Universidad de Alcalá

Escuela Politécnica Superior
Grado de ingeniería de Computadores

Proyecto Fin de Carrera

Sistema de seguimiento de objetos mediante agrupación de trayectorias en terminales móviles

Autor: Christian Sánchez Montero
Director: Pedro Gil Jiménez

TRIBUNAL:

Presidente: .....

Vocal 1º: .....

Vocal 2º: .....

CALIFICACIÓN:..... FECHA:.....

Sistema de seguimiento de objetos mediante agrupacion de trayectorias en terminales móviles

Christian Sánchez Montero

29 de junio de 2015

Resumen

Este proyecto pretende la realización de un sistema de seguimiento de objetos mediante agrupación de trayectorias en terminales móviles. Para ello en este proyecto habrá que acceder a la cámara de un terminal móvil, así como al flujo de video generado por este terminal. El procesamiento se basará en recoger puntos de interés, realizar la detección de trayectorias y objetos dentro de estos, con fin de calcular la densidad de tráfico mediante una aplicación para terminales móviles embarcados en vehículos. Este procesado se realiza mediante una librería que se ha facilitado y el trabajo es el poder adaptar todo el sistema para que tenga su funcionamiento en un terminal móvil con sistema operativo Android.

Todo ello programado en lenguaje Java para Android, con las librerías de OpenCV, para tratamiento y procesado de imágenes.

Abstract

This project aims at the realization of a system for tracking objects by aggregation of paths in mobile terminals. To do this, you need to access the camera from a mobile terminal and also the video stream generated by this terminal in order to process each frame that the camera gives us. Processing will be based on collecting points of interest, making the detecting trajectories and objects inside of them, with the purpose of calculating the density of traffic by using a terminal application mobile board of vehicles. This processing is performed by a library that has been provided and the work is adapting it to the whole system to have its operation in a mobile terminal with Android operating system.

All of this has been programmed in Java for Android, with OpenCV libraries for processing and image processing.

Resumen Extendido

La congestión del tráfico es hoy en día uno de los problemas más grandes en los países desarrollados. Esto es por el gran número de vehículos que utilizan un conjunto limitado de infraestructuras, como pueden ser la vías urbanas e interurbanas. Este problema, dado que es uno de los que más ha preocupado en los últimos años, surge un gran interés en el desarrollo de un sistema o programa que ayude en cierto modo a optimizar la congestión del tráfico, permitiéndonos optimizar una ruta mejorando el tiempo que se utiliza en el desplazamiento.

Este sistema o programa debe de poder informar y recoger información de las vías urbanas e interurbanas, y por tanto sabremos el estado de la densidad del tráfico y congestión. También hay que pensar en la planificación de una ruta óptima, donde influyen muchos factores, como la densidad del tráfico, la velocidad media de la vía, tipo de carreteras y conexiones entre ellas etc...

El problema de estos programas es el rendimiento que nos aporta, estos sistemas tienen que ser rápidos y precisos por lo que el programa necesita un buen tiempo de reacción. Para un buen funcionamiento se debe de informar del estado actual del tráfico en tiempo real.

Por otro lado, hay que tener en cuenta los Smartphones, o terminales móviles inteligentes. Ésta tecnología cuenta con una capacidad de procesado y calidad de imágenes más que aceptables, además de que ya la poseen un gran número de conductores usuarios. Los Smartphones son muy útiles ya que aportan ciertas prestaciones que pueden ayudar a la circulación, un ejemplo es el GPS. Este GPS, junto a la cámara de vídeo integrada en el terminal móvil, nos podría permitir obtener la información en tiempo real del tráfico.

El proyecto propone el desarrollo de una aplicación móvil de procesado de video mediante la cámara integrada en el terminal, y así poder obtener la información de la densidad del tráfico. Este sistema es una parte de un proyecto aún más completo.

Por tanto, la aplicación se centrará exclusivamente en técnicas de procesado de imágenes de vídeo, que son captadas por el terminal. Éstas técnicas de procesado, vienen dadas por una librería que se proporciona, el proyecto se centrará en poder desarrollar una aplicación para terminales móviles, adaptando esta librería de procesado.

Antes de desarrollar la aplicación hay que decidir en qué sistema operativo se va a desarrollar en estos terminales. El sistema escogido en Android, dado que es un sistema operativo

que cuenta con una cuota de mercado muy amplia, es decir que habrá muchos conductores usuarios con la posibilidad de tener un terminal móvil con sistema Android. Además Android es un sistema operativo libre y gratuito.

Un ejemplo del mercado que tiene Android en España es el siguiente:

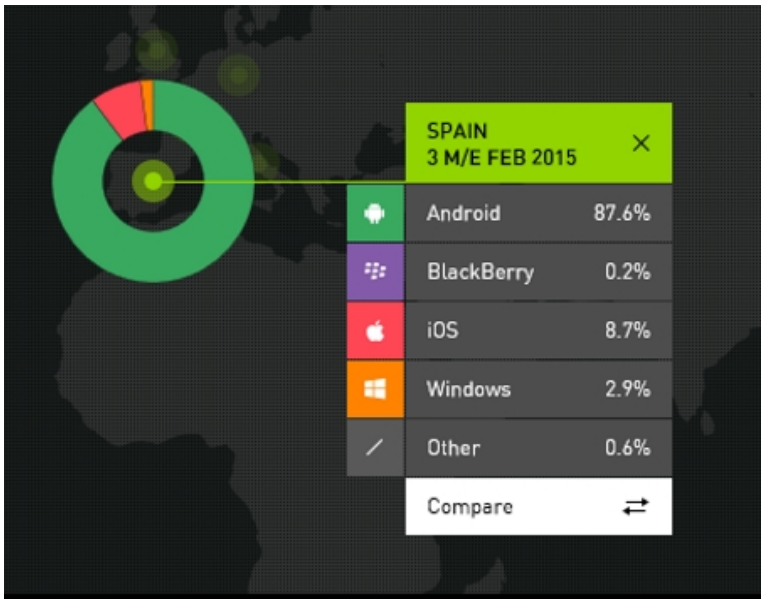


Figura 1: Cuota de Android 2015 en España

Android nos permite acceder a las funciones del teléfono, ésta aplicación necesita obligatoriamente hacer uso de la cámara del terminal móvil, así como al flujo de vídeo generado. Para el desarrollo de esta aplicación ha sido necesario la inclusión de la librería de procesado dada, así como la posibilidad de utilizar OpenCV para esta aplicación. OpenCV es una librería que está diseñada para el procesado y tratamiento de imágenes. Con OpenCV se accederá a la cámara del terminal fácilmente, además de darnos el flujo de video, y con la librería que se proporciona, se podrá procesar las imágenes adquiridas y poder informar de la densidad del tráfico, que es la finalidad del proyecto.

La librería proporcionada, ésta programada en Java, y la aplicación a desarrollar en Android también es en Java, por lo tanto será fácil poder incluir la librería en el proyecto y acceder a las funciones que nos permite. Un ejemplo del procesado que puede hacer la librería esta en la figura 2.

Figura 2: Imagen procesada por la librería

El desarrollo de ésta aplicación se ha realizado en un entorno de desarrollo llamado Android Studio. Es un entorno que facilita la creación de aplicaciones en Android.

El sistema ha consistido en crear una interfaz gráfica sencilla para el acceso a cualquiera de las funciones que proporciona la librería. Ésta interfaz se ha desarrollado pensando que ésta librería en un futuro puede ser ampliada, de modo que las opciones de la interfaz se crean dinámicamente en función de la librería, como por ejemplo dependiendo del número de vistas que tenga la librería o el número de parámetros. Pudiendo así que la aplicación siga en funcionamiento a pesar de cambios en la librería.

Índice general

Capítulo 1

Introducción.

Hoy en día la congestión del tráfico es uno de los problemas de transporte principales. Con la gran cantidad de vehículos que hay, siendo más en países o ciudades desarrolladas. Entonces, debido a esto surge la necesidad de sistemas que puedan gestionar el tráfico, optimizar el uso de las vías urbanas e interurbanas.

Para ésto se necesita un programa que informe de la densidad del tráfico, un problema puede ser el rendimiento de éstos programas, que tienen que ser muy precisos y intentar dar una estimación en tiempo real.

En los últimos años, hay un gran número de terminales móviles inteligentes (Smartphones) que existe entre los conductores usuarios, que se utilizan por ejemplo para navegadores GPS. Estos terminales, en la actualidad ofrecen capacidades de procesado más que aceptables.

Dado el gran número de terminales entre los conductores, y que estos tienen sistemas de navegación GPS, cámara de video con una calidad considerable, y buena capacidad de procesado, se puede usar ésta tecnología para poder obtener una visión del estado del tráfico o densidad de éste.

Se va a desarrollar una aplicación en Android que permita poder procesar imágenes de la cámara, gracias a la librería de procesado que se ha dado. El trabajo del desarrollo ha sido el de poder adaptar ésta librería, junto al la librería de OpenCV, para el procesado y tratamiento de imágenes. Adaptando todo esto a la aplicación Android para que el usuario puede ver y obtener información útil respecto a la densidad y congestión del tráfico.

Para el desarrollo del proyecto se ha realizado en lenguaje de programación Java, ya que facilita la programación en Android, y la librería que se ha facilitado también está programada en Java y se hace más fácil la utilización de sus funciones.

1.1. Android

1.1.1. ¿Qué es Android?

Android es un sistema operativo inicialmente pensado para teléfonos móviles. Está basado en Linux, un núcleo de sistema operativo libre, gratuito y multiplataforma. El sistema permite programar aplicaciones en una variación de Java. El sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, la cámara, etc.) de una forma muy sencilla en un lenguaje de programación muy conocido como es Java.
La arquitectura de Android ésta dividida en varias capas.

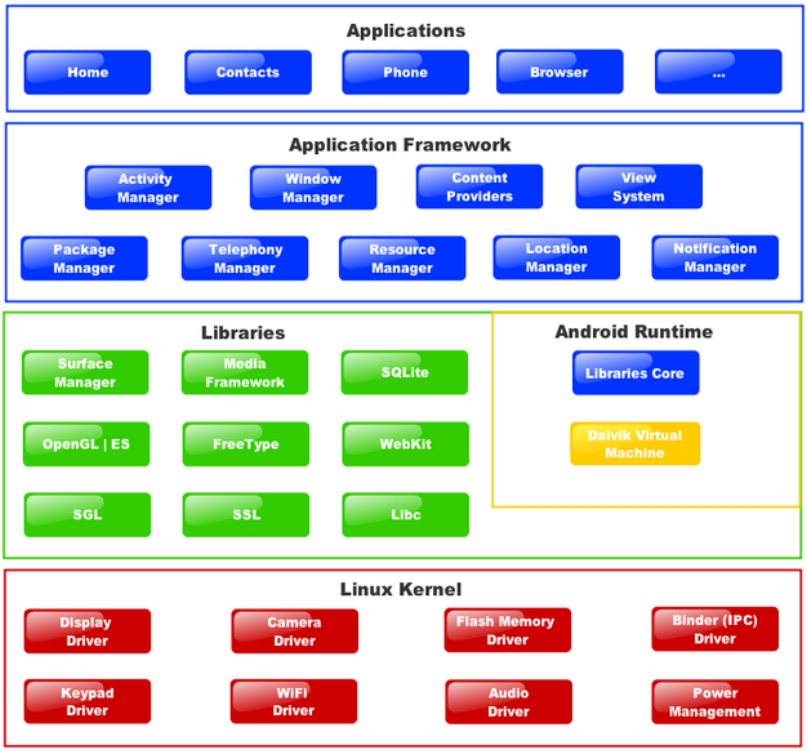


Figura 1.1: Arquitectura Android

Cada una de estas capas utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos propios a las capas de niveles superiores (Figura 1.1).

1.1.2. Programación en Android del proyecto

Para la programación en Android se ha utilizado un entorno de desarrollo llamado Android Studio, es un entorno específico para desarrollar las aplicaciones de Android.

Para crear aplicaciones en Android es muy importante un archivo llamado *manifest*, éste archivo tiene todos los recursos que va a necesitar la aplicación a desarrollar, también el *activity* principal, como los demás *activitys* a ejecutar en el programa. En caso de éste proyecto uno de los recursos es la cámara.

Otros elementos importantes son los *layouts*, que nos permite montar la interfaz gráfica de la aplicación de forma sencilla, y una vez construida hay que enlazar los elementos del *layout* en el *Activity*.

En Android es muy importante los eventos a los que están asociados los elementos que se vayan a utilizar en la aplicación. Uno de los eventos más importantes de los que vamos a utilizar en éste proyecto es el evento de la cámara, éste evento estará siempre activo mientras la cámara nos proporciona fotogramas. Otro evento que se utiliza es el que se produce cuando se toca la pantalla.

1.2. Librería del proyecto

Android es programación en Java, por lo tanto la inclusión de librerías en java no supone ningún problema para la aplicación a desarrollar. Esto permite poder desarrollar un librería en Java y poder utilizar sus funciones o variables sin problemas.

Como el proyecto se ha llevado acabo en Android, se ha facilitado una librería creada en lenguaje de programación Java, que procesa las imágenes que recibe de la aplicación, para poder analizar la densidad del tráfico y el seguimiento de trayectorias.

El trabajo ha consistido en poder adaptar la librería a la aplicación Android y utilizar sus funciones para que realice el tratamiento y devuelva por pantalla en el terminal la imagen ya procesada(Figura 1.2).

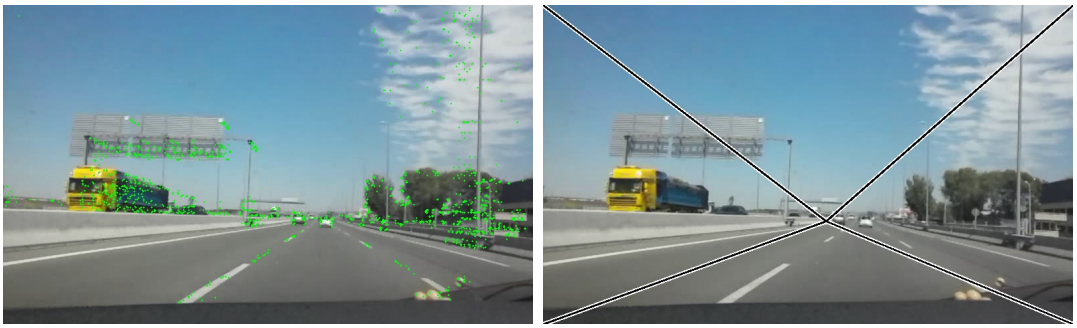


Figura 1.2: Imagenes procesadas por la librería

1.3. Elementos de un terminal móvil

Hay muchos elementos en un terminal móvil que pueden ser utilizados, pantalla, micro, cámara, etc... En una aplicación que necesita un procesamiento de imágenes es muy importante la utilización del elemento de la cámara.

La cámara para el desarrollo de esta aplicación es indispensable, éste proyecto necesita que el terminal móvil pueda coger el flujo de imágenes y que sea capaz de procesar cada frame que es recogido de la cámara.

En algunos terminales móviles hay hasta 2 cámaras, la cámara trasera y la cámara delantera. Teniendo mejor calidad de imágenes la cámara trasera. Por tanto es importante que se abra la cámara adecuada para el desarrollo de ésta aplicación.

Capítulo 2

Librerías

Éste proyecto Android cuenta con una librería llamada *trajectories.jar*, Ésta librería es muy importante para el desarrollo de la aplicación. Ésta librería se ha facilitado, para la realización de éste proyecto y se va a realizar una pequeña introducción sobre las funciones con las que cuenta ésta librería. Cuenta con 2 elementos importantes, *trajectories* y *foe*, estos 2 paquetes son los que nos facilitan el procesado de la imagen para ésta aplicación en Android. Estos paquetes cuentan con unos métodos y variables que han sido los mas utilizados para ésta aplicación, son los siguientes:

2.1. Paquete Trajectories

En ésta sección se va hablar de las principales funciones o variables del paquete *trajectories* que se han utilizado en el sistema Android.

- `public static java.lang.String[] view`
  Éste vector es el que indica el nombre de cada una de las vistas de la cámara.En la versión actual tenemos 4 vistas que son:
  - Camera: Es la cámara sin ningún tipo de operación.
  - Points: Dibuja los puntos de interés.
  - Flow: Dibuja el flujo óptico.
  - Trajectories: Dibuja la trayectoria de la imagen.
- `public static Parameter[] parameters`
  Éste vector es el que indica el nombre de cada uno de los parámetros del módulo, aparte de sus valores mínimos y máximos que puede tomar cada uno de ellos. Los parámetros de la versión actual se detallan a continuación.
  - Length: La longitud mínima para que una trayectoria sea válida.
  - Occlusions: Número máximo de oclusiones antes del final de trayectoria.
  - Scale: Éste parámetro indica la escala inicial (la resolución de la imagen).

  - Octaves: Octavas totales para calcular.
  - Distance: Umbral para el punto de interés y el vector de puntos coincidentes.
  - Harris: Umbral para detectar el punto "*Harris*".
  - LucasIter: Número de iteraciones para el algoritmo "*Lucas-kanade*".
  - LucasEps: Valor de épsilon para el algoritmo de "*Lucas-kanade*".

- `public void update(org.opencv.core.Mat image, long millis)`
  Es la función principal de la clase. Ésta función toma el frame actual del video y extrae las trayectorias. Los parámetros que se le pasan son, la imagen actual, y la hora en milisegundos en la que ha sido dada la imagen, esto es porque se está trabajando con una frecuencia de imagen no uniforme entonces se necesita una precisión de milisegundos, que es lo que se recomienda.

- `public Trajectories(org.opencv.core.Mat image, long millis)`
  Es el constructor del objeto *trajectories*, necesita como parámetros la imagen inicial y el tiempo en milisegundos para trabajar con precisión.

- `public void drawView(java.lang.String view,org.opencv.core.Mat image)`
  Éste método dibuja los resultados del sistema en la imagen dada, en función de la vista que se solicita al método se dibuja una u otra vista. En trajectories tenemos las 4 vistas ya antes mencionadas, que son la "*Camera*", "*Points*", "*Flow*" y "*Trajectories*".

- `public void changeParameter(java.lang.String parameter, double value)`
  Éste método cambia el valor de los *parameters*, son los comentados anteriormente. Hay que pasarle al método el nombre del parámetro a cambiar y el valor del parámetro en tipo *double*.

2.2. Paquete FOE

En ésta sección se va hablar de las principales funciones o variables del paquete *foe* que se han utilizado en la aplicación Android.

- `public static java.lang.String[] view`
  Éste vector es el que indica el nombre de cada una de las vistas de la cámara. En la versión actual tenemos 2 vistas que son:

  - Vanish: Desvanecimiento.
  - FOE: Foco de expansión.

- `public static Parameter[] parameters`
  Éste vector es el que indica el nombre de cada uno de los parámetros de las vistas aparte de sus valores mínimos y máximos que puede tomar cada uno de ellos. Los parámetros de la versión actual se detallan a continuación.

- BW: Media del desplazamiento del ancho de banda.
- GeoError: Máximo error geométrico permitido.

- `public FOEComputation(java.util.LinkedList<Trajectory> trajectoryList)` Es el constructor, que necesita como parámetro *trajectoryList*, que es un enlace a la lista de trayectorias. La clase necesita acceso constante en cada iteración. Para pasarle éste parámetro se facilita un método de la clase *trajectories* llamado *public java.util.LinkedList<Trajectory>getTrajectories()*.

FOEComputation tambien tiene los métodos `drawView` y `changeParameter` , y el funcionamiento que tienen es el mismo que el comentado anteriormente, para dibujar la vista y cambiar los parámetros de foe respectivamente.

2.3. Prueba de ejecución en PC

Se va a realizar un estudio del funcionamiento de la librería *trajectories.jar*, para ello se va hacer una pequeña prueba de la librería en un ordenador pasándole en este caso un vídeo en vez de lo que recoge una cámara. La prueba se va a realizar en el entorno de desarrollo NetBeans (Java), previamente antes de comenzar el proyecto hay que instalar la librería de OpenCV para java en Netbeans.

2.3.1. Código fuente de la prueba de ejecución en PC

A continuación se va a exponer el código de una prueba realizada, el código fuente ésta hecho precisamente para un vídeo.

```
public class TrajectoriesTest {

    public static void main(String args[]) throws
    ParserConfigurationException {
        int i=0;
        //Open video
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
        VideoCapture video = new VideoCapture();
        video.open("C:/Users/Christian/Desktop/video.avi");
        if (!video.isOpened()) {
            System.out.println("Error. Video not found, or video error.");
            return;
        }
```

Creamos un VideoCapture para poder pasarle la ruta de dónde se encuentra el vídeo.

```
Mat image = new Mat();
        video.read(image);
        long time=System.currentTimeMillis();
```

```
Trajectories trajectories = new Trajectories(image,time);
FOEComputation foe = new
FOEComputation(trajectories.getTrajectories());
trajectories.changeParameter("Occlusions", 4.0);
trajectories.changeParameter("Length", 4.0);
trajectories.changeParameter("Harris", 0.3);
```

Se lee la imagen del vídeo y se crea el objeto *trajectories*. También se cambian los valores de un par de parámetros para que veamos cómo se comporta.

```
while (video.read(image)) {
            time=System.currentTimeMillis();
            trajectories.update(image,time);
            foe.computeFOE(time);
            Mat imgAux = image.clone();
            foe.drawView("FOE",imgAux);
            Highgui.imwrite("imgF"+i+".jpg", imgAux);
            i++;

        }
```

Éste es el bucle principal, mientras el vídeo tenga imágenes disponible, actualizará con el método *trajectoriaes.update(image,time)* las trayectorias. En este caso se va a pintar el foco de expansión, pero se podrían representar cualquiera de las funciones comentadas anteriormente. Escribimos el resultado en imgF.jpg para ver el resultado.

2.3.2. Resultado de la prueba de ejecución en PC

A continuación se van a exponer unos resultados de la prueba que se ha realizado con el PC en java. La prueba se ha realizado con un vídeo avanzando en la autovía con un vehículo. Para que se vea el cambio y el funcionamiento más a fondo de la librería, en la prueba se van a poner 3 fotogramas seguidos, observándose así el cambio de cada fotograma (Figura 2.1, Figura 2.2 y Figura 2.3).

Figura 2.1: imagen 1 de prueba PC.

Figura 2.2: imagen 2 de prueba PC.

Figura 2.3: imagen 3 de prueba PC.

En las figuras anteriores, arriba a la izquierda se muestra los puntos de interés detectados, arriba a la derecha, el flujo óptico, abajo a la izquierda están las trayectorias detectadas, y abajo a la derecha el foco de expansión obtenido.

Capítulo 3

Desarrollo

El desarrollo de ésta aplicación como ya hemos mencionado se ha hecho sobre el sistema operativo Android. Para la implementación del programa se ha tenido que realizar un análisis sobre ciertos puntos antes de empezar. La herramienta utilizada para el desarrollo de la aplicación en Android ha sido Android Studio.

Un punto a analizar ha sido la posibilidad de poder incluir las librerías de OpenCV en Android ya que son imprescindibles para la realización de la aplicación. OpenCV es muy importante ya que nos permite abrir la cámara, aparte de otras funciones de procesado, la forma de abrir la cámara ha sido mediante la implementación de CameraBridgeViewBase.

Otro punto es la inclusión de la librería *trajectories.jar*, ya que es la que realiza las operaciones de procesado sobre las imágenes. En el apéndice A.2 se analiza cómo incluir una librería .jar en Android Studio.

Y por último la realización de una base de datos para que la configuración de los parámetros no se pierda al volver a iniciar la aplicación. Ésta base de datos se ha programado en ActiveAndroid, es una base de datos muy sencilla de Android.

Éste programa se ha desarrollado con dos clases, *CameraActivity*, que es la clase del acceso a la cámara y al procesado de las imágenes, y *MainActivity*, que es la clase del menú de la aplicación donde se escoge la vista, o los parámetros a cambiar.

El funcionamiento de la aplicación es el siguiente (figura 3.1).

La aplicación según se inicia se abre la cámara, con la vista *"Vanish"* por defecto, para ir al menú hay que realizar un toque en la pantalla, para que salte el evento onTouch.

Una vez en el menú tenemos dos opciones, pulsar los botones de la vista o los botones de los parámetros. Si se pulsa algún botón de la vista la cámara se abre con la vista seleccionada, si se pulsa algún parámetro se abre la cámara con la vista anterior, con una *seekBar* (Barra de progreso). Con la barra se puede modificar el parámetro seleccionado,

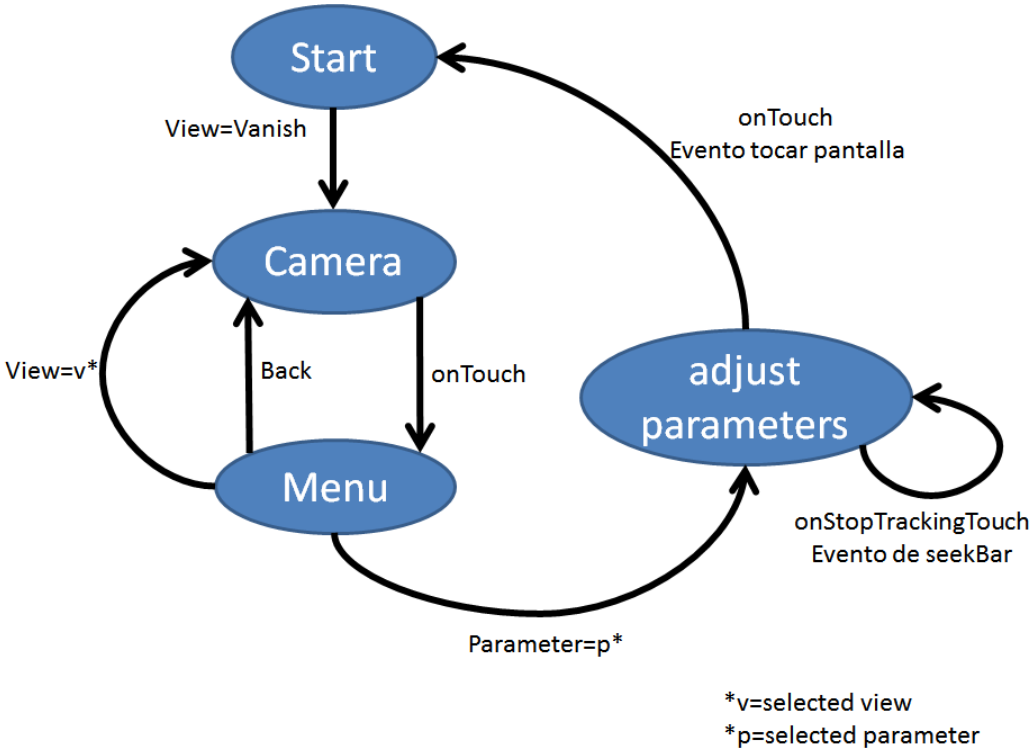


Figura 3.1: Diagrama

ésta modificación se realiza cada vez que la barra se suelta, entonces en Android salta el evento onStopTrackingTouch.

Si se vuelve a producir el evento de tocar la pantalla, onTouch, la barra de progreso desaparecerá, dando por concluida la modificación del parámetro, y si se vuelve a tocar la pantalla, se volverá al menú de la aplicación.

Las interfaces gráficas que usa cada una de las clases java han sido diseñadas para futuros cambios en la aplicación en caso de aumentar la librería con más vistas o parámetros.

3.1. CameraActivity

En ésta clase hay varios puntos importantes que explicar cómo el funcionamiento y uso de la cámara, el evento que se utiliza para la cámara, el modo de ajustar los parámetros y la salida de la cámara al menú.

3.1.1. Funcionamiento CameraBridgeViewBase

OpenCV proporciona ésta clase, *CameraBridgeViewBase*, que representa la cámara en vivo. Por otra parte ésta clase implementa 2 interfaces, que son, *CvCameraViewListener*

o *CvCameraViewListener2*.

Entre las dos interfaces se ha escogido *CvCameraViewListener2* por lo siguiente. Las dos tienen en común que nos proporcionan eventos de la cámara para el momento en el que la cámara se inicia, el momento cuando para y el momento de captura de cada frame. La interfaz *CvCameraViewListener* recibe un frame en color RGBA, que se pasa como una instancia *Mat* de la clase OpenCV. *CvCameraViewListener2* recibe cada frame como instancia de la clase *CvCameraViewFrame*. Desde ésta clase, *CvCameraViewFrame*, podemos obtener una imagen *Mat* en color RGBA y en formato de escala de grises.

La conclusión es que *CvCameraViewListener2* nos permite poder escoger entre una imagen en RGBA y otra en escala de grises a través de la misma instancia *CvCameraViewFrame*, aportando así mayor flexibilidad.

La clase *CameraBridgeViewBase* de OpenCV implementa dos tipos de cámara, una es *JavaCameraView* y *NativeCameraView*, esta última es un envoltorio de Java de una clase C++, esta última produce mayores velocidades de fotogramas por segundo y se utiliza para el desarrollo de la aplicación.

Para el funcionamiento de la aplicación Android, al estar desarrollado con la librería de OpenCV se necesita en el terminal móvil la aplicación .ºpenCV Manager", que se encontrará en el Google Play Store.

Una parte importante del código es la función *BaseLoaderCallback*, que es importante ya que interacciona con la aplicación .ºpenCV Manager", comprobando que las librerías están disponibles.

```
private BaseLoaderCallback mLoaderCallback =
            new BaseLoaderCallback(this) {
                @Override
                public void onManagerConnected(final int status) {
                    switch (status) {
                        case LoaderCallbackInterface.SUCCESS:
                            Log.d(TAG, "OpenCV loaded successfully");
                            mCameraView.enableView();
                            break;
                        default:
                            super.onManagerConnected(status);
                            break;
                    }
                }
            };
```

En caso de que no se conecte bien con la aplicación “OpenCV manager”,el programa se cerrará informando del error mediante la variable *”status”*.

3.1.2. Funcionamiento de los eventos de la Cámara

Cuando se implementa *CameraBridgeViewBase.CvCameraViewListener2* hay unos métodos que son necesarios, esos métodos son los siguientes:

`public void onCameraViewStarted(final int width, final int height)`
Éste método se ejecuta cuando salta el evento del inicio de la cámara, en este caso éste método no lo hemos utilizado.

`public void onCameraViewStopped()`
Éste otro método se utiliza cuando salta el evento de cerrar la cámara, éste método tampoco se ha utilizado igual que el método anterior.

`public Mat onCameraFrame(final CameraBridgeViewBase.CvCameraViewFrame inputFrame)`

Éste método es uno de los más importantes en la aplicación, ya que se ejecuta siempre que la cámara este activa y pasa el frame que captura la cámara, por tanto éste método es quien nos da la imagen que recoge de la cámara para realizar el procesado mediante la librería *"trajectories.jar"*.

El método se va a dividir en dos partes, una el procesado de la imagen mediante el objeto *trajectories*, y otra la elección de la vista.

```
@Override
  public Mat onCameraFrame(final CameraBridgeViewBase.CvCameraViewFrame
  inputFrame)throws Exception {
      Mat imagen = inputFrame.rgba();
      try {
          long tiempo = System.currentTimeMillis();
          trajectories.update(imagen, tiempo);
          foe.computeFOE(tiempo);
      } catch (NullPointerException e) {
          long tiempo = System.currentTimeMillis();
          trajectories = new Trajectories(imagen, tiempo);
          foe = new FOEComputation(trajectories.getTrajectories());
          cambiarParametros();
      }
```

Cuando se tiene el objeto *trajectories* creado o actualizado lo que hay que hacer es visualizar la vista escogida, esto se hace mediante una variable llamada índice que se calcula en función del botón que se le haya pulsado en el menú, entonces se visualizará una vista u otra.

El método *drawView* tanto de *trajectories* como de *foe* lo que hace es modificar la imagen con el procesamiento de cada vista. Entonces el índice marca qué tipo de vista hay que abrir, y para foe lo mismo, la diferencia es para que no se salga del rango, se rel tamaño de las vistas de *trajectories*, que a la hora de calcular el índice se había sumado previamente.

```
        Mat img = new Mat();
        Imgproc.cvtColor(imagen, img, Imgproc.COLOR_RGBA2RGB);
        if (indice < Trajectories.view.length) {
            trajectories.drawView(Trajectories.view[indice], img);
            return img;

        } else {
            foe.drawView(FOEComputation.view[indice -
            Trajectories.view.length], img);
            return img;
        }
    }

}
```

3.1.3. Ajustar parámetros y funcionamiento de la seekBar

Para ajustar los parámetros que tiene la librería *trajectories.jar*, se ha decidido por utilizar un elemento de Android llamado *seekBar*, se trata de una barra de progreso que se puede cambiar manualmente.

Con ésta barra de progreso se pasa el valor de los parámetros a cada uno de ellos, pero cada parámetro tiene un valor mínimo y máximo distinto.

La *seekBar* está en el *activity* de la cámara, pero cuando no hay parámetros se hace invisible y cuando hay parámetros se hace visible para poder ajustar estos parámetros.

```
/* ajustar límites de la seekBar on el parámetro o hace invisble
la barra si no hay parametro*/
private void ajusteBarra() {
  if (valor == null) { //No hay parametro, la barra es invisible
      progreso.setVisibility(View.INVISIBLE);
  else { // Hay parámetro, la barra se ajusta a valores del parámetro
      if (!FOEoTrajectories) {// Parametro de trajectories
          progreso.setMax((int) ((Trajectories.parameters
```

```
            [indiceValor].maxValue - Trajectories.parameters
            [indiceValor].minValue) * 10));
            variable_array = new Select().from(Parametros.class).
            where("nombre=?",
            Trajectories.parameters[indiceValor].name).execute();
            parametro = variable_array.get(0).getValor();

            if (parametro != Trajectories.parameters[indiceValor].
            minValue) {
                parametro = parametro * 10;
                progreso.setProgress((int) parametro);
            }

        }else {//Parámetro de FOE
            progreso.setMax((int)((FOEComputation.parameters
            [indiceValor].maxValue - FOEComputation.parameters
            [indiceValor].minValue) * 10));
            variable_array = new Select().from(Parametros.class).
            where("nombre=?",
            FOEComputation.parameters[indiceValor].name).execute();
            parametro = variable_array.get(0).getValor();

            if (parametro != FOEComputation.parameters[indiceValor].
            minValue) {
                parametro = parametro * 10;
                progreso.setProgress((int) parametro);
            }
        }
    }
}
```

Tenemos la variable *FOEoTrajectories* que es una variable que distingue entre sí es un parámetro de *trajectories* o de *foe*. El cálculo de ésta variable se produce cuando se calcula "indiceValor" distinguiendo los parámetros de uno u otro. Para modificar, una vez que sabemos qué parámetro es, cogemos sus valores mínimos y máximos para que la *seekBar* se ajuste a lo que tiene ese parámetro. Un problema con la *seekBar* es que siempre empieza en 0, entonces se le resta el mínimo al máximo y luego a la hora de cambiar el parámetro le sumamos el mínimo para no perder la diferencia.

Otra anotación respecto a la *seekBar* es que los números que nos devuelve o al cambiar la barra de progreso son enteros, entonces se multiplica por 10 para poder tener un decimal de ajuste con nuestro parámetro (que son de tipo double).
La *seekBar* tiene 3 eventos :

- `public void onStartTrackingTouch(SeekBar seekBar)`
  Éste evento se inicia cuando la barra empieza a moverse, éste evento se ha optado por no utilizarlo.

- `public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser)`
  Éste evento se inicia mientras la *seekBar* cambia. Éste evento se ha utilizado para obtener el valor de la barra mediante la variable *progress*.

- `public void onStopTrackingTouch(SeekBar seekBar)`
  Éste evento se inicia cuando se ha soltado la *seekBar*. Éste evento se usa para el cambio de parámetros, y que se guarden en la base de datos.

```
@Override
public void onStopTrackingTouch(SeekBar seekBar) {
    parametro = operacionDeAjuste();
    // divide entre 10 para ternlo con un decimal
    guardarEnDatabase();
    if (FOEoTrajectories) {// Parametro de trajectories
        parametro = parametro +
                FOEComputation.parameters[indiceValor].minValue;
        foe.changeParameter(FOEComputation.parameters[indiceValor].name,
                parametro);
    } else {//Parametro de FOE
        parametro = parametro +
                Trajectories.parameters[indiceValor].minValue;
        trajectories.changeParameter(Trajectories.parameters
        [indiceValor].name,parametro);
    }
}
```

3.1.4. Funcionamiento del evento onTouch

El funcionamiento de *onTouch* es simple, se produce cuando se toca la pantalla, pero la manera de tocar la pantalla es variada, es decir, hay varias acciones que Android nos permite utilizar.

Algunas de las opciones al tocar la pantalla son:

- **ACTION_DOWN**
  Éste evento se da cuando pulsamos la pantalla, es decir, presionamos la pantalla y no movemos el puntero hacia ningún lado.

- **ACTION_MOVE**
  Éste evento se da cuando tras pulsar la pantalla, procedemos a mover el puntero por ella, es decir, cuando hagamos esto surgirán 2 eventos, el ACTION_DOWN y a continuación al mover el puntero aparecerá el evento ACTION_MOVE.

- **ACTION_UP**
  Éste evento se da cuando levantamos el dedo o puntero de la pantalla, para que se dé éste evento debe haberse dado anteriormente el evento ACTION_DOWN.

```
public boolean onTouch(View v, MotionEvent event) {
        int action = MotionEventCompat.getActionMasked(event);
        switch (action) {
            case (MotionEvent.ACTION_DOWN):
                if (valor == null) {//Si no hay parametro
                    Intent i = new Intent(this, MainActivity.class);
                    startActivity(i);
                    return true;
                } else { //Si hay parametro
                    progreso.setVisibility(View.INVISIBLE);
                    valor = null;
                }
            default:
                return false;
        }

    }
```

En este caso se ha elegido la acción ACTION_DOWN para el método onTouch, se ha utilizado para en caso de que se haya abierto la cámara con la barra de progreso, se toque la pantalla y desaparezca la barra de progreso, y si se vuelve a tocar la pantalla se irá al menú de la aplicación.

3.1.5. Interfaz gráfica de la cámara

La interfaz de la cámara se ha realizado sobre un *layout* llamado *camera_ layout*. Ésta interfaz requería de la cámara y del *seekBar* para cambiar el valor de los parámetros.

```
<org.opencv.android.NativeCameraView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:visibility="gone"
```

```
    android:id="@+id/activity_native_view"
    opencv:show_fps="true"
    opencv:camera_id="any" />

<SeekBar
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/progreso"
    android:layout_gravity="center_horizontal|bottom" />
```

Por tanto no se ha necesitado más que estos dos elementos para realizar la interfaz, la cámara que se ha elegido es *NativeCameraView*.

Ésta cámara depués en el código Android hay que asiganarla a la variable correspondiente cuando se crean todos los elementos en Android en el método *onCreate*.

```
mCameraView = (CameraBridgeViewBase) findViewById
(R.id.activity_native_view);
```

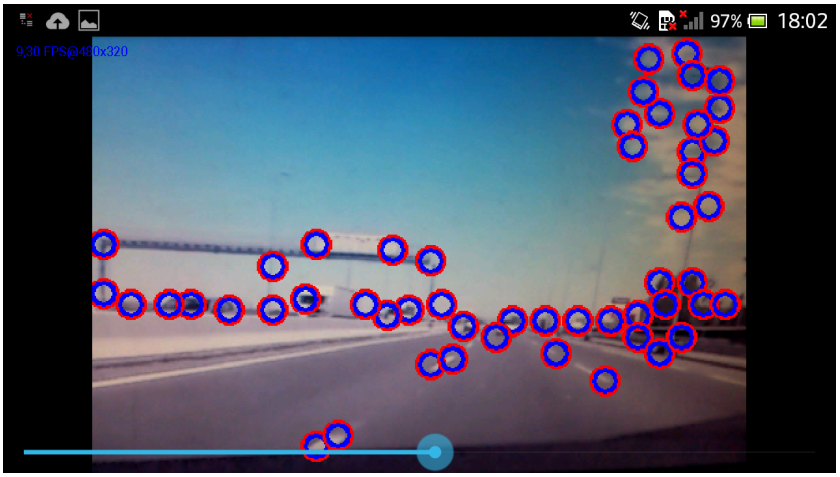


Figura 3.2: Captura de la camara con *seekBar*

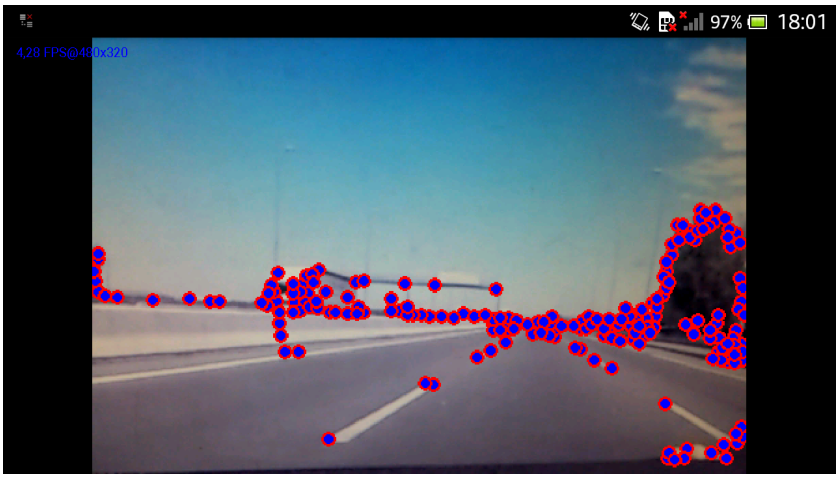


Figura 3.3: Captura de la cámara

Después de realizar esto la cámara ya está lista para ser utilizada y el aspecto será el mostrado en las figura 3.2 y figura 3.3.

3.1.6. Base de datos

La base de datos de la aplicación se ha realizado en ActiveAndroid, es un modelo de base de datos de Android muy sencillo.
La base de datos de ésta aplicación es muy simple, está formada por 2 tablas. Una tabla para los parámetros y otra para la vista, con el nombre de la tabla como "Parametros" y "NombreCam" respectivamente. En la tabla "Parametros" se guarda el valor de cada uno de los parámetros para no tener que ajustar los cada vez que se quiera usar la aplicación. La tabla de la vista es para saber cuál fue la última vista utilizada y abrir la cámara con esa vista.

Cuando se inicia la aplicación se ejecuta un método llamado *inicializarDatabase*, que es una función que rellena la tabla de los parámetros con los valores mínimos de cada uno de los parámetros, y la tabla del nombre de la vista, con la vista por defecto que es *"Vanish"* (que es el valor de *"cam"* al iniciar la aplicación).

```
private void inicializarDatabase() {
    variable_array = new Select().from(Parametros.class).execute();
    nombreCam_array = new Select().from(NombreCam.class).execute();
    if (variable_array.isEmpty()) {
        for (int i = 0; i < FOEComputation.parameters.length; i++) {
            variable = new Parametros();
```

```
            variable.setNombre(FOEComputation.parameters[i].name);
            variable.setValor(FOEComputation.parameters[i].minValue);
            variable.save();
        }
        for (int j = 0; j < Trajectories.parameters.length; j++) {
            variable = new Parametros();
            variable.setNombre(Trajectories.parameters[j].name);
            variable.setValor(Trajectories.parameters[j].minValue);
            variable.save();
        }
    }
    if (nombreCam_array.isEmpty()) {
           nombreCam = new NombreCam();
           nombreCam.setNombre(cam);
           nombreCam.save();
    }

}
```

Se realiza una consulta en la base de datos de la tabla vacía entonces se rellena la tabla, que tiene dos entradas en la tabla, una es el nombre del parámetro, y la otra el valor que el parámetro tiene. En un principio tendrá el mínimo, pero en el caso de que el progreso del parámetro cambie, se ejecutará el método correspondiente para guardar el nuevo valor en la base de datos.

Una vez que se produce el evento onStopTrackingTouch, es cuando el parámetro se guarda en la base de datos con el método *guardarEnDatabase*.

```
 private void guardarEnDatabase() {
    variable_array = new Select().from(Parametros.class).execute();
    if (variable_array != null) {
        for (int i = 0; i < variable_array.size(); i++) {
            if (variable_array.get(i).getNombre().equals(valor)) {
                variable_array.get(i).setValor(parametro);
                variable_array.get(i).save();
                break;

            }
        }
    } else {
        variable = new Parametros();
        variable.setNombre(valor);
        variable.setValor(parametro);
```

```
        variable.save();

    }
}
```

Es una consulta a la base de datos y en cuanto encuentra el nombre del valor correspondiente en la tabla introduce el nuevo valor en la tabla y se guarda.

La otra tabla de la base de datos que se ha llamado "NombreCam" se ha utilizado para guardar nada más el nombre de la vista para saber cual ha sido la utilizada, y así abrir la cámara con la última vista seleccionada.

```
 private void guardarCamara() {
    nombreCam_array = new Select().from(NombreCam.class).execute();
    if (nombreCam_array.isEmpty()) {
        nombreCam = new NombreCam();
        nombreCam.setNombre(cam);
        nombreCam.save();
    }else{//(nombreCam_array != null) {
        nombreCam_array.get(0).setNombre(cam);
        nombreCam_array.get(0).save();
    }
}
```

Funciona parecido que en la tabla "Parametros", la diferencia es que solo tiene una entrada en la tabla que es el nombre de la vista y que solo vamos a guardar un valor.

3.2. MainActivity

En la clase *MainActivity* se ha creado el menú de la aplicación, en Android es sencillo crear menús con ayuda de los *Layout*, pero en este caso el menú se crea dinámicamente mediante código Java. Esta manera de crear el menú en ésta aplicación tiene la finalidad de que en un futuro, la librería *trayectories.jar*, se haya ampliado con más parámetros, o con más vistas que introducir en la aplicación, y que por tanto siga funcionando ésta interfaz gráfica que se ha creado sin tener que modificar nada de la aplicación en un futuro.

Primero se hablará de cómo está formada la interfaz gráfica de la aplicación mediante los *Layouts*, y luego como se crea mediante código Java los elementos Android que se utilizarán para la aplicación.

3.2.1. Interfaz gráfica del menú de la aplicación

La interfaz gráfica del menú de la aplicación es sencilla, el Layout de la interfaz del menú se ha dividido en dos partes. Estas dos partes son con 2 ScrollView, que es un elemento de Android que nos permite mover la pantalla cuando no caben todos los elementos en la pantalla.

Estos ScrollView corresponde uno a la parte de las vistas de las cámaras que tiene la aplicación y el otro corresponde a los parámetros.

Éste Layout, como he dicho, tiene dos ScrollView, pero dentro de cada ScrollView hay otro elemento de Android llamado LinearLayout, éste elemento nos permite colocar elementos de Android uno a continuación de otro de una manera ordenada. Éste elemento se utiliza para colocar los elementos que creamos mediante código, que en este caso serán botones.

```
 <ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/scrollView3"
    android:layout_gravity="center_vertical"
    android:layout_toLeftOf="@+id/textView2"
    android:layout_toStartOf="@+id/textView2"
    android:layout_below="@+id/textView">

    <LinearLayout
        android:layout_width="280dp"
        android:layout_height="match_parent"
        android:id="@+id/linearLayout"
        android:orientation="vertical"
        android:layout_below="@+id/textView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true">


    </LinearLayout>
</ScrollView>
```

Esta es la estructura de uno de las ScrollView que se utiliza para poner los botones dentro del LinearLayout, y en caso de que no quepa en la pantalla, el scrollView nos dará la posibilidad de mover la pantalla hacia abajo en busca de las demás opciones que se presentan.

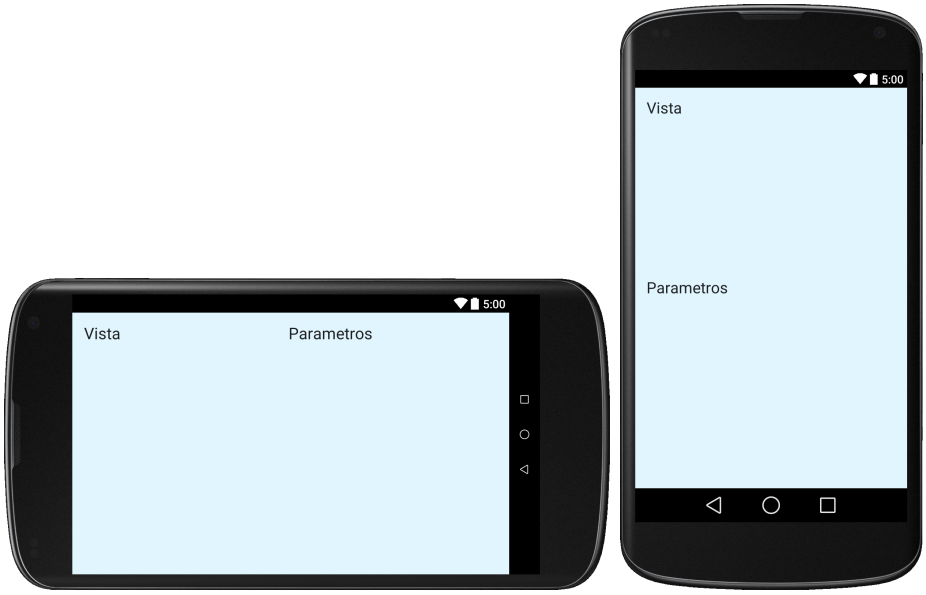


Figura 3.4: Layout del menú

Una imagen de cómo queda la interfaz antes de rellenar los elementos con código (Figura 3.4).

3.2.2. Crear menú de la aplicación mediante código Java

Mediante código se pueden crear elementos en Android, en este caso tanto las vistas como los parámetros funcionarán mediante un botón.

A la hora de crear los botones, estos se tienen que asociar con el *LinearLayout* y también tenemos que darle el tamaño o colores y forma que tendrá.

No hay que olvidar el evento al que está asociado el botón, éste evento se llama *setOnClickListener*. Éste evento se ejecuta cuando el botón ha sido pulsado. Al haber muchos botones el evento sería el mismo para todos pero esto en Android se puede solucionar o bien con el ID (no tenemos porque el elemento se ha creado dinámicamente) o bien asociando al botón, su evento solamente a él.

```
LinearLayout layout = (LinearLayout) findViewById(R.id.linearLayout);
      for (int i = 0; i < FOEComputation.view.length; i++) {
          final Button btnCam = new Button(this);
          btnCam.setLayoutParams(new LinearLayout.LayoutParams
            (LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.MATCH_PARENT));
          LinearLayout.LayoutParams layoutParams =
```

```
            new LinearLayout.LayoutParams
            (LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.MATCH_PARENT);
            btnCam.setText(FOEComputation.view[i]);
            btnCam.setBackground(getResources().
            getDrawable(R.drawable.boton_principal));
            btnCam.setTextColor(Color.WHITE);

            layoutParams.setMargins(40, 20, 50, 0);
            layout.addView(btnCam, layoutParams);
            btnCam.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    camName = (String) btnCam.getText();
                    camActivity(camName);
                }
            });
        }
```

Éste código es válido tanto si el móvil está en posición vertical cómo horizontal, esto es por la forma que tiene Android de asignar los recursos, es decir, que el dispositivo estando en vertical se le asignará un *layout* con orientación vertical y si está en horizontal se le asignara un *layout* con orientación horizontal. Para hacer esto nada más hay que crear una carpeta en el mismo directorio del proyecto donde se encuentra la carpeta *layout*, y la carpeta ha de llamarse *layout-land*, en está carpeta creas los *layouts* de orientación horizontal, una vez hecho esto, Android asignará los recursos sin que nosotros tengamos la necesidad de programar nada más. En el Android Studio aparecerá de la siguiente manera.

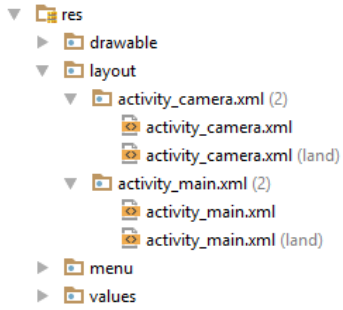


Figura 3.5: Captura de pantalla

En este caso, queremos que todos los botones ejecuten *CamareActivity*, entonces obtiene el texto del botón para mandárselo a *CameraActivity*, así sabe qué vista abrir o qué parámetro hay que cambiar. Éste texto está proporcionado y controlado por la librería.

En el código se ajusta la medida, se le cambia el texto y también se le cambia la forma del botón, esto se hace mediante un código de colores y se reducen las esquinas del botón, etcTambién hay que ajustar los márgenes del botón con el *LinearLayout* e introducir estos cambios en el *Layout*.

Aquí se puede ver cómo quedó finalmente el menú (ver Figura 3.6).

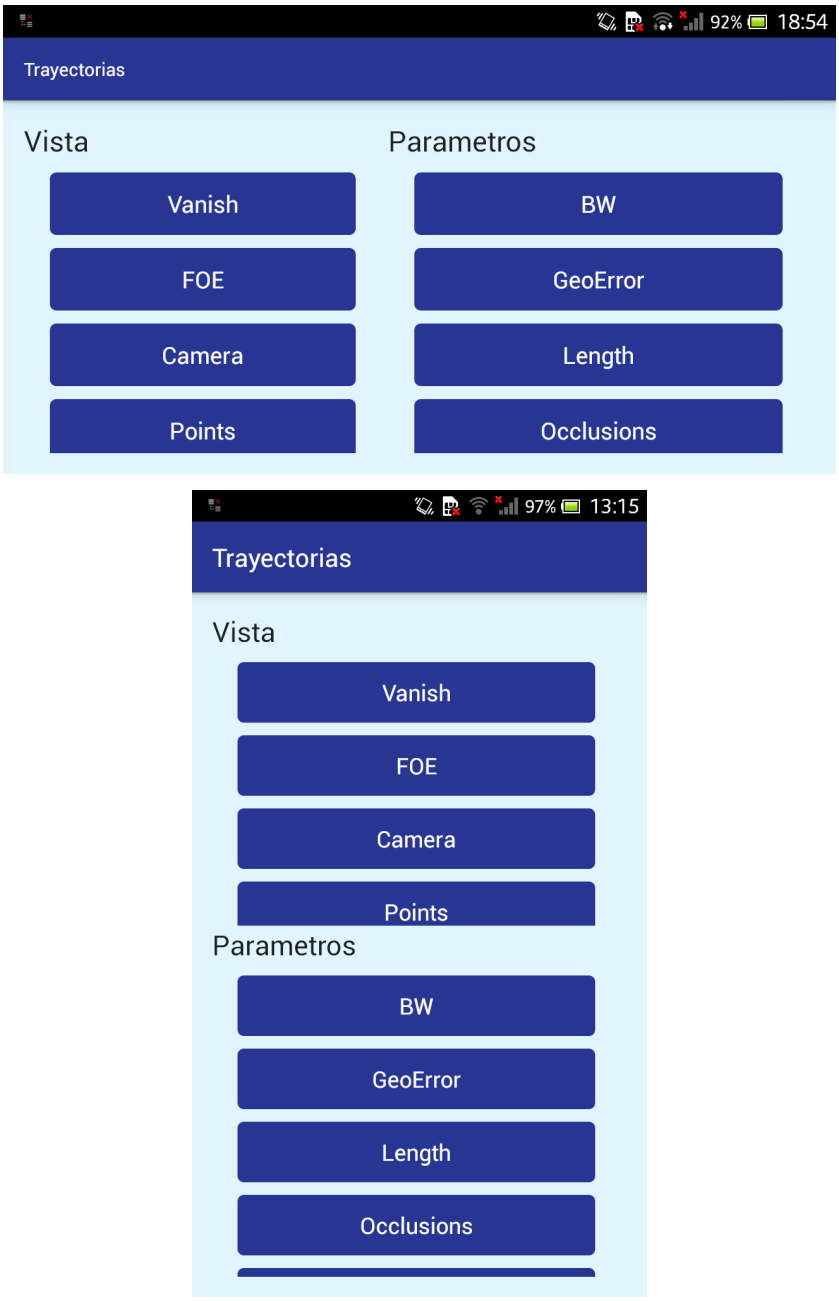


Figura 3.6: Captura del menú

Capítulo 4

Manual de usuario

El uso de ésta aplicación es específico para cuando se está en un vehículo. El móvil se tiene que colocar con un soporte en el vehículo mirando hacia la carretera con la cámara.

Figura 4.1: Terminal colocado en el vehículo

Antes de poder utilizar la aplicación hay que realizar la instalación de la aplicación OpenCV Manager que se encuentra en el Play Store (Figura 4.2).

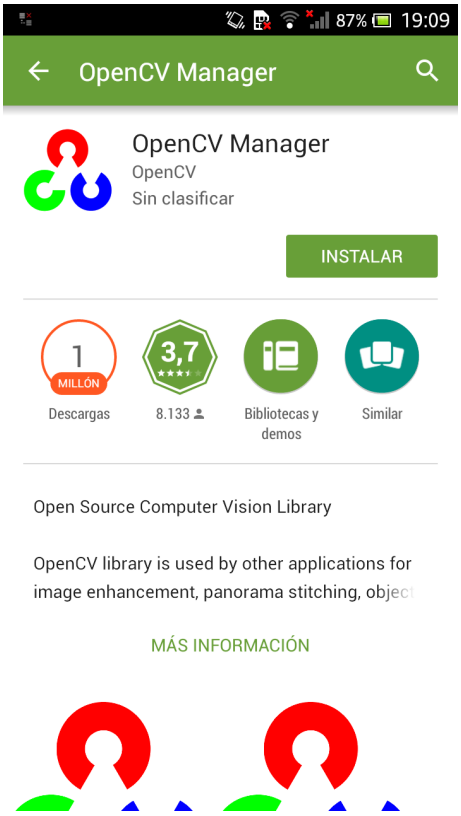


Figura 4.2: imagen del Play Store

Después de realizar la instalación de OpenCV Manager, la aplicación está preparada.

Cuando la aplicación se inicie se abrirá la cámara con una de las vistas, por defecto, la vista "Vanish". Las vistas se podrán cambiar en el menú, para acceder al menú toque la pantalla del terminal móvil (Figura 4.3).

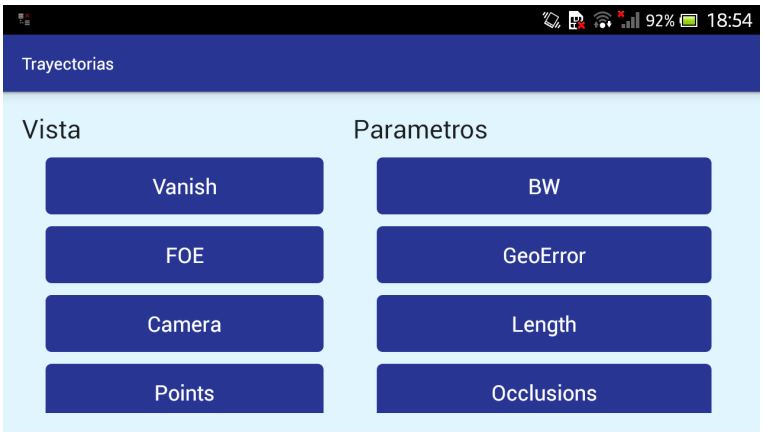


Figura 4.3: imagen del menu

En el menú puede pulsar un botón de vista o parámetros. Pulsando el botón de la vista que quiere usar la cámara se lanzará con la vista escogida. Si quiere cambiar de vista solo tendrá que tocar la pantalla otra vez (figura 4.4).

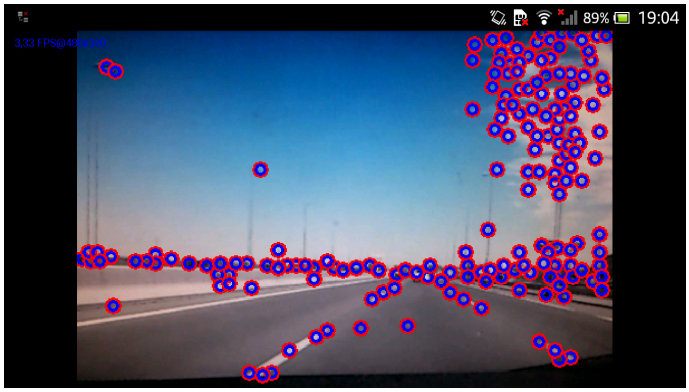


Figura 4.4: imagen de la camara

En caso de que se deseen cambiar los parámetros de la vista ya que se consideran que no son oportunos para su terminal móvil, cámara o para el procesamiento de esa vista en general, tendrá que acceder al menú de la aplicación y pulsar el botón del parámetro que se desea cambiar. Aparecerá la vista que tenía seleccionada con una barra de progreso que podrá ajustar hasta que se considere que el parámetro está bien ajustado a las condiciones deseadas (Figura 4.5).

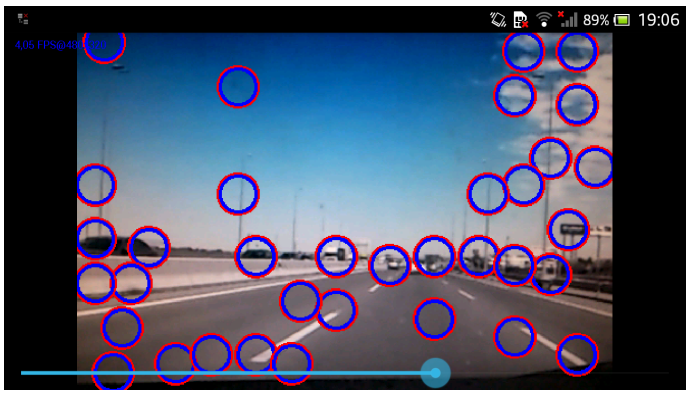
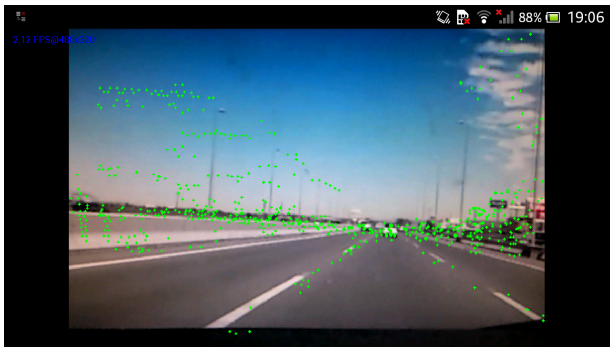
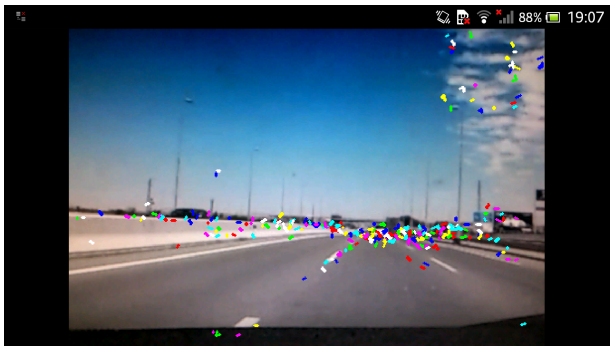


Figura 4.5: imagen de la camara para ajustar el parametro

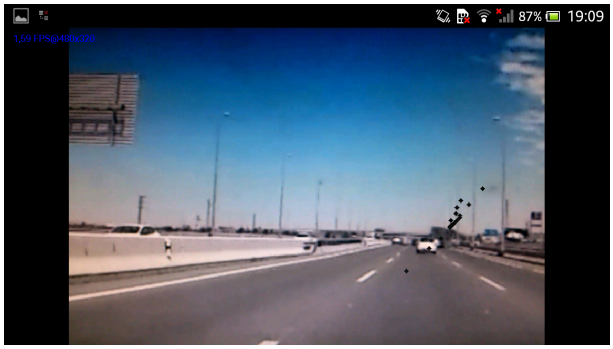
Para ajustar el parámetro tiene que arrastrar la barra y soltar, una vez se suelte la barra, se actualizará el parámetro. Cuando el parámetro se tenga como se quiere, para hacer desaparecer la barra, toque la pantalla una sola vez, la barra desaparecerá, y si toca la pantalla otra vez se volverá al menú de la aplicación.



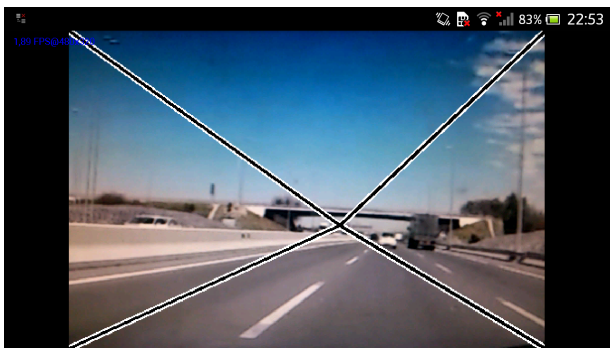
(a) *Vista Flow*



(b) *Vista Trajectories*



(c) *Vista Vanish*



(d) *Vista FOE*

Figura 4.6: Capturas de vistas

Apéndice A

Apéndice

A.1. Instalar OpenCV en Android Studio

Lo primero que hay que hacer es descargarse OpenCV for Android, una vez tengamos OpenCV for Android y nuestro proyecto de Android creado, tenemos que ir a File ->New ->Import Module ... (http://opencv.org/downloads.html).
Una vez aquí, tenemos que buscar nuestra carpeta de OpenCV y escoger la carpeta de sdk->java.

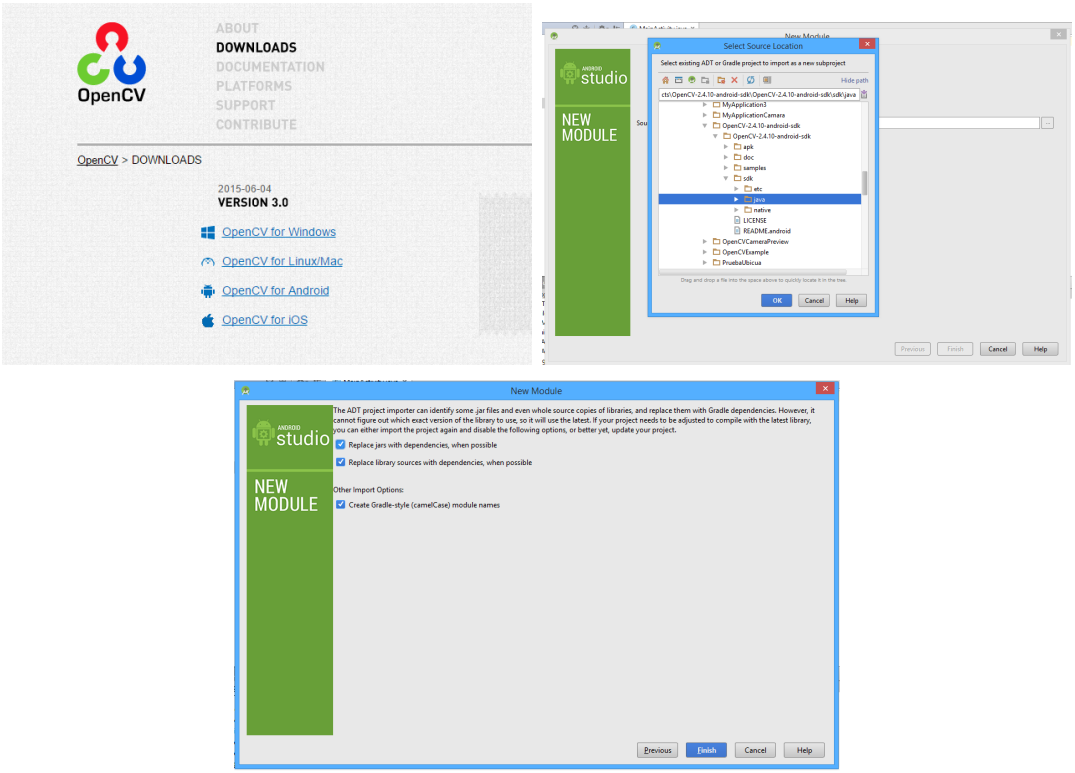


Figura A.1: Capturas de Android Studio

Una vez hecho esto y tras la espera de unos segundos el archivo de configuración de Android gradle ya tendrá configurado OpenCV. Después hay que configurar las dependencias, para ello vamos a File->Project Structure...
Aquí una vez seleccionado "app" vamos a la pestaña de dependencias y le damos al "+" para añadir una nueva, y seleccionamos la opción, "Module dependency", y una vez seleccionado, marcamos la opción de OpenCV .

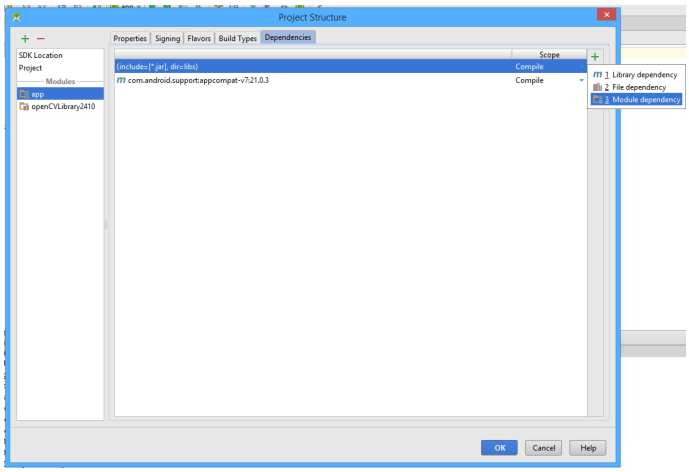


Figura A.2: Capturas de Android Studio

Haciendo esto nuestro proyecto Android ya está listo para utilizar OpenCV.

A.2. Importar una librería .jar en Android Studio

Para añadir una librería de java .jar en Android Studio, hay que crear una carpeta en el directorio donde se encuentran nuestras clases java o el "MainActivity", con el nombre de "libs". Después pegamos la librería .jar en ésta carpeta. Si vamos a Android Studio y miramos deberá ya salir el jar en la parte del proyecto y en su directorio correspondiente.

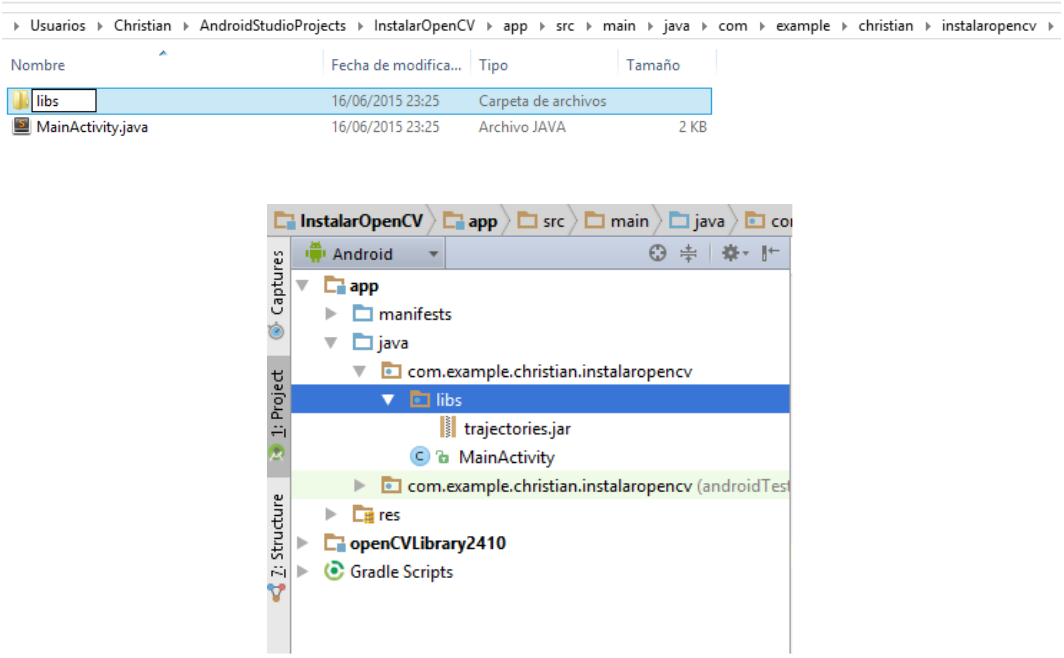


Figura A.3: Capturas de Android Studio

Una vez aquí, en el Android Studio, damos a botón derecho encima de la librería .jar y aparecerá una opción llamada “Add as Library...”, aparecerá una ventana donde desactivamos la opción “doc” y le damos a “OK“, aparecerá otra ventana y también le damos a “OK”.

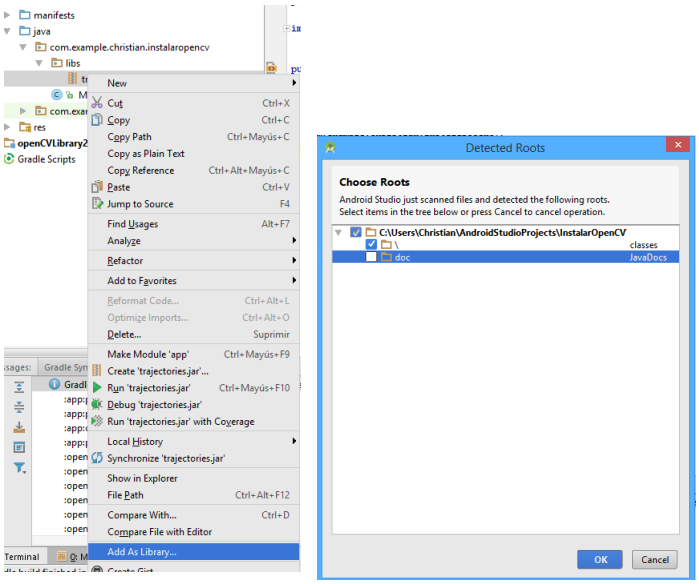


Figura A.4: Capturas de Android Studio

Y haciendo todo esto ya habremos añadido nuestra librería .jar para utilizarla en nuestro proyecto Android.

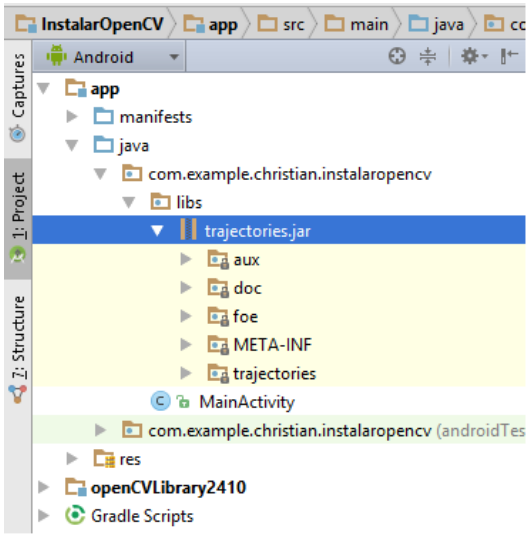


Figura A.5: Capturas de Android Studio

A.3. Ejecutar y depurar aplicación en el móvil

Para compilar y ejecutar la aplicación en el móvil, hay que tener el móvil con las opciones de desarrollador activadas, algunos móviles lo tienen por defecto otros móviles hay que activarlo, para ello hay que ir a los ajustes del terminal y buscar la opción "acerca del teléfono" (en otros móviles puede llamarse de distinta manera), y hay que buscar el número de compilación, y lo que hay que hacer es tocar varias veces repetidamente encima del número de compilación y cuando sea suficiente saltará un mensaje diciendo que se han activado las opciones de desarrollador.
Una vez activadas las opciones de desarrollador, para ejecutarlo en el ordenador, hay que ir a ajustes, opciones de desarrollador y activar la opción depuración USB.

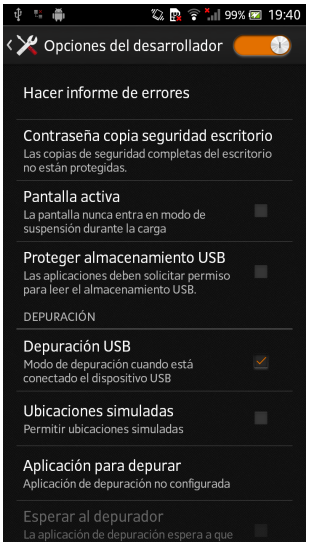


Figura A.6: Captura del móvil

Una vez hecho todo esto conectamos el móvil al ordenador y en nuestro Android Studio debería salir el dispositivo conectado. Para ejecutar la aplicación solo hay que pulsar el botón de, run .ªpp", o Mayús. + F10. Aparecerá una ventana de dónde queremos ejecutar nuestra aplicación, elegimos nuestro terminal móvil y esperamos unos segundos y la aplicación se pondrá en funcionamiento.

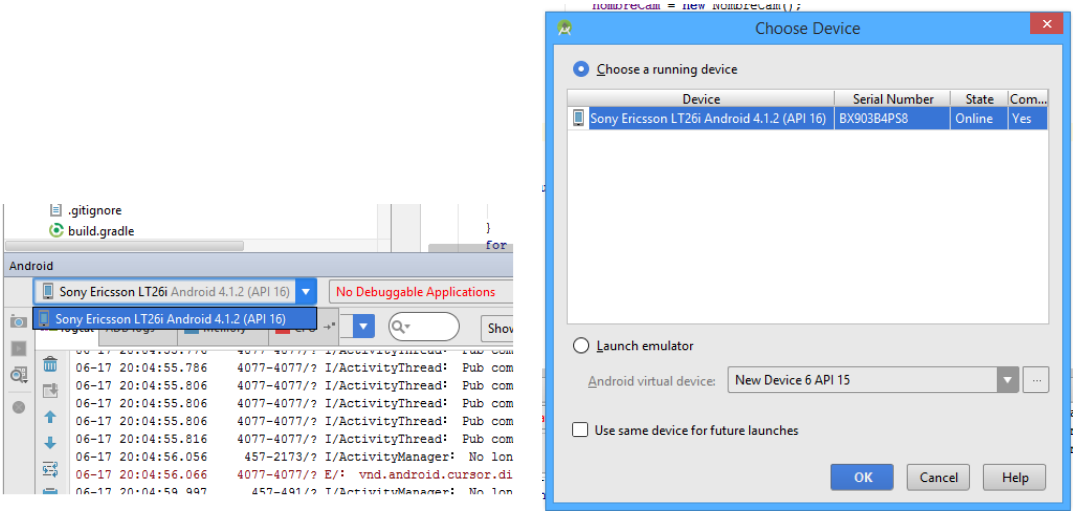


Figura A.7: Capturas de Android Studio

A.3.1. Depuración

Para la depuración el procedimiento es el mismo lo único que hay que pulsar el botón debug .ªpp", o Mayús. +F9. Previamente habrá que colocar un punto de depuración en

el código. La aplicación se quedará parada en ese punto del código y se puede ejecutar paso a paso con el botón "Step Over", o F8.

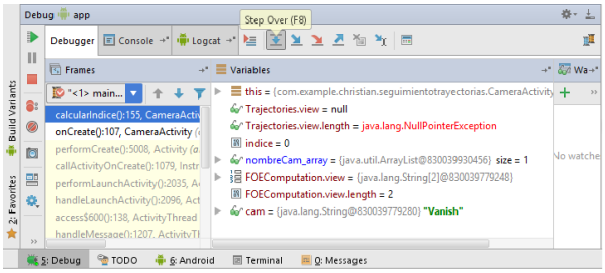


Figura A.8: Captura de Android Studio