitidas
- Ver cómo es posible iniciar la captura y previsualización de una cámara.
- Ver cómo poder capturar una imagen para analizarla cuando se realice una acción como tocar un botón.

- Ver cómo poder acceder a los píxeles de las imágenes capturadas de modo continuo.
- Ver cómo realizar un zoom digital.
- Ver cómo es posible detectar caras

1.2. Obtener información sobre las cámaras

En este apartado, veremos cómo construir una aplicación que permita obtener dicha información y la escriba en el logcat. Para ello crearemos una aplicación nueva haciendo File/New/New Project, seleccionando un **API mínimo de 21** y creando una actividad Vacía.

En `AndroidManifest.txt` añada las siguientes líneas para permitir que la aplicación pueda acceder a la cámara y almacenamiento externo.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
<uses-feature android:name="android.hardware.camera2.full" />
```

Añada las siguientes variables a la clase MainActivity:

```
private static final String TAG = "TestCapacidadesCamara2";
private static final String TAG2 = " *** CaractPrincipal";
```

Para enumerar las cámaras, averiguar sus propiedades o abrirlas, es necesario obtener una instancia de la clase `CameraManager`. Por tanto, el primer paso consiste en obtener un objeto de tipo `CameraManager`.

A partir del mismo será posible obtener una lista de todas las cámaras, y después, para cada una de ellas determinar qué características tiene. En el método `onResume()` escriba lo siguiente:

```
protected void onResume() {
    super.onResume();
    Log.e(TAG, "onResume");
    enumeraCamaras();
}
```

La función deberá tener de momento el siguiente código:

```
private void enumeraCamaras() {
    CameraManager manager =
         (CameraManager) getSystemService(Context.CAMERA_SERVICE);

    try {
        String[] cameras;
        cameras = manager.getCameraIdList();
        for (String id : cameras) {
            CameraCharacteristics characteristics =
```

```
                                    manager.getCameraCharacteristics(id);
            List <CameraCharacteristics.Key<?>> keys =
                                      characteristics.getKeys();
            for (CameraCharacteristics.Key<?> key :keys){
                String nombrecaracteristica = key.getName();
                Log.e(TAG, "Cámara :"+id +":"+nombrecaracteristica);
            }
        }
    } catch (CameraAccessException e) {
        Log.e(TAG, "No puedo obtener lista de cámaras");
        e.printStackTrace();
    }
}
```

Al ejecutar el programa, en el logcat deberá aparecer, para cada una de las cámaras del dispositivo, una relación de características de las mismas. Por ejemplo en el dispositivo en el que realizo las pruebas veo algo similar a :

```
Cámara : 0 : android.colorCorrection.availableAberrationModes
Cámara : 0 : android.control.aeAvailableAntibandingModes
Cámara : 0 : android.control.aeAvailableModes
Cámara : 0 : android.control.aeAvailableTargetFpsRanges
Cámara : 0 : android.control.aeCompensationRange
Cámara : 0 : android.control.aeCompensationStep
Cámara : 0 : android.control.afAvailableModes
Cámara : 0 : android.control.availableEffects
Cámara : 0 : android.control.availableSceneModes
...
```

Compruebe que aparece información para cada una de las cámaras de su dispositivo en el logcat.

1.2.1. Nivel de Hardware Soportado

El nivel de hardware soportado es una descripción de alto nivel de las capacidades de una cierta cámara. Resume varias capacidades en un único campo. El orden de los niveles es:

LEGACY: Dispositivos que funcionan en modo compatibilidad hacia atrás. Tiene unas capacidades muy limitadas.

LIMITED: estos dispositivos tienen el conjunto de características básicas y pueden también incluir algunas capacidades presentes en FULL.

FULL: Estos dispositivos permiten control manual a nivel de imagen del sensor, flash, lente y ajustes de postprocesado. Permiten capturar a una tasa de imágenes por segundo alta.

LEVEL_3: Dispositivos que soportan adicionalmente el reprocesado de imágenes YUV y captura de imágenes en modo RAW 1.

Las siguientes funciones permiten determinar el nivel el nivel de hardware soportado y verificar si un dispositivo soporta un cierto nivel de hardware.

```
void printNivelHardware(String cameraId,
                    CameraCharacteristics characteristics) {
    int nivel = characteristics.get(
           CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL);
    if( nivel ==
           CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY)
      Log.i(TAG2, "Camera " + cameraId + ": Nivel Hw Legacy");
    else
      Log.i(TAG2, "Camera " + cameraId + ": Nivel Hw =" +
                                              String.valueOf(nivel));
}

// Devuelve true si el dispositivo soporta el nivel de hw requerido u otro
mejor
boolean isHardwareLevelSupported(CameraCharacteristics c,
                                               int requiredLevel) {
  int deviceLevel =
              c.get(CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL);

  if(deviceLevel==
            CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY) {
    return requiredLevel == deviceLevel;
  }
  // deviceLevel no es LEGACY, se puede usar ordenamiento numérico
  return requiredLevel <= deviceLevel;
}
```

Añada en la función enumeraCamaras() una llamada dentro del bucle para sacar por el logcat el nivel de hardware soportado por cada cámara.

1.2.2. Orientación y posición de las cámaras

Una primera característica importante de las cámaras es su disposición en el dispositivo. Esto se puede conocer mediante la característica LENS_FACING. Se permiten 3 tipos de posibles resultados:

- Frontal, para la cámara situada en el mismo lado de la pantalla (cámara de selfies)

---

1 El modo RAW corresponde a los píxeles "crudos" tal cual los captura el sensor. Normalmente, las imágenes que vemos están "cocinadas", es decir se han aplicado diferentes métodos de reducción de ruido, compensación de contraluces, ... para que se vean bien. El modo RAW, permite que el usuario pueda realizar esos ajustes por su cuenta.

- Trasera, para la cámara situada en el lado opuesto de la pantalla; normalmente la cámara principal o de más calidad.
- Externa, cuando la cámara está conectada por ejemplo mediante un cable USB.

```
private void printCaracteristicasPrincipales(String cameraId, Camera-
Characteristics caracteristicas) {
  int lensfacing = caracteristicas.get(CameraCharacteristics.LENS_FACING);
  String lf = "Desconocida";
  if (lensfacing == LENS_FACING_FRONT)
    lf = "Frontal";
  else if (lensfacing == LENS_FACING_BACK)
    lf = "Trasera";
  else if (lensfacing == LENS_FACING_EXTERNAL)
    lf = "Externa";
  Log.i(TAG2, "Cámara " + cameraId + ": LENS FACING = " + lf);

  int orientation = caracteristi-
cas.get(CameraCharacteristics.SENSOR_ORIENTATION);
  Log.i(TAG2, "Camera " + cameraId + ": SENSOR ORIENTATION = " +
                                String.valueOf(orientation));
}
```

Otra característica importante es la **orientación del sensor**. Esta característica indica cuántos grados hay que rotar la imagen en sentido anti-horario para que la parte superior de la imagen aparezca en la parte superior de la pantalla del dispositivo en su orientación nativa. A este respecto conviene indicar que:

- La orientación nativa de los teléfonos es vertical.
- La orientación nativa de las tabletas es horizontal.

Tanto el sensor de la cámara como la pantalla, tienen un lado mayor y otro menor. Normalmente se alinean para que los lados largos de la cámara coincidan con los lados largos de la pantalla. No obstante, esto deja todavía un grado de libertad. En la figura siguiente el rectángulo rojo representa el sensor de la cámara. La parte amarilla, representa la parte superior de la imagen. Se muestra los valores que devuelve la orientación del sensor para cada tipo de dispositivo según la disposición del sensor de la cámara. No tener en cuenta la orientación del sensor en los casos 180 y 270, hace que cuando se visualizan las imágenes en la pantalla, aparezcan invertidas.

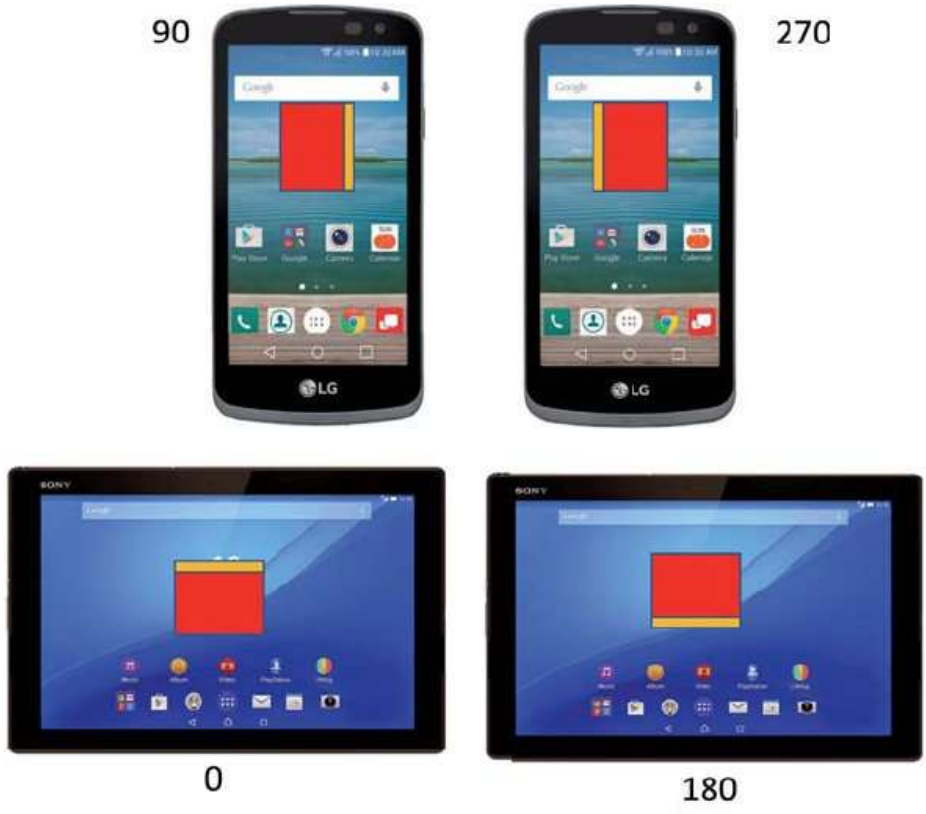


Figura 1 Posibilidades para la orientación del sensor.

Cree la anterior función, y realice una llamada a la misma dentro del bucle de la función `enumeraCamaras()`.

1.2.3. Métodos de enfoque y exposición

El enfoque y la exposición son quizá las características más críticas para obtener imágenes de calidad. Es importante conocer qué características soporta nuestra cámara. Para ello podemos relizar unas funciones que nos digan los valores posibles para nuestra cámara. Para los modos de enfoque:

```
private void printModosEnfoque(String cameraId,
                               CameraCharacteristics characteristics) {
  Log.i(TAG2, "MODOS ENFOQUE");
  int[] afmodes;
  afmodes = characteristics.get(
                      CameraCharacteristics.CONTROL_AF_AVAILABLE_MODES);
  for (int m : afmodes) {
    if (m == CaptureRequest.CONTROL_AF_MODE_AUTO)
      Log.i(TAG2, "Camera " + cameraId +
```

```
                                ": Auto Enfoque Básico disponible");
    if (m == CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_VIDEO)
      Log.i(TAG2, "Camera " + cameraId +
                          ": Auto Enfoque VIDEO continuo disponible");
    if (m == CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE)
      Log.i(TAG2, "Camera " + cameraId +
                        ": Auto Enfoque IMAGEN continuo disponible");
    if (m == CaptureRequest.CONTROL_AF_MODE_OFF) {
      Log.i(TAG2, "Camera " + cameraId + ": Enfoque manual disponible");
      if (isHardwareLevelSupported(characteristics,
          CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_LIMITED)) {
        float minimaDistanciaEnfoque = characteristics.get(
                  CameraCharacteristics.LENS_INFO_MINIMUM_FOCUS_DISTANCE);
        if (minimaDistanciaEnfoque > 0)
          Log.i(TAG2, "Camera " + cameraId +
                                ": Distancia Mínima de Enfoque = " +
                             String.valueOf(minimaDistanciaEnfoque));
        else
          Log.i(TAG2, "Camera " + cameraId + ": Foco Fijo ");
      }
    }
  }
  if (isMeteringAreaAFSupported(cameraId, characteristics)) {
    Log.i(TAG2, "Camera " + cameraId +
                               ": Regiones de autoenfoque soportadas");
  } else {
    Log.i(TAG2, "Camera " + cameraId +
                             ": Regiones de autoenfoque NO soportadas");
  }
}

private boolean isMeteringAreaAFSupported(String cameraId,
                                  CameraCharacteristics characteristics) {
  int numRegiones = characteristics.get(
                        CameraCharacteristics.CONTROL_MAX_REGIONS_AF);
  return numRegiones >= 1;
}
```

En el código anterior, también se ha incluido a modo de ejemplo cómo determinar la distancia mínima a la que puede enfocar una cámara si el nivel de hardware soportado lo permite. También se muestra como determinar el número de regiones de auto-enfoque. Si es mayor o igual que uno quiere decir que le podremos especificar en qué zona de la imagen deseamos enfocar. Esto es lo que sucede cuando se pulsa en un punto de la pantalla para indicar que esa es la zona en la que deseamos enfocar.

Para los modos de exposición:

```
  private void printModosExposicion(String cameraId, CameraCharacteristics
characteristics) {
  Log.i(TAG2, "MODOS EXPOSICION");
  int[]  aemodes;
  aemodes = characteris-
```

```
tics.get(CameraCharacteristics.CONTROL_AE_AVAILABLE_MODES);
  for (int m : aemodes) {
    if (m == CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH)
      Log.i(TAG2, "Camera " + cameraId +
        ": Control Automático Exposición con Flash automático disponible");
    if (m == CaptureRequest.CONTROL_AE_MODE_ON)
      Log.i(TAG2, "Camera " + cameraId +
          ": Control Automático Exposición con Flash apagado disponible");
    if (m == CaptureRequest.CONTROL_AE_MODE_ON_ALWAYS_FLASH)
      Log.i(TAG2, "Camera " + cameraId +
        ": Control Automático Exposición con Flash encendido disponible");
    if (m == CaptureRequest.CONTROL_AE_MODE_OFF)
      Log.i(TAG2, "Camera " + cameraId +
                      ": Ajuste Manual Exposición Disponible");
  }
}
```

Llame a ambas funciones dentro del bucle de `enumeraCamaras()` para obtener información tanto de los modos de enfoque como de los de exposición.

1.2.4. Tamaños de imagen

Otro aspecto importante es el de la resolución o tamaño de las imágenes a capturar. Las cámaras permiten un conjunto finito de posibles tamaños de imagen. Los tamaños disponibles dependen no sólo del número de píxeles del sensor sino del destino de las imágenes. Es posible obtener una lista de todos los tamaños que permite una cierta cámara para un cierto destino. Para ello escribiremos la siguiente función:

```
void printResolucionesCamara(String cameraId, CameraCharacteristics carac-
teristicas) {
  StreamConfigurationMap map = caracteristi-
cas.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
  assert map != null;
  Size[] dimensiones;
  dimensiones = map.getOutputSizes(SurfaceTexture.class);
  for (Size s : dimensiones) {
    Log.e(TAG, cameraId + " : Resolucion=" + s.toString());
  }
}
```

Añadiremos una llamada a dicha función desde dentro del bucle de la función `enumeraCamaras()`. Los valores de tamaño obtenidos son aquellos con los que es posible establecer un flujo en tiempo real de video entre la cámara y un objeto de tipo `SurfaceTexture`. Un poco más adelante veremos que este tipo de objetos son los usados para previsualizar la cámara en la pantalla del dispositivo.

Normalmente la primera resolución de la lista es la que corresponde con imágenes de mejor calidad (más resolución).

Abajo tiene un ejemplo del tipo de salida que se genera para la cámara trasera de un cierto tipo terminal:

```
CaractPrincipal: Camera 0: Nivel Hw Legacy
CaractPrincipal: Cámara 0: LENS FACING = Trasera
CaractPrincipal: Camera 0: SENSOR ORIENTATION = 90
0 : Resolucion=1440x1080
0 : Resolucion=1280x720
0 : Resolucion=1056x864
0 : Resolucion=960x720
0 : Resolucion=720x480
. . .
. . .
CaractPrincipal: MODOS  ENFOQUE
0: Auto Enfoque Básico disponible
0: Enfoque manual disponible
0: Auto Enfoque IMAGEN continuo disponible
0: Auto Enfoque VIDEO continuo disponible
CaractPrincipal: MODOS EXPOSICION
0: Control Automático Exposición con Flash apagado disponible
0: Control Automático Exposición con Flash automático disponible
0: Control Automático Exposición con Flash encendido disponible
```

Finalmente incluimos para mayor claridad cómo quedaría el código de la función enumeraCamaras() tras todas las llamadas necesarias.

```
private void enumeraCamaras() {
. . .
. . .
for (String id : cameras) {
  CameraCharacteristics characteristics = manager.getCameraCharacteristics(id);
  List<CameraCharacteristics.Key<?>> keys = characteristics.getKeys();
  for (CameraCharacteristics.Key<?> key : keys) {
    String nombrecaracteristica = key.getName();
    Log.e(TAG, "Cámara : " + id +  " : " + nombrecaracteristica);
  }
  printNivelHardware(id, characteristics);
  printCapabilities(id, characteristics);
  printCaracteristicasPrincipales(id, characteristics);
  printResolucionesCamara(id, characteristics);
  printModosEnfoque(id, characteristics);
  printModosExposicion(id, characteristics);
}
. . .
```

Ejercicio: Obtener información

Cree el programa anterior, y extraiga del logcat todas las informaciones mencionadas para cada una de las cámaras.

1.3. Arrancar la captura y visualizar

Una vez visto como conocer las características de nuestras cámaras veremos como iniciar la captura de imágenes en las mismas.

Para capturar imágenes desde una cámara, la aplicación debe crear primero una sesión de captura (`CameraCaptureSession`) con un conjunto de destinos (`Surfaces`) donde se depositarán las imágenes capturadas.

Cada `Surface` debe ser pre-configurada con el tamaño y formato adecuados de entre los soportados por la cámara. La `Surface` de destino puede ser obtenida a partir de una variedad de clases, incluyendo `SurfaceView`, `TextureView`, `SurfaceTexture`, `MediaCodec`, `MediaRecorder` e `ImageReader`.

En general, las imágenes de pre-visualización de la cámara se envían a objetos de clase `SurfaceView` o `TextureView`. La captura de imágenes estáticas se suele hacer con la clase `ImageReader` con formato **JPEG**. El análisis de flujos de vídeo en tiempo real se suele hacer con la clase `ImageReader` con formato YUV_420_888. Este formato es una transformación del formato conocido formato RGB (Rojo, verde, azul) en el que se tiene una primera componente Y, correspondiente a la luminancia (imagen monocroma) y otras dos, U=B-Y y V=R-Y. Las componentes de crominancia (U,V), son las responsables del color y tienen una resolución espacial mitad que la luminancia tanto en horizontal como verticalmente.

Una vez creada la sesión de de captura (`CaptureSession`), se tiene que crear una petición de captura (`CaptureRequest`) que contiene toda la información necesaria para que la cámara pueda capturar una imagen.

La petición de captura debe ser enviada a la sesión de captura:

- bien de forma única (por ejemplo para toma tomar una foto)
- bien de forma repetida para,por ejemplo para ofrecer una pre-visualización continua de donde apunta la cámara.

Se puede generar una petición única, mientras está en marcha una petición repetida (por ejemplo para tomar una foto mientras se está previsualizando la cámara). En ese caso, la previsualización se interrumpe momentáneamente para atender la petición única y se retoma tras finalizar la adquisición de la foto. En el ejemplo que veremos a continuación esto es lo que sucede.

Tras procesar una petición de catptura, la cámara produce un objeto de tipo `TotalCaptureResult` que contiene toda la información de la captura (hora de la captura, uso del flash, enfoque, ...). Las imágenes resultantes se depositan en todas las Surfaces previamente configuradas.

Veamos a continuación cómo crear una aplicación que permite pre-visualizar la cámara y que cuando se pulse un botón se guarde una imagen en formato JPEG. El programa que veremos es muy básico en cuanto a que no permite seleccionar la cámara ni el tamaño de la captura, pero combinándolo con lo visto en el apartado anterior sobre cómo seleccionar la cámara, los tamaños posibles de captura, etc. se podría crear una aplicación tan completa como se desee.

Ejercicio: Crear una aplicación para pre-visualizar la cámara

1. Crearemos una aplicación nueva llamada PreviewBasico haciendo nueva haciendo File/New/New Project, seleccionando un API mínimo de 21 y creando una actividad Vacía.

2. En AndroidManifest.txt añada las siguientes líneas para permitir que la aplicación pueda acceder a la cámara y almacenamiento externo.

```
android:minSdkVersion="21"
android:targetSdkVersion="23" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
<uses-feature android:name="android.hardware.camera2.full" />
```

3. En el archivo de layout `activity_main.xml`, añada una vista de tipo `TextureView`. Este tipo de vista es adecuado para recibir un flujo de imágenes y refrescar la visualización de la pantalla cada vez que se recibe una nueva imagen.

```
<TextureView
    android:id="@+id/textureView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

4. En el archive de estilos cambie el estilo de partida a:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
```

5. En el archivo `MainActivity.java`, añada algunas variables a la clase que luego necesitaremos:

```
private final String TAG = "PreviewBasico";
private TextureView textureview;
private String mCameraId;
private CameraDevice mCameraDevice;
private CameraCaptureSession mCaptureSession;
private CaptureRequest.Builder mPreviewRequestBuilder;
private Size dimensionesImagen;
 //* Thread adicional para ejecutar tareas que no bloqueen Int usuario.
private HandlerThread mBackgroundThread;
private Handler mBackgroundHandler;
```

6. Copie lo siguiente para el método `OnCreate()`:

```
protected void onCreate(Bundle savedInstanceState) {
  Log.i(TAG, "En onCreate !!!!");
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  textureview = (TextureView) findViewById(R.id.textureView);
  assert textureview != null;
}
```

7. En el método `onResume()` añadiremos un "escuchador" al textureView. La misión de este escuchador es responder a los eventos que se produzcan en el

`textureView`. Los eventos que resultan de más interés son `onSurfaceTextureAvailable` (cada vez que la vista pasa a ser visible), y `onSurfaceTextureUpdated` (cada vez que se actualiza la imagen). En el primer caso, abriremos el dispositivo de cámara (más abajo veremos cómo) y en el segundo, de momento, solo enviaremos un mensaje al logcat.

```
TextureView.SurfaceTextureListener textureListener = new Texture-
View.SurfaceTextureListener() {

  public void onSurfaceTextureAvailable(SurfaceTexture surface, int width,
int height) {
    //open your camera here
    Log.i(TAG, "Abriendo camara desde onSurfaceTextureAvailable");
    abrirCamara();
  }

  public void onSurfaceTextureSizeChanged(SurfaceTexture surface, int
width, int height) { }

  public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) {
    return false;
  }

  public void onSurfaceTextureUpdated(SurfaceTexture surface) { }
};

protected void onResume() {
  super.onResume();
  Log.e(TAG, "onResume");
  startBackgroundThread();
  Log.i(TAG, "Setting textureListener a textureview");
  textureview.setSurfaceTextureListener(textureListener);
}
```

8. Es necesario abrir un nuevo thread para que la captura de imágenes no obstaculice el interfaz de usuario de la aplicación. Para ello crearemos dos funciones una para arrancar el hilo y otra para pararlo.

```
protected void startBackgroundThread() {
  mBackgroundThread = new HandlerThread("Camera Background");
  mBackgroundThread.start();
  mBackgroundHandler = new Handler(mBackgroundThread.getLooper());
}

protected void stopBackgroundThread() {
  mBackgroundThread.quitSafely();
  try {
    mBackgroundThread.join();
    mBackgroundThread = null;
    mBackgroundHandler = null;
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
```

9. Seguidamente veremos el código para abrir la cámara. En este código, estamos cogiendo la primera cámara. Se podría modificar para elegir por ejemplo la cámara frontal o la trasera, haciendo un bucle sobre las cámaras, y eligiendo aquella que su característica LENS_FACING tuviera el valor deseado. Igualmente, como resolución de captura se ha tomado la primera (la de mayor resolución normalmente). Por último cabe mencionar, que **la aplicación debe tener los permisos de cámara activos**. Se podría incluir código que en caso de que no estuvieran activos, solicitara activarlos, pero por claridad se ha preferido omitir esta parte y generar una excepción en el caso de que los permisos no estén disponibles.

```
private void abrirCamara() {
  Log.i(TAG, "En abrir Camara");
  try {
    CameraManager manager = (CameraManager) getSystem-
Service(Context.CAMERA_SERVICE);
    mCameraId = manager.getCameraIdList()[0]; //La primera cámara
    CameraCharacteristics characteristics = manag-
er.getCameraCharacteristics(mCameraId);
    StreamConfigurationMap map = characteris-
tics.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
    assert map != null;
    Size tamanos[] = map.getOutputSizes(SurfaceTexture.class);
    for (Size tam : tamanos)
      dimensionesImagen = tam;

    Log.i(TAG, "Dimensiones Imagen =" +
String.valueOf(dimensionesImagen));
    manager.openCamera(mCameraId, stateCallback, null);
  } catch (CameraAccessException e) {
    e.printStackTrace();
  } catch (SecurityException e) {
    e.printStackTrace();
  }
}
```

10. Al abrir la cámara hay que pasarle un `stateCallback`. Esto no es más que un conjunto de métodos que se ejecutarán según el estado en que resulte la cámara tras la apertura del dispositivo. El método más intereseante es `onOpened()` que es el que se ejecuta una vez la cámara ha sido abierta . Añada el código siguiente antes del método `abrirCamara()`.

```
private final CameraDevice.StateCallback stateCallback =
                                  new CameraDevice.StateCallback() {

  public void onOpened(CameraDevice camera) {
    Log.e(TAG, "onOpened");
    mCameraDevice = camera;
    crearPreviewCamara();
  }
```

```
  public void onDisconnected(CameraDevice camera) {
    camera.close();
  }

  public void onError(CameraDevice camera, int error) {
    camera.close();
    mCameraDevice = null;
  }
};
```

11. Como vemos en el código anterior, si tras intentar abrir la cámara hay éxito se ejecutará el método `crearPreviewCamara()`. Es en este método donde va a tener lugar todo lo relativo a la captura propiamente dicha. Esencialmente en este método crearemos:

    - Una **sesión de captura** vinculada a la cámara. Tenemos que pasarle una lista de `Surfaces` donde dejar las imágenes. También se le pasa un callback que veremos más abajo, donde se ejecutarán distintas acciones dependiendo del resultado del intento de creación de la sesión de captura.
    - Una **petición de captura** vinculada a la cámara. En la creación, se le debe pasar una plantilla para que internamente se ajusten los parámetros necesarios de la misma. Por ejemplo cuando se le dice TEMPLATE_PREVIEW, se prioriza la velocidad en la adquisición frente a la calidad. Si se le dice TEMPLATE_STILL, lo que se le dice es que priorice la calidad frente a la velicidad. Una vez creada la petición de captura, se le debe indicar mediante el método `addTarget()` donde queremos que deposite las imágenes. Observe que el destino de las imágenes es la `Surface` de la clase `TextureView` que tenemos en el layout. Por tanto, las imágenes irán a la pantalla.
    - En la petición de captura, se configuran las opciones de exposición , enfoque, flash, etc.

```
private void crearPreviewCamara() {

  try {
    SurfaceTexture texture = textureview.getSurfaceTexture();
    assert texture != null;
    texture.setDefaultBufferSize(dimensionesimagen.getWidth(),
                                 dimensionesimagen.getHeight());
    Surface surface = new Surface(texture);
    mPreviewRequestBuilder =
        mCameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);
    mPreviewRequestBuilder.addTarget(surface);
    mPreviewRequestBuilder.set(CaptureRequest.CONTROL_MODE,
                               CameraMetadata.CONTROL_MODE_AUTO);
    mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
          CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_VIDEO);
```

```
    CameraCaptureSession.StateCallback statecallback =
         new CameraCaptureSession.StateCallback() {. . . };
     mCameraDevice.createCaptureSession(Arrays.asList(surface),
                       statecallback, null);

  } catch (CameraAccessException e) {
    e.printStackTrace();
  }
}
```

12. Veamos ahora el callback necesario para la apertura de la sesión de captura que se ha omitido antes por claridad. El método más interesante es `onConfigured()`. En dicho método, se llamará a la función `comenzarPreview()`, que será la que tras algunas comprobaciones arrancará la captura propiamente dicha. Reemplace la línea que empieza por "CameraCaptureSession.StateCallback" por:

```
CameraCaptureSession.StateCallback statecallback =
                  new CameraCaptureSession.StateCallback() {

  public void onConfigured(CameraCaptureSession cameraCaptureSession) {
    Log.i(TAG, "Sesión de captura configurada para preview");
    //The camera is cLosed
    if (null == mCameraDevice) {
      return;
    }
    // Cuando La sesion este Lista empezamos a visuaLizer imags
    mCaptureSession = cameraCaptureSession;
    comenzarPreview{);
  }


  public void onConfigureFailed(@NonNull CameraCaptureSession cameraCap-
tureSession) {
    Toast.makeText(MainActivity.this, "Configuration change failed",
Toast.LENGTH_SHORT).show();
  }
};
```

13. Como vemos, en el caso de que la sesión de captura se configure adecuadamente, se llama al método `comenzarPreview()`, que es el que pondrá en marcha la captura de imágenes mediante la llamada al método `setReapeatingRequest()` de la petición de captura.

```
protected void comenzarPreview() {
  if (null == mCameraDevice) {
    Log.e(TAG, "updatePreview error, return");
  }
  try {

    mCaptureSession.setRepeatingRequest(mPreviewRequestBuilder.build(),
```

```
                                null, mBackgroundHandler);
    Log.v(TAG, "*****setRepeatingRequest");
  } catch (CameraAccessException e) {
    e.printStackTrace();
  }
}
```

14. Prueba en el dispositivo: Llegados a este punto es el momento de probar la aplicación realizada. Para ello deberemos:

    - Bloquear el cambio automático de orientación del dispositivo. Cuando se cambia la orientación del dispositivo cambian las dimensiones de la Textura de destino de las imágenes. Por claridad, se ha preferido omitir el tratamiento de este caso en el código, por lo que bloqueando el cambio automático la orientación evitaremos problemas en la ejecución.
    - En versiones de Android mayores o iguales a la 6.0, será necesario activar los permisos de la cámara manualmente, yendo a Ajustes/Aplicaciones en el dispositivo. Igual que antes, se puede añadir código que pida el permiso si no está disponible, pero se ha preferido omitirlo por claridad.
    - Al ejecutar la aplicación veremos las imágenes de la cámara en la pantalla.

Ejercicio: Posibles mejoras en la aplicación para practicar

Con el fin de afianzar las ideas puede probar a:

- Elegir la cámara frontal (selfies) comprobando para todas las cámaras la característica LENS_FACING y abriendo la cámara frontal.
- Elegir de entre las resoluciones posibles de captura la menor de todas (última de la lista) y ver qué sucede en la pantalla. Las imágenes de la cámara
    - ¿Ocupan una parte de la pantalla?
    - ¿Aparecen pixeladas?
    - ¿Aparecen borrosas?

1.3.1. Capturando fotos

La aplicación que acabamos de presentar nos ha permitido adentrarnos en el API camera2 para la adquisición de imágenes. No obstante, la aplicación realmente no hace nada de utilidad aparte de mostrar las imágenes de la cámara en la pantalla. En este apartado veremos como modificar la aplicación anterior para tomar una imagen cuando se pulse un botón hacer “algo” con la imagen, en este caso guardarla en un archivo. Observe que lo que vamos a hacer podría ser el “esqueleto”

de una aplicación que analizara imágenes "bajo demanda", como por ejemplo un scanner de códigos QR.

Los pasos a realizar, para adquirir una imagen, mientras se está realizando una previsualización son los siguientes:

- Determinar la lista de tamaños posibles para la captura de una foto estática. Esta lista es diferente de la lista de tamaños de previsualización. Elegiremos uno de los tamaños posibles. En el ejemplo que haremos, tomaremos la mejor resolución posible de la cámara.
- Crearemos un objeto del tipo `ImageReader`, y lo configuraremos para el tamaño de captura seleccionado. Dicho objeto contiene un objeto de tipo `Surface` que es donde se dejarán las imágenes que capturemos al pulsar el botón.
- Creamos una petición de captura de acuerdo con la plantilla TEMPLATE_STILL_CAPTURE. Le indicaremos que el destino de las imágenes será el `Surface` del `ImageReader` anterior.
- Configuraremos los modos de enfoque y exposición para la petición de captura.
- Para el objeto `ImageReader`, configuraremos un escuchador que se encargará de guardar el archivo (o cualquier otra acción que pudiéramos desear) cuando se complete la captura de la imagen.
- Finalmente crearemos una sesión de captura. Dicha sesión de captura **interrumpirá la petición de captura repetida del preview** por ser de mayor prioridad Por tanto cuando se complete la sesión de captura será **necesario relanzar** la captura para la pre-visualización. Esto se hace en otro callback.

Veamos a continuación el detalle del código paso a paso.

Creación paso a paso de una aplicación que permite capturar fotos y guardarlas en archivos.

Tomaremos como punto de partida el programa que permitía pre-visualizar la cámara que acabamos de ver. Realizaremos una copia del mismo, y siguiendo las indicaciones de abajo haremos que tenga un botón que al ser pulsado permita capturar una imagen y almacenarla.

1. En el archivo de layout `activity_main.xml`, incluiremos un botón

```
<TextureView
    android:id="@+id/textureView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_above="@+id/btnCaptura" />

<Button
    android:id="@+id/btnCaptura"
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:text="CAPTURAR" />
```

2. En el archivo `MainActivity.java`, añadiremos nuevas variable a la clase

```
private Button btnCapture;
private CaptureRequest.Builder mPreviewRequestBuilder;
private CaptureRequest.Builder mJPEGRequestBuilder;
private ImageReader mImageReader;

private Size dimensionesPreview;
private Size dimensionesJPEG;
```

3. Al final del método `onCreate()`, añadiremos la llamada en respuesta a la pulsación del botón

```
btnCapture = (Button) findViewById(R.id.btnCaptura);
btnCapture.setOnClickListener(new View.OnClickListener() {
  public void onClick(View view) {
    tomarImagen();
  }
});
```

4. Cambiar el códido de la función `abrirCamara()` para determinar las resoluciones tanto de la pre-visualización como del ImageReader

```
private void abrirCamara() {
  try {
    CameraManager manager =
             (CameraManager) getSystemService(Context.CAMERA_SERVICE);
    mCameraid = manager.getCameraIdList()[0]; //La primera cámara
    CameraCharacteristics characteristics =
                      manager.getCameraCharacteristics(mCameraid);
    StreamConfigurationMap map = characteristics.get(
                  CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);

    assert map != null;
    dimensionesPreview = map.getOutputSizes(SurfaceTexture.class)[0];
//EL primer tamaño posible Normalmente el mayor
    dimensionesJPEG = map.getOutputSizes(ImageFormat.JPEG)[0];
//EL primer tamaño posible Normalmente el mayor

    Log.i(TAG, "Dimensiones Preview =" +
                                    String.valueOf(dimensionesPreview));
    Log.i(TAG, "Dimensiones JPEG =" + String.valueOf(dimensionesJPEG));

    manager.openCamera(mCameraid, stateCallback, null);
  } catch (CameraAccessException e) {
    e.printStackTrace();
  }
}
```

5. En la función `comenzarPreview()`, llamaremos a la función `configurarImageReader()` donde se creará la instancia, se le especificará las dimensiones y el formato. También se inicializará la petición de captura de la foto estática . Esta petición de captura se activará, como veremos más adelante, al pulsar el botón. Obsérvese que al crear la sesión de captura, debemos pasarle tanto la Surface para el preview como la que utilizaremos para capturar las fotos. Un poco más adelante hablaremos de la variable `readerListener` que aparece abajo.

```
private void comenzarPreview() {
  configurarImageReader();
  try {
    SurfaceTexture texture = textureview.getSurfaceTexture();
    assert texture != null;
    texture.setDefaultBufferSize(dimensionesPreview.getWidth(),
                                 dimensionesPreview.getHeight());
    Surface surface = new Surface(texture);
    mPreviewRequestBuilder = mCameraDevice.createCaptureRequest(
                                    CameraDevice.TEMPLATE_PREVIEW);
    mPreviewRequestBuilder.addTarget(surface);

    mCameraDevice.createCaptureSession(
                    Arrays.asList(surface, mimageReader.getSurface()),
                    cameraCaptureSessionStatecallback, null);
  } catch (CameraAccessException e) {
    e.printStackTrace();
  }
}

private void configurarImageReader() {
  try {
    mimageReader = ImageReader.newinstance(dimensionesJPEG.getWidth(),
                      dimensionesJPEG.getHeight(), ImageFormat.JPEG, 1);
    mimageReader.setOnImageAvailableListener(readerListener,
                                                    mBackgroundHandler);
    mJPEGRequestBuilder = mCameraDevice.createCaptureRequest(
                                     CameraDevice.TEMPLATE_STILL_CAPTURE);
    //DecirLe donde se dejarán Las imágenes
    mJPEGRequestBuilder.addTarget(mimageReader.getSurface());
  } catch (CameraAccessException e) {
    e.printStackTrace();
  }
}
```

6. Inserte el siguiente código en la clase, para generar el callback para la creación de la sesión de captura:

```
CameraCaptureSession.StateCallback cameraCaptureSessionStatecallback =
                              new CameraCaptureSession.StateCallback() {
  public void onConfigured(CameraCaptureSession cameraCaptureSession) {
    if (null == mCameraDevice) {
      return;
    }
```

```
    mCaptureSession = cameraCaptureSession;
    updatePreview();
  }
  public void onConfigureFailed(CameraCaptureSession camCaptureSession) {
    Toast.makeText(MainActivity.this, "Configuracion fallida",
                   Toast.LENGTH_SHORT).show();
  }
};
```

7. El código de la función updatePreview() es idéntico al de la pre-visualización básica. Lo reproducimos aquí por mayor facilidad:

```
protected void updatePreview() {
  if (null == mCameraDevice) {
    Log.e(TAG, "updatePreview error, return");
  }

  mPreviewRequestBuilder.set(CaptureRequest.CONTROL_MODE,
                             CameraMetadata.CONTROL_MODE_AUTO);
  mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
                  CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
  mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
                             CaptureRequest.CONTROL_AE_MODE_ON);
  try {
    mCaptureSession.setRepeatingRequest(mPreviewRequestBuilder.build(),
null, mBackgroundHandler);
    Log.i(TAG, "*****setRepeatingRequest. Captura preview arrancada");
  } catch (CameraAccessException e) {
    e.printStackTrace();
  }
}
```

8. Hasta este punto, tendríamos un programa que hace lo mismo que la pre-visualización básica. Al pulsar el botón de captura se llama a la función `tomarImagen()` que será la encargada de tomar la foto. Veamos el código de la función `tomarImagen()`. Vemos que lo que se hace es seleccionar los parámetros de enfoque y exposición y se llama al método `capture()` de la sesión de captura. Este método tiene más prioridad que el `setRepeatingRequest()` que empleamos para la pre-visualización, por lo que en cuanto se complete la siguiente imagen de pre-visualización, se abandonarán las peticiones repetidas y se atenderá el `capture()` asociado a la captura de un JPEG de alta calidad. Es interesante observar, que en el callback que se le pasa al método capture, se le está indicando que al finalizar la captura se debe volver a activar la pre-visualización. El código para `tomarImagen()` es:

```
private void tomarImagen() {
  try {
    mJPEGRequestBuilder.set(CaptureRequest.CONTROL_MODE,
                            CameraMetadata.CONTROL_MODE_AUTO);
    mJPEGRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
                    CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
    mJPEGRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
```

```
                                        CaptureRequest.CONTROL_AE_MODE_ON);

    CameraCaptureSession.CaptureCallback CaptureCallback
        = new CameraCaptureSession.CaptureCallback() {
      public void onCaptureCompleted(CameraCaptureSession session, Cap-
tureRequest request, TotalCaptureResult result) {
        comenzarPreview{);
      }
    };

    mCaptureSession.capture(mJPEGRequestBuilder.build(),
                                                    CaptureCallback, null);
  } catch (CameraAccessException e) {
    e.printStackTrace();
  }
}
```

9. El código que acabamos de ver ha capturado la foto, pero ¿donde está la foto? En la función `configurarImageReader` vimos que al `ImageReader` se le puede asignar un escuchador. Dicho escuchador es capaz de ejecutar código cuando el ImageReader correspondiente recibe una imagen ejecutando el método `onImageAvailable()`. Dentro de dicho método llamando a `acquireLatestImage` obtenemos la imagen, que es un objeto de tipo `Image`. Más adelante describiremos como acceder a los píxeles. En nuestro caso, por tratarse de una imagen comprimida (en la que no se puede acceder a los píxeles sin descomprimir previamente la imagen) nos limitaremos a guardar los bytes en un archivo llamando a la función guardar(). Un aspecto importante es que cuando hayamos terminado de hacer lo que queramos con la imagen debemos llamar al método close() porque si no no podríamos volver a capturar una segunda foto. Veamos como:

```
ImageReader.OnImageAvailableListener readerListener =
                      new ImageReader.OnImageAvailableListener() {
  public void onimageAvailable(ImageReader reader) {
    Image imagen = null;
    try {
      imagen = reader.acquireLatestImage();
      //Aquí se podría guardar o procesar La imagen
      ByteBuffer buffer = imagen.getPlanes()[0].getBuffer();
      byte[] bytes = new byte[buffer.capacity()];
      buffer.get(bytes);
      guardar(bytes);
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      if (imagen != null) {
        imagen.close();
      }
    }
  }
```

```
  private void guardar(byte[] bytes) throws IOException {
    final File fichero =
                new File(Environment.getExternalStorageDirectory() +
                                                    "/micamara2.jpg");
    Toast.makeText(MainActivity.this, "/micamara2.jpg saved",
                                        Toast.LENGTH_SHORT).show();
    OutputStream output = null;
    try {
      output = new FileOutputStream(fichero);
      output.write(bytes);
      output.close();
    } finally {
      if (null != output) {
        output.close();
      }
    }
  }
};
```

Compila el programa y ejecútalo en un dispositivo.

- Comprueba que al pulsar el botón, se captura la imagen.
- Compruebe que la resolución de la imagen capturada es la máxima ofrecida por el sensor de la cámara.

1.4. Analizando imágenes de forma continua

El programa que acabamos de ver, permitiría previsualizar las imágenes de forma contínua, y bajo la intervención del usuario lanzar una captura de alta resolución para disponer de una imagen que se podría analizar, transmitir, almacenar, etc. Supongamos que el objetivo es analizar de manera continua un flujo de imágenes. En este caso las imágenes que capture la cámara deberán ir simultáneamente a dos destintos:

- La pantalla
- Alguna función que haga "algo" con las imágenes.

En este apartado veremos cómo especificar un doble destino a las imágenes. Como el objetivo del apartado es aprender el proceso de captura, lo que haremos con las imágenes capturadas será especialmente sencillo. Sacaremos por el logcat el brillo del pixel central de la imagen capturada.

Ejercicio: Analizar imágenes de forma continua en tiempo real

Partiremos del programa generado en el apartado 1.3.

1. En la función `abrirCamara()` llamaremos a la función `configurarImageReader`, que será la encargada de determinar el tamaño de las imágenes que analizaremos.

```
private void abrirCamara() {
    try {
        CameraManager manager =
             (CameraManager) getSystemService(Context.CAMERA_SERVICE);
        mCameraid = manager.getCameraIdList()[0];
        CameraCharacteristics characteristics =
                             manager.getCameraCharacteristics(mCameraid);
        StreamConfigurationMap map = characteristics.get(
                    CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
        configurarimageReader(map);
         . . .
         . . .
```

2. En la función `configurarImageReader`, determinaremos el tamaño de las imágenes a analizar e indicaremos el **escuchador** (que describiremos más adelante) que se encargará de analizar las imágenes a medida que vayan estando disponibles. Para poder analizar las imágenes, no usaremos en este caso el formato comprimido JPEG sino el formato YUV_420_888. Este formato ya se ha descrito en otra parte del libro. Un aspecto importante es que al crear el ImageReader se le dota con **capacidad para dos** imágenes como podemos ver. La razón de esto se explica más adelante. El código sería:

```
void configurarImageReader(StreamConfigurationMap map) {
    Size tam = map.getOutputSizes(ImageFormat.YUV_420_888)[0];
    mimageReader = ImageReader.newinstance(tam.getWidth(),
                             tam.getHeight(), ImageFormat.YUV_420_888, 2);
    mimageReader.setOnImageAvailableListener( mOnimageAvailableListener,
                                                    mBackgroundHandler );
}
```

3. En la función crearPreviewCamara() que se invoca una vez la cámara ha sido abierta es donde le indicaremos los dos destinos para las imágenes. Para ello, cuando se crea la sesión de captura se le pasa una lista con dos Surfaces:

```
private void crearPreviewCamara() {
    try {
        SurfaceTexture texture = textureView.getSurfaceTexture();
        assert texture != null;
        texture.setDefaultBufferSize(dimensionesimagen.getWidth(),
                                     dimensionesimagen.getHeight());
        Surface surface = new Surface(texture);
        mPreviewRequestBuilder =
        mCameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);
        List surfaces = new ArrayList<>();
        mPreviewRequestBuilder.addTarget(surface);
        surfaces.add(surface);
        mPreviewRequestBuilder.addTarget(mimageReader.getSurface());
        surfaces.add(mimageReader.getSurface());

        CameraCaptureSession.StateCallback statecallback = new CameraCap-
tureSession.StateCallback(){
            public void onConfigured(
```

```
                            CameraCaptureSession cameraCaptureSession) {
                    Log.i(TAG,"Sesión de captura configurada para preview");
                    if (null == mCameraDevice) {
                        return;
                    }
                    mCaptureSession = cameraCaptureSession;
                    comenzarPreview();
                }
                public void onConfigureFailed(CameraCaptureSession cameraCap-
tureSession) {
                    Toast.makeText(MainActivity.this, "Configuration change
failed", Toast.LENGTH_SHORT).show();
                }
            };

            mCameraDevice.createCaptureSession(surfaces, statecallback, null);
        } catch (CameraAccessException e) {
            e.printStackTrace();
        }
}
```

4. El comienzo de la captura propiamente dicho se hace en la función `comenzarPreview()`:

```
protected void comenzarPreview() {
    if(null == mCameraDevice) {
        Log.e(TAG, "Error en comenzarPreview");
    }
    mPreviewRequestBuilder.set(CaptureRequest.CONTROL_MODE,
                                        CameraMetadata.CONTROL_MODE_AUTO);
    mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
                        CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
    try {
         mCaptureSession.setRepeatingRequest(
                mPreviewRequestBuilder.build(), null, mBackgroundHandler);
    } catch (CameraAccessException e) {
        e.printStackTrace();
    }
}
```

5. Como se puede ver, el código es muy similar al de le pre-visualización except por el hecho de que al crear la sesión de captura, le hemos pasado dos Surfaces en vez de una. Lo único que nos falta por ver es qué sucede con las imágenes cada vez que escuchador del `ImageReader` tiene una nueva imagen disponible. Vemos que lo único que se hace es llamar a una función que describiremos en el siguiente apartado y finalmente deberemos hacer un `close()` de la imagen para que el `ImageReader` pueda seguir recibiendo imágenes. De alguna manera, cuando llamamos a `acquireLatestImage`, **estamos tomando prestada la imagen**, de modo que mientras estemos haciendo algo con ella, la cámara no nos la sobre-escribirá. Por tanto, al finalizar con ella deberemos devolvérsela al ImageReader para que la cámara pueda seguir entregando imágenes. Para ser capaz de no perder imágenes el ImageReader debe disponer de dos imágenes.

- Una es la que yo estaré analizando
- Otra es la que la cámara estará llenando.

Siempre y cuando yo termine de analizar la imagen antes de que la cámara llene la otra imagen, seré capaz de procesar en tiempo real las imágenes sin perder ninguna. Si empleo mucho tiempo en el análisis de la imagen y la cámara termina de adquirir una imagen antes de que yo haya terminado mi análisis, la cámara se encontrará con que no tiene donde dejarle la imagen al reader y se esperará hasta que yo haga el close(). En ese caso el número de imágenes por segundo analizadas disminuirá.

```
private final ImageReader.OnImageAvailableListener
 mOnimageAvailableListener = new ImageReader.OnImageAvailableListener() {
 public void onImageAvailable(ImageReader reader) {
    Image imagen = reader.acquireLatestimage();
    if (imagen == null)
      return;
    procesarimagen(imagen);
    imagen.close();
  }
};
```

1.4.1. Accediendo a los pixels

En este apartado veremos como acceder a los pixels de una imagen en formato YUV_420_888. En este tipo de imágenes se tienen 3 planos de imagen:

- Y: luminancia, tiene la misma cantidad de píxeles que el tamaño de la imagen. Sus valores representan el brillo (nivel de gris) de los distintos píxeles.
- U,V: crominancia. Son dos planos que conjuntamente indican la tonalidad del color (rojo, azul, verde, ...). Para grises, tanto el valor de U como el de V. Una característica que tienen los planos U y V es que su resolución es la mitad que la luminancia.

Para poder acceder a los píxeles de una imagen, lo primero que debemos hacer es acceder a sus planos:

```
private void procesarImagen(Image imagen) {
  int width = imagen.getWidth();
  int height = imagen.getHeight();
  int format = imagen.getFormat();

  Image.Plane planes[] = imagen.getPlanes();
  Image.Plane yplane = planes[0];
  Image.Plane uplane = planes[1];
  Image.Plane vplane = planes[2];
```

La clase Image.Plane, representa un plano de la imagen como el Y, U o V.

Aunque conceptualmente una imagen es un rectángulo, sus píxeles están ordenados en memoria de forma lineal, primero por planos y luego por filas. Es decir,

primero vienen todos los píxeles de la primera fila, luego todos lo de la segunda fila y así sucesivamente. Para acceder a un píxel debemos calcular el offset (en bytes) desde el principio del plano para acceder al pixel de coordendadas (x,y). Para ello, necesitamos conocer dos atributos del plano:

- PixelStride : cuantos bytes hay que avanzar en memoria para acceder al pixel de la derecha de otro. En el caso del plano de luminancia esta cantidad siempre vale 1. En el caso de la crominancia, puede valer 1 o 2 dependiendo de si los valores de UV van entrelazados o primero todos los de U y luego todos los de V.
- RowStride : cuántos bytes hay que avanzar en memoria para alcanzar el pixel de abajo de otro. En el caso de la luminancia suele coincidir con la anchura de la imagen (si no hay relleno, es decir columnas invisibles a la derecha de la imagen). En el caso de la crominancia es la mitad del de la luminancia, pues tanto la U como la V tienen un tamaño mitad del de la Y.

```
private void procesarImagen(Image imagen) {
  int width = imagen.getWidth();
  int height = imagen.getHeight();
  int format = imagen.getFormat();

  Image.Plane planes[] = imagen.getPlanes();
  Image.Plane yplane = planes[0];
  Image.Plane uplane = planes[1];
  Image.Plane vplane = planes[2];

  int y_pixelstride = yplane.getPixelStride();
  int y_rowstride = yplane.getRowStride();
  int u_pixelstride = uplane.getPixelStride();
  int u_rowstride = uplane.getRowStride();

  Log.v(TAG, "Tengo una imagen NV21 !!" + String.valueOf(width) + "x" +
String.valueOf(height) );
  Log.v(TAG, "Tengo una imagen NV21 Stride PixelY=" +
String.valueOf(y_pixelstride) +
      " StridePixelUV=" + String.valueOf(u_pixelstride) +
      " StrideRowY=" + String.valueOf(y_rowstride) +
      " RowStrideUV=" + String.valueOf(u_rowstride));
```

Una vez determinados los valores de pixelstride y rowstride, ya podemos acceder al pixel que queramos. Para acceder al pixel de coordenadas (x,y) lo que debemos hacer es:

```
ByteBuffer yBuffer = imagen.getPlanes()[0].getBuffer();
byte pixelcentral = yBuffer.get(x * y_pixelstride + y * y_rowstride);
int pixel = pixelcentral & 0xFF;
```

En Java, cuando un byte se convierte a entero, se considera que el byte es un **entero de 8 bits con signo.** Para los valores de U como los de V esto es correcto, ya que tanto U como V pueden tomar valores positivos y negativos. Sin embargo, la luminancia siempre toma valores positivos. Por tanto si simplemente pusiéramos

```
pixel=pixelcentral
```

Cada vez que un pixel fuera mayor de 127, consideraría un valor negativo. Esa es la razón de escribir:

```
int pixel = pixelcentral & 0xFF;
```

Para terminar pondremos el código completo de la función procesarImagen().

```
private void procesarImagen(Image imagen) {
  int width = imagen.getWidth();
  int height = imagen.getHeight();
  Image.Plane planes[] = imagen.getPlanes();
  Image.Plane yplane = planes[0];
  Image.Plane uplane = planes[1];

  int y_pixelstride = yplane.getPixelStride();
  int y_rowstride = yplane.getRowStride();
  int u_pixelstride = uplane.getPixelStride();
  int u_rowstride = uplane.getRowStride();

  Log.v(TAG, "Tengo una imagen YUV420 !!" + String.valueOf(width) + "x" +
                                                  String.valueOf(height) );
  Log.v(TAG, "Tengo una imagen YUV420 Stride PixelY=" +
                                          String.valueOf(y_pixelstride) +
                       " StridePixelUV=" + String.valueOf(u_pixelstride) +
                           " StrideRowY=" + String.valueOf(y_rowstride) +
                          " RowStrideUV=" + String.valueOf(u_rowstride));

  int x = width / 2;
  int y = height / 2;

  ByteBuffer yBuffer = imagen.getPlanes()[0].getBuffer();
  byte pixelcentral = yBuffer.get(x * y_pixelstride + y * y_rowstride);
  int pixel = pixelcentral & 0xFF;

  Log.v(TAG, "Tengo una imagen YUV420. Pixel Central=" +
String.valueOf(pixel));
}
```

Prueba en el dispositivo

Al ejecutar el programa anterior en el dispositivo podremos observar en el logcat los valores de pixelstride y rowstride de los distintos planos y el nivel de intensidad del pixel central. Apuntando el teléfono hacia un lugar donde el pixel central sea claro, veremos que el valor que se indica en el logcat aumenta mientras que si se apunta hacia una zona oscura el valor disminuye.

1.4.2. Conclusión

En este apartado hemos visto las bases que permiten capturar-previsualizar-analizar en tiempo real de modo continua.

Hemos visto que el truco consiste en vincular dos Surfaces en la creación de la sesión de captura

Como el objetivo no era hacer sofisticados análisis de imagen, nos hemos limitado a mostrar cómo acceder a los píxels en Java introduciendo las ideas de pixelstride y rowsrtride.

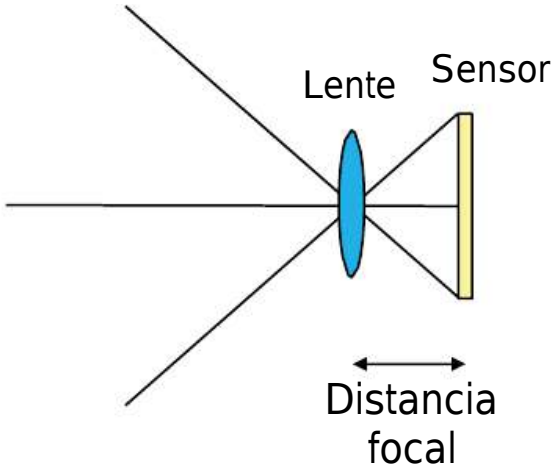
El análisis de imágenes a nivel de pixel no debería realizarse en Java por cuestiones de eficiencia. Una vez obtenida la imagen, para realizar análisis serios de imagen podríamos:

- Utilizar código nativo
- Utilizar librerías como OpenCV que vienen de compilar código nativo aunque se llamen desde Java.

Aunque OpenCV también permite capturar imágenes (usando el API Camera en vez de Camera2), el uso del API Camera2 directamente nos permitiría un control más fino de las muchas posibilidades que ofrece dicho API.

1.5. Zoom Digital

El ángulo del campo visual que capta una cámara depende del tamaño del sensor (en mm.) y de la distancia focal (distancia entre la lente y el sensor).



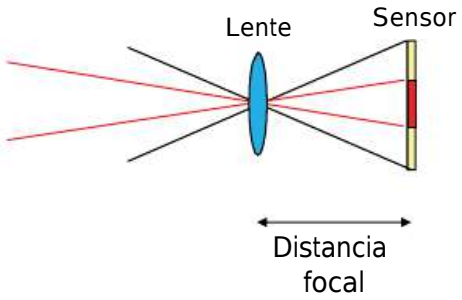
El ángulo del campo de visión viene dado por la fórmula (D: tamaño del sensor en mm, f: distancia focal en mm):

$$a = 2 \arctan \frac{D/2}{f}$$

Si se desea poder hacer zoom lo que hay que hacer es modificar la separación lente-sensor. Esto es lo que se conoce con el nombre de zoom óptico.

Normalmente, en la mayoría de teléfonos y tabletas, dicha distancia es fija. Por tanto en principio no se podría hacer zoom.

Otra alternativa para hacer zoom consistiría en reducir el tamaño del sensor. En realidad lo que se hace es usar solo una parte de los píxeles del mismo. Los sensores que llevan las cámaras normalmente tienen resoluciones mucho mayores que lo que luego se utiliza realmente. Por ejemplo, con un sensor de 13 megapíxeles, es posible que solo podemos grabar vídeos de 1920x1080 (aproximadamente 2 Mpíxeles). Para pasar de los píxeles del sensor a los píxeles finales las cámaras realizan un escalado. Siempre y cuando la cantidad de píxeles que usemos del sensor sigan siendo más que los del destino (en el ejemplo 1920x1080) no habrá pérdida de calidad. La figura siguiente ilustra la idea:



1.5.1. Determinación de las dimensiones del sensor y la distancia focal

Es posible determinar por programa las dimensiones del sensor de cámara y la distancia focal. Para ello cree la siguiente función en el programa determinaba información sobre las cámaras de la sección 1.2 y llámela:

```
private void printDimensionesSensor(String cameraId, CameraCharacteristics
characteristics) {
  SizeF dimensionesSensor =
     characteristics.get(CameraCharacteristics.SENSOR_INFO_PHYSICAL_SIZE);
  float distanciasFocales[] = characteristics.get(
                CameraCharacteristics.LENS_INFO_AVAILABLE_FOCAL_LENGTHS);
  Log.i(TAG2, "Camera " + cameraId + ": Tamaño Sensor = " +
                                     dimensionesSensor.toString() + "mm.");
  for (float f : distanciasFocales) {
    Log.i(TAG2, "Distancia Focal "+cameraId+String.valueOf(f)+"mm.");
  }
}
```

Ejecutándolo sobre la cámara trasera de un cierto teléfono se ha obtenido un tamaño del sensor 4.76x3.57 mm. y una distancia focal de 4.6mm. Aplicando la fórmula anterior, el campo visual sería:

Horizontal: 54.7º

Vertical: 42.41º

Ejercicio: Determinar el ángulo de visión de las cámaras

Para cada una de las cámaras, de su dispositivo determine:

- Dimensiones del sensor en mm. Normalmente el de la cámara trasera será mayor que el de la frontal.
- Distancia focal. Si el teléfono tiene la posibilidad de zoom óptico,
- Ángulo de visión horizontal y vertical.

1.5.2. Determinación del máximo zoom digital

Para realizar esta aplicación tomaremos como base la aplicación vista en el apartado 1.3 que realizaba la pre-visualización d las imágenes de la cámara. Incluya las siguientes variables en la clase

```
public float separacion = 0;
public double zoom_level = 1.0;
private int pixels_anchura_sensor;
private int pixels_altura_sensor;
float maxzoom;
Rect zoom;
```

Antes hemos explicado que si tomamos solo la mitad del tamaño del sensor, será equivalente a hacer un zoom x2. Además mientras la zona usada del sensor siga siendo mayor que el destino de las imágenes, no se perderá calidad. Existe un tamaño mínimo de la zona del sensor a usar (máximo zoom). Es posible determinar dicho valor por programa. También es posible determinar el número de píxeles del sendor. Para ello modificaremos la función abrirCamara() para que quede así:

```
private void abrirCamara() {
    try {
        CameraManager manager =
                (CameraManager) getSystemService(Context.CAMERA_SERVICE);
        mCameraid = manager.getCameraIdList()[0]; //La primera cámara
        CameraCharacteristics characteristics =
                             manager.getCameraCharacteristics(mCameraid);

        maxzoom = characteristics.get(
               CameraCharacteristics.SCALER_AVAILABLE_MAX_DIGITAL_ZOOM) ;
        Rect m = characteristics.get(
                    CameraCharacteristics.SENSOR_INFO_ACTIVE_ARRAY_SIZE);
        pixels_anchura_sensor = m.width();
        pixels_altura_sensor = m.height();

        StreamConfigurationMap map = characteristics.get(
                   CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
        assert map != null;
```

```
        dimensionesimagen = map.getOutputSizes(SurfaceTexture.class)[0];

        zoom_level = maxzoom;
        int width = (int)(pixels_anchura_sensor/zoom_level);
        int height = (int)(pixels_altura_sensor/zoom_level);
        int startx = (pixels_anchura_sensor - width)/2;
        int starty = (pixels_altura_sensor - height)/2;
        Rect zonaActiva = new Rect(startx, starty, startx + width,
starty+height);
        zoom = zonaActiva;
        Log.i(TAG, "Dimensiones imagen Preview =" +
String.valueOf(dimensionesimagen) +
                "Dimensiones Sensor: "+ m.toString() + "Maxzoom="
+String.valueOf(maxzoom) );

        manager.openCamera(mCameraid, stateCallback, null);
    }
    catch (CameraAccessException e) {
        e.printStackTrace();
    } catch (SecurityException e) {
        e.printStackTrace();
    }
    catch (NullPointerException e) {
        e.printStackTrace();
    }
}
```

La variable `zoom_level` contendrá el nivel de zoom activo. La inicializaremos al valor del máximo zoom posible, pero obviamente es posible fijarlo entre cualquier valor entre 1.0 y `maxzoom`. La variable `zoom` contendrá el rectángulo activo del sensor. El valor `maxzoom`, es el cociente entre la anchura (en pixels) del sensor y la anchura minima de la zona a usar. En un cierto teléfono, el valor que he obtenido ha sido de 6. Eso quiere decir que el máximo factor de ampliación será de x6.

Si el sensor tiene 4160x3120, la zona mínima que se podrá seleccionar será de 4160/6 = 694 por 3120/6= 520.

1.5.3. Seleccionando una zona útil del sensor

Una vez determinada la zona a emplear del sensor, que estará en la variable `zoom`, deberemos configurar la captura para que utilice dicha zona. Esto se hace en el mismo lugar donde indicamos los métodos de enfoque y exposición, en la función:

```
mPreviewRequestBuilder.set(CaptureRequest.CONTROL_MODE,
                                    CameraMetadata.CONTROL_MODE_AUTO);
mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
                        CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_VIDEO);
mPreviewRequestBuilder.set(CaptureRequest.SCALER_CROP_REGION, zoom);
```

En el código anterior, se ha fijado el factor de zoom al máximo posible

Finalmente hemos de indicarle, en la petición de captura dicha zona:

Dicho código se debe invocar desde la función crearPreviewCamara() vista en el apartado 1.3.

Esto es todo. Con estos pequeños cambios, seremos capaces de implementar un zoom digital simulando el efecto de un aumento de la distancia focal.

Ejercicio: Programa "Lupa"

Modificar el programa básico que lanzaba la captura de imágenes y las previsualizaba para que realice un zoom para ampliar al máximo la escena. Este programa puede ser útil para ayudar a leer textos pequeños.

Es curioso observar la piel de la mano desde aprox. unos 10cm. de la cámara.

OPCIONAL: Añada la capacidad de guardar la imagen que se está viendo al pulsar un botón o tocar la pantalla. Para ello se sugiere modificar ligeramente el programa del apartado 1.3.1 añadiendo el código que selecciona la zona activa del sensor.

1.5.4. Controlando el nivel de zoom mediante gestos

La aplicación anterior ajustaba el nivel de zoom al máximo posible. Sin embargo, podría interesar hacer ajustable el nivel de zoom mediante algún tipo de control deslizante, o algún gesto al que los usuarios estén acostumbrados. Normalmente separar dos dedos sobre la pantalla significa aumentar el zoom, mientras que juntarlos significa reducir zoom.

Ajuste el valor inicial de `zoom_level`. Alternativas razonables son 1.0, el valor `maxzoom`, `maxzoom/2.0`. Este ajuste debe hacerse en la función `abrirCamara()`.

Defina el código del escuchador para cuando se pulse la pantalla:

```
private View.OnTouchListener handleTouch = new View.OnTouchListener() {
    public boolean onTouch(View v, MotionEvent event) {
        int action = event.getAction();
        float sep_actual;
        if (event.getPointerCount() > 1) {// MuLti touch
            sep_actual = getFingerSpacing(event);
            if (separacion != 0) {
                if (sep_actual > separacion && maxzoom > zoom_level + 0.1)
{
                    zoom_level += 0.1;
                } else if (sep_actual < separacion && zoom_level >= 1.1) {
                    zoom_level -= 0.1;
                }
                int width = (int) (pixels_anchura_sensor / zoom_level);
                int height = (int) (pixels_altura_sensor / zoom_level);
                int startx = (pixels_anchura_sensor - width) / 2;
                int starty = (pixels_altura_sensor - height) / 2;
                Rect zonaActiva = new Rect(startx, starty,
```

```
                        startx + width, starty + height);
                mPreviewRequestBuilder.set(
                        CaptureRequest.SCALER_CROP_REGION, zonaActiva);
            }
            separacion = sep_actual;
        } else {
            separacion = 0;
        }
        try {
            mCaptureSession.setRepeatingRequest(
                 mPreviewRequestBuilder.build(), null, null);
        } catch (CameraAccessException e) {
            e.printStackTrace();
        } catch (NullPointerException ex) {
            ex.printStackTrace();
        }
        return true;
    }
};
```

Finalmente, en el método `onCreate()`, asigne el escuchador de eventos de toque a la `textureView` que muestra las imágenes:

```
textureview.setOnTouchListener(handleTouch);
```

Lea con detenimiento el código anterior para entender como se detectan los incrementos o decrementos de la separación entre dedos.

Ejecute el programa anterior y comprueba como es capaz de variar el nivel de zoom.

1.5.5. Conclusión

En este punto hemos visto como usar una de las muchas posibilidades que ofrece el API Camera2 para ajustar un zoom digital. Es interesante observar que usando OpenCV directamente no es posible realizar un zoom digital de calidad seleccionando una parte del sensor como zona activa.

1.6. Detección Facial

La detección facial, en imágenes presenta numerosas aplicaciones posibles. Algunas de ellas podrían ser el bloqueo automático cuando no hay una cara mirando el teléfono, ajustar la exposición o el enfoque a la zona de la cara, detección de gestos, etc.

El API camera2 de Android permite realizar de forma nativa las caras en una imagen, indicando la posición de la misma en imagen. A continuación veremos, paso a paso, como modificar el programa anterior para que detecte caras. Dado que el objetivo de este apartado es explicar cómo funciona la API de detección de caras, nos limitaremos a indicar el resultado de la detección en el logcat.

En primer lugar, modificaremos el programa para que emplee la cámara frontal. En la función `abrirCamara()` , modificar el código para coger la cámara frontal.

```
private void abrirCamara() {
    try {
        CameraManager manager =
              (CameraManager) getSystemService(Context.CAMERA_SERVICE);
        //mCameraid = manager getCameraidList()[eJ; //La primera  cámara
        //intentar buscar una cámara  frontaL
        String[] cameras = manager.getCameraIdList();
        for (String id : cameras) {
            CameraCharacteristics caracteristicas =
                                    manager.getCameraCharacteristics(id);
            int lensfacing =
                   caracteristicas.get(CameraCharacteristics.LENS_FACING);
            if (lensfacing == LENS_FACING_FRONT){
                mCameraid = id;
                break;
            }
        }
     . . .
```

A continuación, desde `abrirCamara()` llamaremos a una función que inicializará la detección de caras:

```
CameraCharacteristics characteristics=
                            manager.getCameraCharacteristics(mCameraId);
StreamConfigurationMap map = characteristics.get(
                  CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
inicializarCaras(characteristics);
```

Existen 2 modos posibles para detectar caras.

- El primero más básico, solo devuelve información sobre la presencia-ausencia de cara y las coordenadas de la cara en la imagen.
- El segundo, permite además localizar los ojos y boca, proporciona una medida de confianza en la detección de la cara, y además permite asignar un id a cada cara para realizar un seguimiento a lo largo del tiempo.

El código para determinar qué capacidades de detección de caras tenemos disponible es:

```
private void inicializarCaras(CameraCharacteristics characteristics) {
  int maxFD;
  int[] FD = characteristics.get(
      CameraCharacteristics.STATISTICS_INFO_AVAILABLE_FACE_DETECT_MODES);
  maxFD = characteristics.get(
                  CameraCharacteristics.STATISTICS_INFO_MAX_FACE_COUNT);
  if (FD.length > 0) {
    List<Integer> fdList = new ArrayList<>();
    for (int FaceD : FD) {
      fdList.add(FaceD);
```

```
      Log.d(TAG, "inicializarCaras: FD type:" + Integer.toString(FaceD));
    }
    Log.d(TAG, "inicializarCaras: FD count" + Integer.toString(maxFD));

    if (maxFD > 0) {
      mDeteccionFacialSoportada = true;
      mModoDeteccionFacial = Collections.max(fdList);
    }
  }
}
```

Crear en la clase las variables `mDeteccionFacialSoportada` y `mModoDeteccionFacial`.

```
boolean mDeteccionFacialSoportada;
int mModoDeteccionFacial;
```

En el método `comenzarPreview()`, insertar la llamada a la función que activará la detección facial:

```
mPreviewRequestBuilder.set(CaptureRequest.CONTROL_MODE,
                                        CameraMetadata.CONTROL_MODE_AUTO);
mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
                        CaptureRequest.CONTROL_AF_MODE_CONTiNUOUS_PiCTURE);
setFaceDetect(mPreviewRequestBuilder, mModoDeteccionFacial);
```

El código de la función es:

```
private void setFaceDetect(CaptureRequest.Builder requestBuild,
 int fDMode) {
   if (mFaceDetectSupported) {
     requestBuild.set(CaptureRequest.STATISTICS_FACE_DETECT_MODE,fDMode);
   }
}
```

Con el código anterior, las caras se detectarán en cada imagen del preview. Para ver que realmente estamos haciendo algo vamos a añadir un código que nos saque por el logcat información de si se está detectando cara o no. Para ello en el método comenzarPreview, cuando se lance la petición de capturas repetidas, introduciremos un callback:

```
try {
  mCaptureSession.setRepeatingRequest(mPreviewRequestBuilder.build(),
capture_callback, mBackgroundHandler);
}
```

Finalmente, dentro del método actualizarPreview(), crearemos un callback de cap-tura, que contendrá las acciones a realizar cada vez que se complete una captura. Como se puede ver, lo único que hace es mirar a ver si se ha detectado alguna cara:

```
CameraCaptureSession.CaptureCallback capture_callback =
                               new CameraCaptureSession.CaptureCallback() {

  public void onCaptureStarted(CameraCaptureSession session,
```

```
                CaptureRequest request, long timestamp, long frameNumber) {
    super.onCaptureStarted(session, request, timestamp, frameNumber);
  }

  public void onCaptureCompleted(CameraCaptureSession session,
                      CaptureRequest request, TotalCaptureResult result) {
    super.onCaptureCompleted(session, request, result);
    process(result);
  }

private void process(CaptureResult result) {
    Face faces[] = result.get(CaptureResult.STATISTICS_FACES);
    if (faces.length > 0) {
      for (Face f : faces) {
        Rect limites = f.getBounds();
        Log.d(TAG, "Cara detectada Limites:" + limites.toString() );
        } //for
    } //if
  } //process
};
```

Finalmente solo nos faltará probar el programa en nuestro dispositivo. Deberemos observar que:

- la cámara usada es la frontal.
- Si miramos en el logcat, observaremos:
  - un mensaje indicando que es posible la detección facial.
  - Mensajes de "Cara Detectada" cada vez que se detecte una cara.
  - Se indica también la posición de la cara en la imagen.

En algunos dispositivos que lo soporten, también es posible que devuelva las coordenadas de los ojos y la boca. Para ello deberemos modificar la función process para que lo indique:

```
private void process(CaptureResult result) {
  Face faces[] = result.get(CaptureResult.STATISTICS_FACES);
  if (faces.length > 0) {
    for (Face f : faces) {
      Rect limites = f.getBounds();
      int score = f.getScore();
      int id = f.getId();
      Point boca = f.getMouthPosition();
      Point ojoizq = f.getLeftEyePosition();
      Point ojoder = f.getRightEyePosition();

      String st = f.toString();
      if (boca == null 11 ojoizq == null 11 ojoder == null) {
        Log.d(TAG, "id:" + Integer.toString(id) + " Limites:" +
```