UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

Escuela Técnica Superior de Ingenieros de Telecomunicación

Universitat Politècnica de València

Cálculo de volúmenes de pilas de madera

Proyecto Final de Carrera

Ingeniería de Telecomunicación

Autor: Sergio Ferrer Rozalén

Director: Miguel Ángel Mateo Pla

11 de septiembre de 2015

"Ser el hombre más rico del cementerio no me interesa. Ir a la cama por las noches pensando que hemos hecho algo maravilloso, eso sí me importa." – Steve Jobs

Me gustaría dar las gracias a mi familia, amigos y compañeros por estar ahí siempre que lo he necesitado. Y, en general, a cualquier persona que haya formado parte de mi vida durante algún momento a lo largo de todos estos años, porque sin toda esa gente no sería quien soy hoy. Gracias a todos por recorrer este camino conmigo, sacarme mil sonrisas y hacer que todo en esta vida sea un poco más fácil.

ÍNDICE

1. Introducción

1.1. Objetivos y motivación

En esta memoria se describe el trabajo desarrollado en el Proyecto Final de Carrera para la obtención del título de Ingeniero de Telecomunicación. El trabajo desarrollado es la automatización del proceso de medición de volúmenes de pilas de madera. Este trabajo puede dividirse en distintos puntos, basados en las diferentes necesidades del consumidor que abordamos.

En primer lugar, se explica la relación del proyecto con la titulación, para a continuación exponer los conceptos teóricos. En este apartado se verá cómo funciona y qué caracteriza al sistema operativo móvil Android, así como las bases de OpenCV utilizadas a lo largo del proyecto y toda la información necesaria sobre el stitching de imágenes para entender lo que vendrá posteriormente.

En segundo lugar, se entrará en detalles sobre el propio diseño de la aplicación tales como los requisitos, tanto funcionales y no funcionales, como de implementación (necesidades del terminal); los actores y los casos de uso.

En el siguiente punto se mostrará la aplicación desarrollada con mucho más detalle y podrá observar algunas capturas tomadas para entender mejor su funcionamiento.

Para acabar, se mostrarán las conclusiones a las que se ha llegado, seguidas de la biografía complementaria al proyecto y los diferentes anexos del mismo.

Como resumen de la aplicación se puede decir que hará las funciones de asistente fotográfico para la realización de fotografías panorámicas. Teniendo en cuenta que las pilas de madera se extienden a lo largo de muchos metros, en ocasiones entre 20 y 40 metros, se necesitará este tipo de fotografías para poder cubrir toda su longitud.

Ilustración 1 – ¿Cuánta madera hay aquí?

La aplicación de cámara que viene por defecto en los terminales convencionales, tiene varios inconvenientes que dificultan el objetivo final del proyecto e impulsan a desarrollar una aplicación independiente para tomar las fotografías. El primer inconveniente es que ofrece pocas opciones de personalización cuando se utiliza el "modo panorama" de la misma. Al seleccionarlo, la cámara automáticamente baja la calidad de cada foto individual a 1 megapíxel y no permite modificarla (se necesitan aproximadamente 3 megapíxeles para un resultado final óptimo). La aplicación por defecto también fija un valor de zoom, no seleccionable por el usuario, para todo el proceso. Debido a los diversos terrenos que se encontrarán en los lugares donde se usará la aplicación y las complicadas condiciones geográficas de algunos, será requisito indispensable que se pueda seleccionar el zoom que mejor se adecúe a las condiciones concretas de cada sesión. El segundo inconveniente aparece al tratar de utilizar el "modo normal" de la cámara para poder seleccionar la calidad de imagen y zoom según las necesidades. Los potenciales clientes, futuros usuarios (muchos de ellos reacios a la tecnología y los sistemas digitales), perdían la capacidad de orientación fácilmente entre una foto y la siguiente, obteniendo fotos inadecuadas para el procesamiento posterior. La aplicación tiene por función guiar al usuario durante todo el proceso.

Adicionalmente, se podrá medir la distancia entre el usuario y la pila de madera para asegurar que ésta no varía a lo largo de las distintas fotografías (tomadas mediante desplazamientos paralelos y no mediante rotación de la cámara, evitando así la distorsión en la medida de lo posible). También se podrán modificar otros parámetros de la cámara como, por ejemplo, el ISO, la apertura y el tiempo de exposición; así como algunos relacionados con el asistente para la realización de panorámicas, que serán explicados en la presente memoria.

Posteriormente, se añade a la aplicación la funcionalidad de *stitching*. El *stitching* es el término técnico para la composición de panorámicas mediante diversos procesos de ajuste y fusión de imagen. Será un proceso opcional, el usuario podrá realizar el *stitching* en la cámara o en el ordenador, a través de una aplicación que realiza exactamente la misma función. Se decide implementar esta aplicación también para PC debido a principalmente dos motivos: el primero, el usuario medio se siente más cómodo con el uso de ordenadores que con el uso de dispositivos móviles para realizar la tarea y, el segundo, se reduce el tiempo del proceso significativamente. Realizar el *stitching* de fotografías de alta definición en el terminal móvil o cámara es notablemente más lento debido al uso de una tecnología inferior, procesador de menor potencia y menor capacidad de otros componentes, por ejemplo memoria RAM.

Para finalizar, se desarrolla la aplicación que detecta la cantidad de troncos existentes en la panorámica, calcula el área de cada uno de sus cortes transversales y, uniendo estos datos con la longitud de los mismos (fija, todos se cortan del mismo tamaño) calcula el volumen de madera existente en la pila. Esta aplicación se ha desarrollado únicamente para PC por los mismos motivos que se mencionan anteriormente. Por un lado, el proceso es mucho más rápido y, por otro lado, requiere en ciertas ocasiones la interacción del usuario. Para esto último ayuda mucho tener una pantalla grande. Además, hay que

considerar la comodidad que supone para el usuario y la precisión que proporciona el uso del ratón frente al manejo de la pantalla táctil.

El principal objetivo por el que se desarrolla este software es ahorrar tiempo al usuario gracias a la automatización de todo el proceso. Lo que anteriormente suponía horas de trabajo midiendo uno a uno los troncos para, posteriormente, realizar los cálculos necesarios, se reduce ahora a unos minutos.

1.2. Relación con la titulación

A lo largo de la carrera se cursan varias asignaturas relacionadas con la programación. En concreto, en programación avanzada se aprenden las bases de Java, que es el lenguaje que se usará para programar en Android. También existen asignaturas relacionadas con el tratamiento de imágenes.

En este proyecto se desarrollan ambas facetas. En primer lugar nos adentramos más en el mundo de la programación. Y, en segundo lugar, el grueso del proyecto está muy relacionado con el procesado de imágenes para la obtención de los datos necesarios para cálculos posteriores.

Por todo lo comentado anteriormente, considero que el proyecto es una gran oportunidad de profundizar en la materia y ampliar, a través de una experiencia práctica, los conocimientos adquiridos en las clases.

2. Conceptos teóricos

2.1. Android

Para empezar, se hablará de Android, el sistema operativo que utilizará la cámara/terminal móvil que ejecutara la aplicación diseñada para asistir al usuario durante la toma de las fotografías.

2.1.1. Definición

Android es un sistema operativo basado en Linux diseñado principalmente para dispositivos móviles. En un principio, fue desarrollado por Android Inc (más tarde, en 2005, Google adquirió la empresa). Android fue presentado en 2007 junto la fundación del Open Handset Alliance: un consorcio de compañías de hardware, software y telecomunicaciones para avanzar en los estándares abiertos de los dispositivos móviles.

Como punto diferenciador frente a otros sistemas operativos móviles como iOS o Windows Phone, cabe destacar que Android se desarrolla de forma abierta y es posible acceder a su código fuente. Sin embargo, cada fabricante puede añadir cosas a posteriori, por ejemplo, código para soportar su proprio hardware; y éste, normalmente, no es público. Otro de los aspectos que lo diferencia de otros sistemas operativos es que es altamente personalizable y es relativamente sencillo realizar cambios en casi cualquier parámetro si el usuario lo desea. *[1]*

Algunas características principales (y relacionadas con el proyecto) son las siguientes:

- **Conectividad:** soporte para GSM, IDEN, CDMA, EV-DO, UMTS, Bluetooth, WiFi, LTE, HSDPA, HSPA+, NFC y WiMAX. *[2]*
- **Almacenamiento** de datos a través de SQLite.
- **Máquina Virtual Dalvik (MVD):** diseñada especialmente para Android y optimizada para dispositivos con características técnicas limitadas. Es posible ejecutar varias instancias de la MVD simultáneamente, lo que permite aislar procesos, facilitando la gestión de memoria y logrando mejorar el rendimiento. Aunque a menudo es considerada una máquina virtual Java, esto no es del todo correcto ya que no ejecuta Java bytecode, sino que el SDK de Android lo transforma a otro formato de archivo (dex). Recientemente están realizándose las primeras pruebas para ser sustituida por ART, que promete mejoras notables de rendimiento al no requerir que una parte del código se compile en cada ejecución. *[3]*
- Permite el **desarrollo de aplicaciones en Java** (lenguaje principal), pero también se pueden añadir partes de código escritas en C/C++ que interactúan con el código en Java por medio de la JNI (Java Native Interface).
- **Soporte multimedia para audio, vídeo y fotografía** en diversos formatos, entre los que destacan: H.264 (en contenedores 3GP o MP4), MP3, WAP, JPEG, PNG, GIF y BMP.

- **Soporte para muchos tipos de hardware:** cámaras de foto, vídeo, acelerómetros, giroscopios, magnetómetros, GPS, pantallas táctiles, sensores de luz, GPU…
- **Multitarea real:** el procesador sigue trabajando con las aplicaciones que no están ejecutándose en primer plano.

2.1.2. Arquitectura

El sistema Android está dividido en varias capas. *[4]*Cada una de ellas utiliza servicios de las anteriores, de un nivel más bajo, y, a su vez, ofrece los suyos a capas de niveles superiores. Esta división facilita la tarea de los desarrolladores de aplicaciones, que disponen de las librerías para acceder a las funcionalidades necesarias (incluso de capas inferiores) sin necesidad de programar a bajo nivel. A continuación se detalla cada capa comenzando por la inferior y acabando en la superior:

- **Kernel de Linux:** Android basa su núcleo en el kernel de Linux versión 2.6 adaptado para su uso en dispositivos móviles y optimizado para conseguir un mayor rendimiento en este tipo de terminales.

El núcleo actúa como una capa de abstracción entre el hardware y el resto de capas de la arquitectura. El desarrollador no accede directamente a esta capa, sino que lo hace a través de librerías disponibles en capas superiores. Con esto conseguimos que los desarrolladores no necesiten conocer el hardware de cada dispositivo en particular.

El kernel también es el encargado de la gestión de recursos del terminal (energía, memoria, etc.) y del sistema operativo (procesos, elementos de comunicación, etc.).

- **Librerías:** esta capa se sitúa justo sobre el kernel y está compuesta por las bibliotecas nativas de Android, también conocidas como librerías. Están escritas en C o C++ y compiladas para la arquitectura hardware específica del dispositivo. Su objetivo es proporcionar funcionalidad a las aplicaciones, garantizando que las tareas se realicen de una forma más eficiente.

  Las librerías que más comúnmente se incorporan son OpenGL (motor gráfico), bibliotecas multimedia (formatos de audio, imagen y video), Webkit (navegador), SSL (cifrado de comunicaciones), FreeType (fuentes de texto), SQLite (base de datos), entre otras.

- **Entorno de ejecución**: algunas veces no es considerado una capa en sí mismo debido a que también está formado por librerías. El componente principal del entorno de ejecución es la máquina virtual Dalvik, de la que ya hemos hablado en esta memoria.
- **Framework de aplicaciones:** representa el conjunto de herramientas que utilizan las aplicaciones en el ejercicio de sus funciones. Está compuesta principalmente por librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual Dalvik. Encontramos las siguientes:

- **Activity Manager:** Se encarga de administrar la pila de actividades de nuestra aplicación (cada una de las pantallas que puede ver el usuario) así como su ciclo de vida.
- **Windows Manager:** Gestiona lo que se mostrará en pantalla. Su función principal es crear las superficies en la pantalla que serán ocupadas posteriormente por las actividades. Utiliza la librería SurfaceManager.
- **Content Provider:** Permite a las aplicaciones compartir datos entre ellas. Se encarga de encapsular para tener control sobre cómo se accede a la información.
- **Views:** Contiene las vistas de los elementos que nos ayudarán a construir las interfaces de usuario (GUI): botones, cuadros de texto, listas e, incluso, un navegador web o un visor de Google Maps.
- **Notification Manager:** Implementa un sistema de notificaciones mediante el cual las aplicaciones pueden comunicar eventos al usuario mostrando alertas en la barra de estado. Incorpora la posibilidad de incluir sonidos, vibraciones o iluminar los LEDs del teléfono si dispone de ellos.
- **Package Manager:** Esta biblioteca permite obtener información sobre los paquetes instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes. Entendiendo paquete como la forma en que se distribuyen las aplicaciones Android, estos contienen el archivo .apk, que a su vez incluye los archivos binarios .dex con todos los recursos y archivos adicionales que necesite la aplicación.
- **Telephony Manager:** Incluye todas las API vinculadas a las funcionalidades propias del terminal como teléfono, es decir, permite realizar llamadas o enviar y recibir SMS/MMS; sin embargo, no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.
- **Resource Manager:** Permite el manejo de todos los elementos que forman parte de la aplicación y que están fuera del código, es decir, cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos o layouts.
- **Location Manager:** Permite determinar la posición geográfica del dispositivo Android mediante GPS o redes disponibles y trabajar con mapas.
- **Sensor Manager:** Permite gestionar sensores incorporados en el hardware del teléfono (por ejemplo: acelerómetro, giroscopio, magnetómetro, sensor de luminosidad, termómetro, etc.)
- **Cámara:** Permite el uso de las cámaras del dispositivo y realizar fotografías y/o vídeo.
- **Multimedia:** Permite tratar con archivos multimedia en los formatos soportados (reproducir audio, vídeo e imágenes en el dispositivo).
    - **Aplicaciones:** es la capa de nivel superior y se encuentra por encima de todas las anteriores. Se incluyen en ella todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, las nativas (programadas en C o C++) y las

administradas (programadas en Java), las que vienen preinstaladas en el dispositivo y aquellas que el usuario ha instalado.

En esta capa encontramos también la aplicación principal del sistema, el launcher, que es la que permite ejecutar otras aplicaciones presentadas a través de una interfaz gráfica amigable para el usuario. Incluso el propio launcher es personalizable y el usuario podrá escoger aquel que más le agrade o que incluya las características de personalización que le satisfagan.

A continuación se puede ver un esquema de las capas explicadas *[5]*.

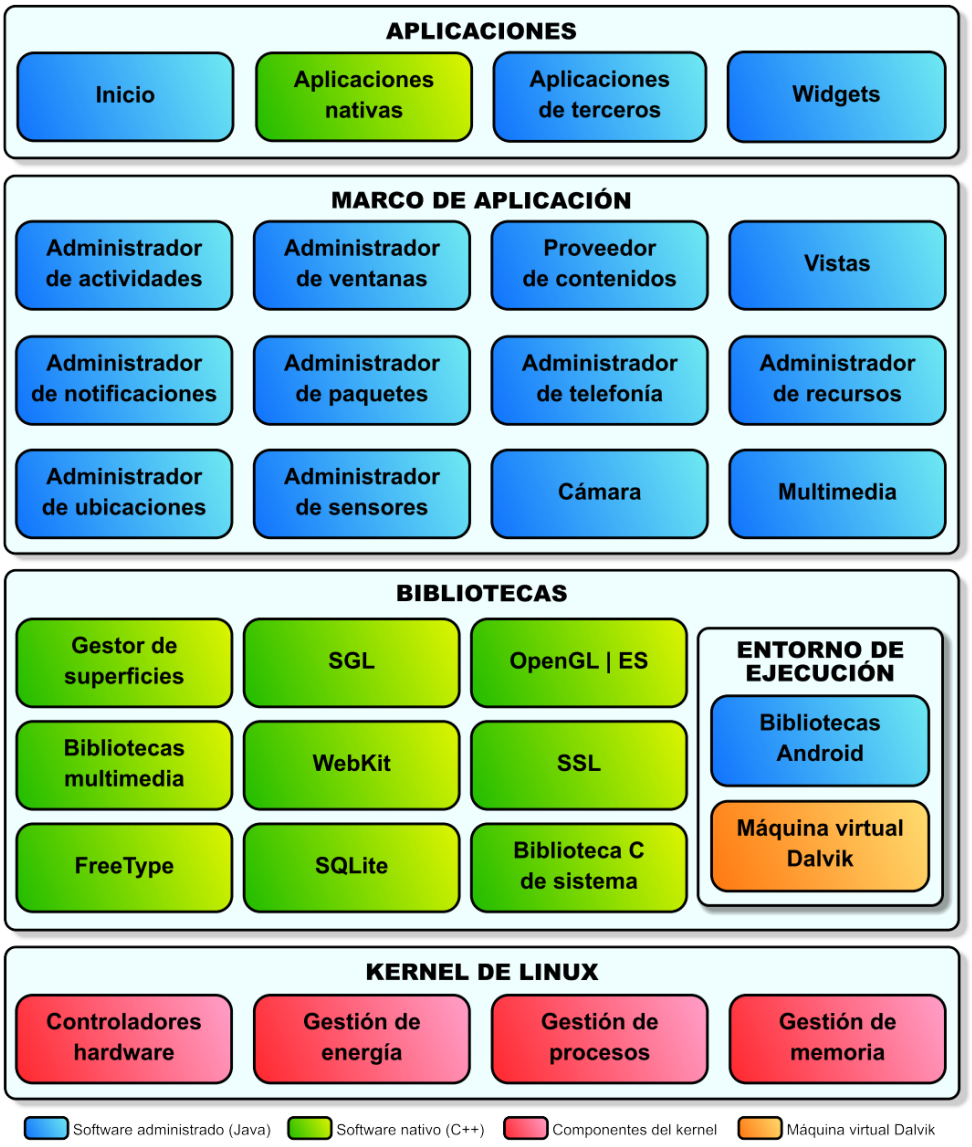

Ilustración 2 – Capas de Android

2.1.3. Características de una aplicación en Android

2.1.3.1. Estructura de una aplicación

La estructura de una aplicación Android está definida por la interacción de distintos componentes o bloques de construcción básicos. No es necesario que una aplicación los incluya todos, hará uso de las APIs que necesite. Normalmente, los componentes encargados de realizar cada tarea puedan ser manipulados o reemplazados sin problemas, asegurando la máxima flexibilidad. Todos estos componentes deben ser declarados de forma explícita en el fichero AndroidManifest.xml, del que hablaremos más adelante, donde se encuentran definidos otros datos importantes como los permisos, siendo un fichero básico en cualquier aplicación.

- **Activity (Actividad):** Éste es, sin duda, el bloque de construcción más utilizado. Como se ha mencionado anteriormente, una actividad es una tarea que se lleva a cabo en la aplicación y que tiene una interacción con el usuario, aunque sea únicamente en uno de los dos sentidos. Se puede comparar con una *ventana* en una aplicación de PC con entorno gráfico.

Las actividades implementan extendiendo de la clase "Activity" y cada una de ellas tiene su proceso de vida propio, que se analizará en el siguiente punto de la memoria. Lo normal es asignar una de las activities para que sea la principal y se muestre una vez entremos a la aplicación. La mayoría de las aplicaciones están compuestas de varias actividades. El paso de una pantalla a otra se consigue mediante la inicialización de una nueva actividad a través de un *Intent*, dejando a la actual en pausa dentro de la pila, lo que permite al usuario navegar entre actividades previamente abiertas.

Se puede definir un *Intent* como la intención de realizar una acción. Al lanzar un *Intent*, una aplicación puede llamar a otra aplicación distinta que sea capaz de realizar la acción solicitada o, en este caso, abrir una nueva actividad.

- **Service (Servicio):** Los servicios son componentes sin interfaz gráfica que se ejecutan en segundo plano. Pueden equipararse a los servicios o demonios presentes en otros sistemas operativos. Los servicios pueden realizar cualquier tipo de acción, por ejemplo actualizar datos, lanzar notificaciones no intrusivas, o incluso mostrar elementos visuales si se necesita en algún momento la interacción con del usuario para obtener una confirmación.

Un ejemplo para comprender el concepto de servicio podría ser el reproductor de música que viene incluido en el sistema operativo. Esta aplicación está dotada de una interfaz gráfica (o Activity) donde se permite al usuario elegir entre una lista de canciones, generar listas de reproducción así como un abanico de opciones extra. Sin embargo, en el momento en que el usuario comienza a reproducir un archivo de audio, puede seguir navegando entre las canciones e incluso comenzar a utilizar otras aplicaciones. Esto se debe a que la reproducción se realiza mediante un servicio y se ejecuta en segundo plano.

Al igual que ocurre con las actividades, se implementa mediante la clase con su mismo nombre.

- **Broadcast Receiver:** Se trata de un componente destinado a detectar y reaccionar ante mensajes o eventos globales del sistema u otras aplicaciones. Algunos ejemplos podrían ser mensajes avisando de una llamada entrante, un nuevo email, etc. También se puede emplear un *Intent* para que sea nuestra aplicación la que dirija mensajes a otros Broadcast Receivers activos.
- **Content Provider:** como se ha comentado anteriormente al hablar de la arquitectura de Android, su función es gestionar como se comparten datos entre aplicaciones. Permite compartir datos sin mostrar detalles sobre su almacenamiento interno, estructura o implementación. Al encapsular la información, ofrece una seguridad mayor. Existen varios tipos ya implementados en la librería correspondiente para compartir todo tipo de ficheros. Aún así, también está la opción de crearlos de forma personalizada.
- **AndroidManifest.xml:** es un fichero que incluye toda la información necesaria para ejecutar la aplicación. Se deben declarar todas los componentes anteriormente citados que usemos (cada una de las actividades, servicios, etc.). Es imprescindible también detallar los permisos sobre hardware que necesitaremos. El usuario los aceptará o no al instalar la aplicación, por ello deben estar sólo los necesarios, los que estén justificarlos. En el caso de este proyecto el permiso para usar la cámara y leer y escribir en la tarjeta de memoria (para guardar las fotografías). También se puede avisar de que será necesario usar ciertas características e indicar si son imprescindibles para el correcto funcionamiento de la aplicación o no, por ejemplo, el autofocus. En la siguiente página se muestra un ejemplo de Android Manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.GDsergio.picsallower"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.camera" />
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    />
    <uses-permission
        android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-feature
        android:name="android.hardware.camera.autofocus"
        android:required="false"/>

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/logo"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        android:hardwareAccelerated="true">

      <activity
        android:screenOrientation="landscape"
        android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
                  android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name="com.GDsergio.picsallower.PicsAllower"
            android:label="@string/app_name"
            android:screenOrientation="landscape">
        </activity>
        <activity android:name=".CannyConfig"></activity>
    </application>
</manifest>
```

2.1.3.2. Árbol de proyectos

Se explica a continuación la estructura de carpetas de un proyecto Android usando, en este caso, Eclipse como IDE (Integrated Development Enviroment).

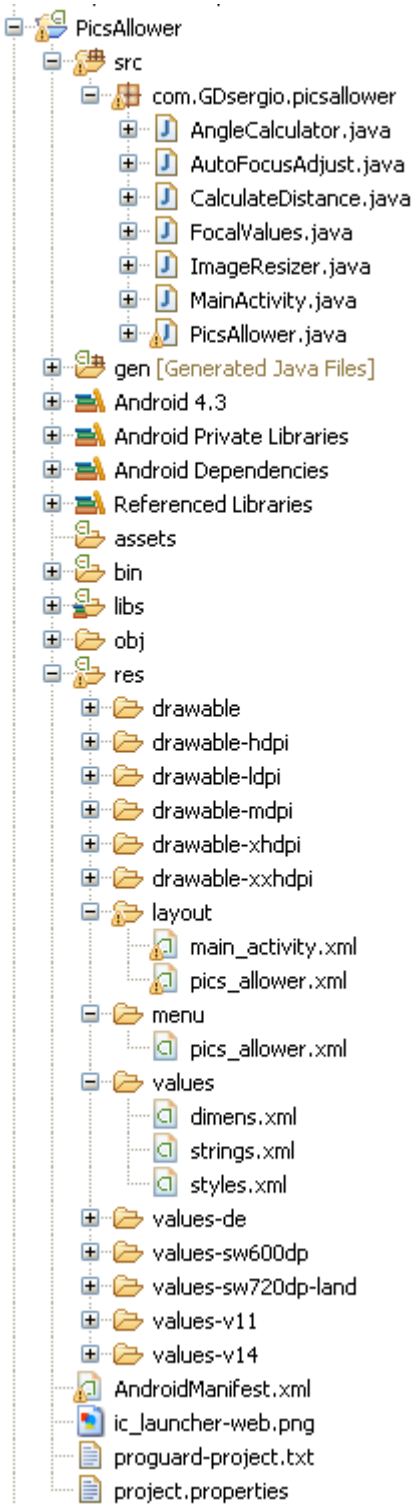

Ilustración 3 – Árbol de proyectos

Cabe destacar los siguientes directorios *[6]*:

- `/src/`: Contiene todo el código fuente de la aplicación, tanto de actividades como de clases auxiliares, etc.
- `/libs/`: Contiene las librerías que se usarán en la aplicación.
- `/res/`: Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, cadenas de texto (strings), etc. Como se puede observar, los diferentes tipos de recursos se distribuyen a su vez en diferentes carpetas.

Además, es imprescindible el fichero AndroidManifest.xlm, del que ya se ha hablado en el apartado anterior. Muchos de los ficheros que aparecen son generados automáticamente y es recomendable no modificarlos, sobre todo si no se es plenamente consciente de su utilidad y función.

2.1.3.3. Ciclo de vida de una actividad

En Android, cada aplicación se ejecuta en su propio proceso *[7]*. Esto aporta beneficios en cuestiones básicas como seguridad y gestión de recursos del sistema. Android se ocupa de lanzar y parar todos estos procesos, controlar su ejecución y decidir qué hacer en función de los recursos disponibles y de las órdenes dadas por el usuario.

Cada proceso, correspondiente a una aplicación, estará formado por una o varias actividades independientes. Cada actividad, como componente básico de Android, tiene un ciclo de vida bien definido; está en manos del desarrollador controlar en qué estado se encuentra en cada momento y programar las acciones que más le interesen según el caso.

Podemos ver el ciclo completo en la siguiente el esquema a continuación.

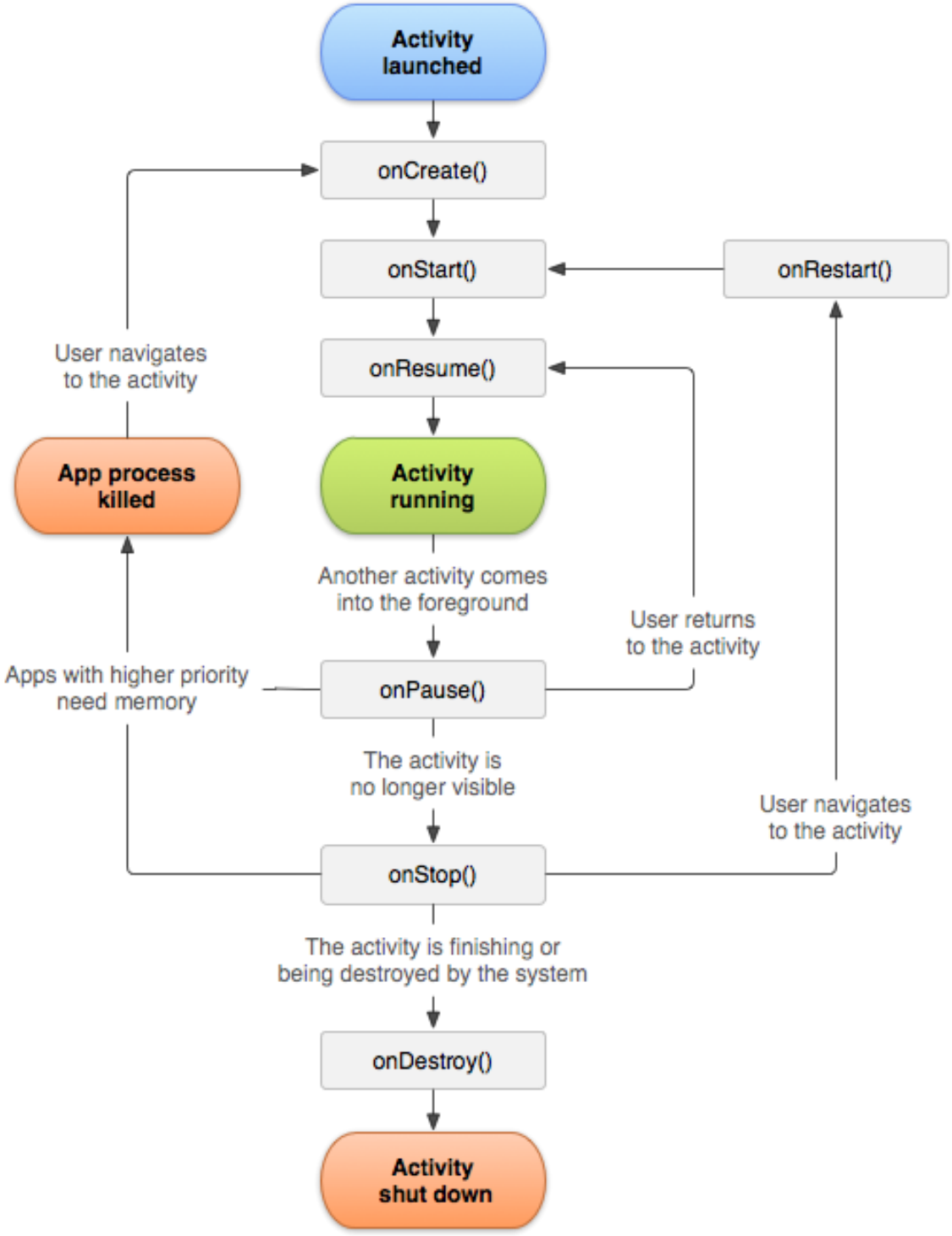

Ilustración 4 – Ciclo de una Activity en Android

2.2. OpenCV

A pesar de todo lo que ofrece Android, en ocasiones se necesitan herramientas adicionales. En este proyecto, OpenCV será una gran ayuda para el tratamiento de imágenes.

2.2.1. Definición

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Está publicada bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas *[8]*.

OpenCV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Además, se ha incorporado recientemente al terreno de los sistemas operativos móviles y, de momento, está disponible parcialmente (faltan ciertas funciones por implementar) para iOS y Android. Contiene una gran variedad de funciones que abarcan una amplia gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión robótica.

OpenCV pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Su programación se ha realizado en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multinúcleo.

2.2.2. JavaCV

Como se ha mencionado anteriormente, OpenCV está disponible tan solo de manera parcial en Android. Además, muchas funciones no están todavía desarrolladas con interfaz java y requieren complejos procesos de fusión de código (Java con C++) para funcionar.

JavaCV *[9]* es el nombre que adquiere un conjunto de librerías cuyo objetivo es tratar de solucionar los problemas de falta de funciones OpenCV en Android. Ofrece la posibilidad de seguir usando exclusivamente Java para llamar a ciertos métodos aún no disponibles de manera oficial en las librerías de OpenCV. Esto lo simplifica mucho el trabajo a la hora de desarrollar aplicaciones. Aún así, en ciertos aspectos sigue estando limitado frente a las posibilidades de OpenCV para PC.

2.2.3. Estructura

OpenCV está compuesto por varios módulos *[10]*. Los siguientes son los principales:

- *core*: contiene las estructuras básicas de OpenCV como, por ejemplo, Mat (estructura para representar matrices) y funciones básicas que utilizan el resto de los módulos.

- *imgproc*: módulo encargado del procesamiento de imágenes (filtrado, transformaciones geométricas, conversiones de color, histogramas etc.).
- *highgui*: para la gestión de entrada y salida y lo referente a la interfaz de usuario. A su vez, provee codecs para imagen y vídeo.
- *calib3d*: posee múltiples algoritmos de visión geometría, calibración de sonido y cámara, estimación de posición de objetos, algoritmos de correspondencia estéreo, reconstrucción 3D.
- *features2d*: posee detectores de características sobresalientes, descriptores y comparadores descriptores.
- *video*: módulo para el análisis de vídeos que incluye estimación de movimiento, sustracción de fondo y seguimiento de objetos.
- *objdetect*: módulo de detección de objetos e instancias de los tipos predefinidos como por ejemplo caras, ojos, tazas, personas, autos, entre otros.

- *ml*: módulo de aprendizaje automático.

- *gpu*: módulo de aceleración por GPU para los algoritmos de los demás módulos.

2.3. Image Stitching

Como se ha comentado con anterioridad, para el proyecto se necesitan imágenes que abarquen longitudes considerables y por ello se opta por las panorámicas. El proceso de unir varias fotos individuales en una panorámica es lo que se llama *image stitching* o stitching de imágenes. Para realizarlo se emplean las librerías de OpenCV.

A continuación se muestra la cadena de pasos a seguir basada en el documento "Automatic Panoramic Image Stitching using Invariant Features" publicado por Matthew Brown y David G. Lowe.

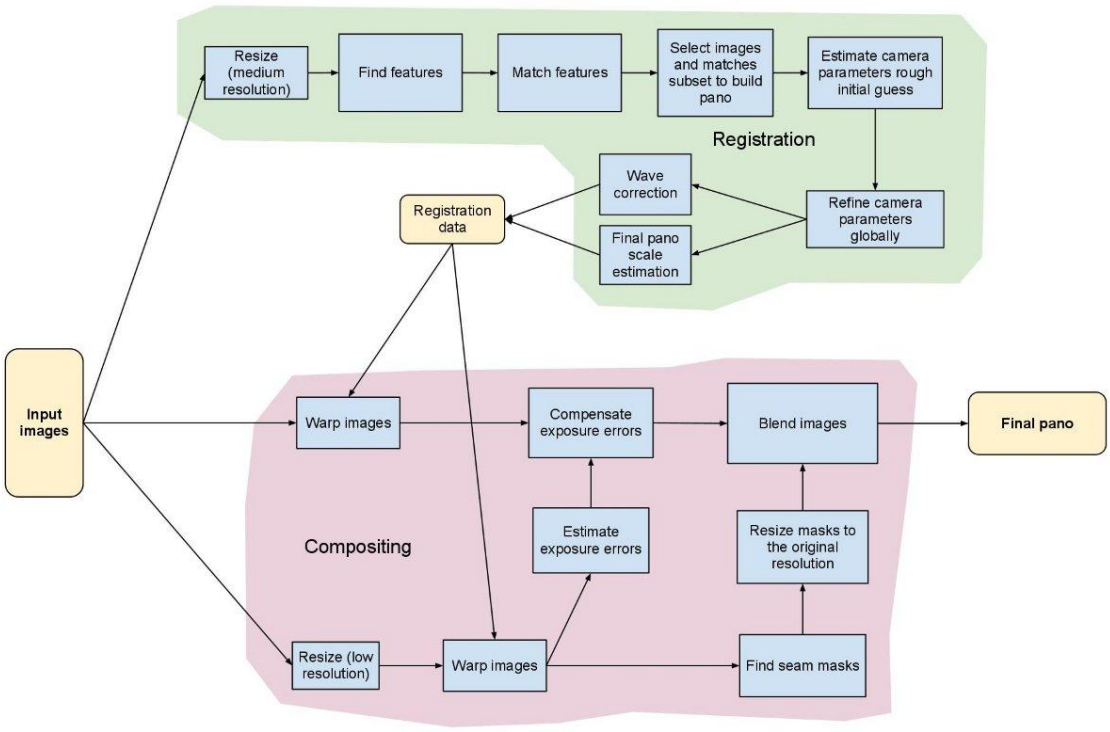

Ilustración 5 – Cadena de procesos para Image Stitching

2.3.1. Feature Finder

El primer paso para la formación de panorámicas es el reconocimiento de puntos de interés en dos imágenes contiguas. Se pueden utilizar diferentes algoritmos para la detección de estas características importantes.

En la versión para PC, OpenCV ofrece una gran cantidad de algoritmos según prime la velocidad o la perfección en la fotografía. En el proyecto empleamos el algoritmo STAR porque es el que mejores resultados nos ha proporcionado tras realizar pruebas con todos. Sin embargo, éste no está implementado para Android y, por tanto, en la versión móvil emplearemos SURF *[11]*.

2.3.2. Feature Matcher

El siguiente paso, una vez encontrados los puntos de interés es relacionarlos entre sí para identificar cuales se corresponden y así poder unir ambas imágenes. En este caso utilizamos el mismo algoritmo en ambas versiones (BestOf2NearestMatcher).

2.3.3. Bundle Adjuster

Detecta las posibles rotaciones existentes entre dos imágenes a enlazar y las iguala, haciendo que ambas tengan el ángulo apropiado para unirse.

2.3.4. Warper

Es la parte del proceso encargada de corregir posibles deformidades entre dos imágenes contiguas. Estas podrían producirse, por ejemplo, en los bordes de la imagen o al emplear distintos valores de zoom para cada una de las fotografías de la cadena. No obstante, en la aplicación detalla en esta memoria, esto último no se permite.

2.3.5. Exposure Compensator

Al tomar varias fotografías es posible que la luminosidad o brillo de la imagen no sea la misma en todas ellas. Esto puede ocurrir debido a la distinta posición del Sol, o de las fuentes lumínicas en general, con respecto a la cámara en cada una de las fotografías. Por ello se necesita un algoritmo que compense la exposición entre ambas imágenes. También es conocido como compensador de ganancia (gain compensator), debido a su modo de funcionamiento.

2.3.6. Blender

A pesar de haber utilizado el compensador de exposición, en ocasiones no se consigue que la intensidad entre píxeles contiguos sea completamente igual. Esto es particularmente notable en los bordes de las imágenes solapadas. En este paso se equilibran esas desigualdades.

A continuación podemos ver un ejemplo sobre el que se aplica compensación de ganancia/exposición y después el Blender (extraído del paper anteriormente mencionado).

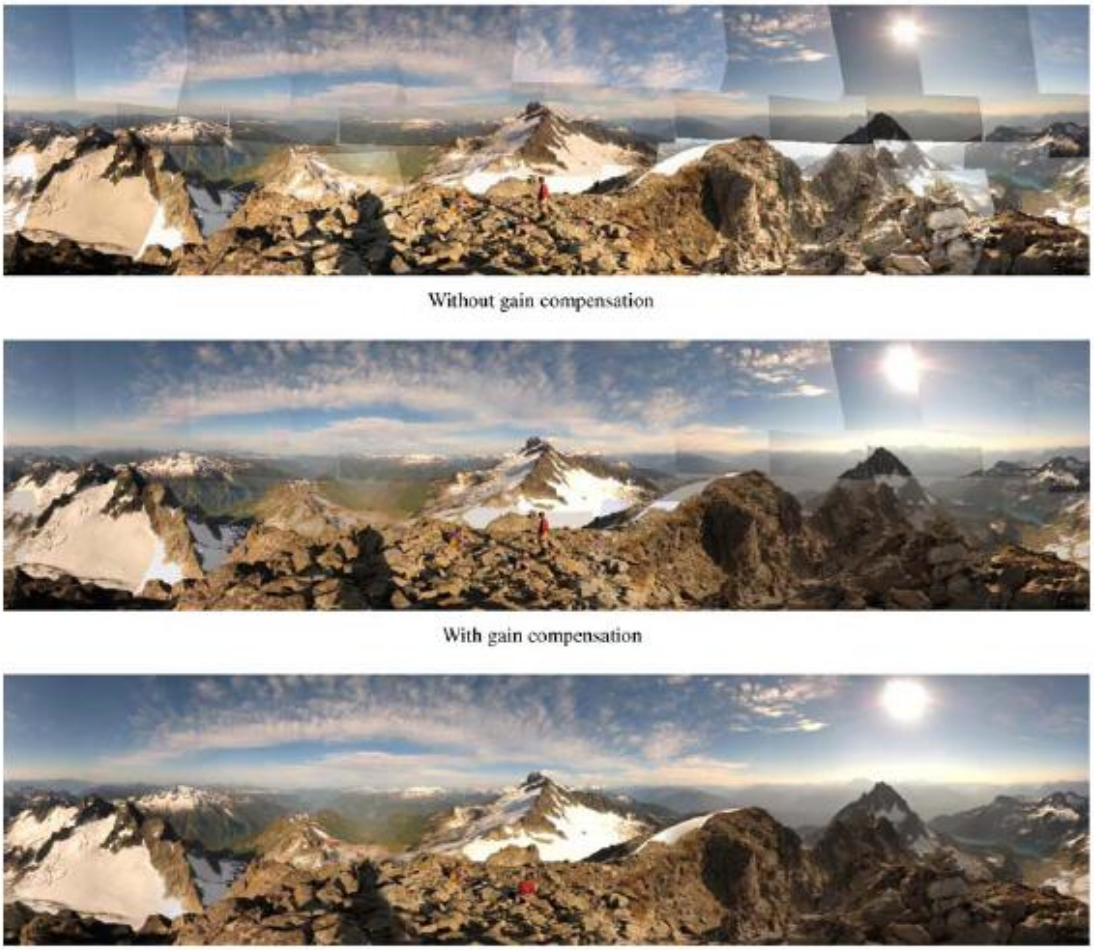

Ilustración 6 – Ejemplo gráfico de los resultados producidos mediante tratamiento de imagen

2.4. Hough Circles

Una vez realizado el stitching de las imágenes y obtenida la panorámica como resultado final se procede a detectar la circunferencia de los troncos de madera. Para ello empleamos la función Hough Circles de OpenCV. Esta función es una especialización de la transformada de Hough permitiendo detectar círculos en imágenes *[12]*.

OpenCV nos permite seleccionar distintos parámetros como los límites máximo y mínimo de tamaño de los círculos o la precisión que queremos para adaptar la función a nuestras necesidades *[13]*.

3. Diseño de la aplicación

En esta parte de la memoria se analizará el trabajo de diseño de la aplicación. En primer lugar, se presentan los requisitos de la aplicación tanto funcionales, como no funcionales y de implementación. Posteriormente, se presentarán los actores que influyen en la aplicación, y tras ello, los casos de uso de la aplicación. Una vez hecho esto, se mostrará el modelo de casos de uso de la aplicación.

3.1 Requisitos

La aplicación requerirá que el dispositivo contenga las siguientes funcionalidades:

- Una cámara que sea accesible por las aplicaciones.
- Sensores imprescindibles para el correcto funcionamiento de la aplicación, tales como el acelerómetro y el giroscopio, que serán empleados para la medición de la inclinación del terminal. La propia aplicación avisará al usuario si la inclinación no es la correcta. También conectará y desconectará los sensores cuando sea necesario, permanecerán apagados mientras la aplicación no esté usándose para ahorrar batería.

A continuación se describen los requisitos propios de la aplicación.

3.1.1. Requisitos funcionales

La aplicación será capaz de:

- Acceder a la cámara del dispositivo. La propia aplicación se encargará de acceder a ella cuando sea necesario para realizar las fotografías. Se liberará siempre cuando no se vaya a usar y también al salir de la aplicación, para que otras aplicaciones puedan tener acceso a ella.
- Acceder, habilitar y desconectar los sensores.
- Medir la distancia a un punto cualquiera en el suelo (normalmente la base de la pila de madera). El usuario será el encargado de apuntar a dicho punto guiado por unos ejes que aparecerán en la pantalla. Este dato se almacenará de forma interna y se utilizará para verificar que todas las fotos se están tomando a la misma distancia. La aplicación avisará si esto no es así para que el usuario se coloque en la posición correcta.
- Enfocar a un punto en concreto tocando la pantalla con un dedo. Esto permite ayudar al programa a enfocar correctamente la pila de madera, ya que en modo automático podría enfocar el fondo u otra zona que no es la de verdadero interés.
- Hacer zoom entre unos intervalos determinados. Los límites se fijarán para que el stitching de la imagen sea siempre viable y no cause deformidades.
- Mostrar una previsualización de la última fotografía tomada para ayudar al usuario a encontrar la posición correcta para la siguiente fotografía. Así el usuario conocerá cuando debe parar su desplazamiento en paralelo a la pila de madera.

- Permitir que el usuario tome las fotografías mediante un desplazamiento paralelo a la pila de madera, preferiblemente de izquierda a derecha. Estar en un punto fijo y simplemente rotar la cámara provoca ciertas distorsiones que impiden una correcta medida del volumen de madera una vez fusionadas las imágenes.
- Permitir que el usuario modifique el ISO de la imagen, así como la apertura y la velocidad de obturación del objetivo. De este modo antes de capturar la fotografía el usuario podrá elegir los valores adecuados por si considera que los ofrecidos por el modo automático no son satisfactorios.
- Guardar cada cadena de fotografías junto a la fusión de ellas (stitching) en una carpeta diferente. El usuario podrá acceder a ellas posteriormente tanto desde el explorador de archivos del dispositivo como conectando el terminal a un ordenador.
- Avisar al usuario si la fusión de las imágenes no ha podido realizarse para que repita las capturas. También avisará si el resultado ha sido satisfactorio.

3.1.2. Requisitos no funcionales

La aplicación deberá:

- Tener una interfaz sencilla y su uso debe ser intuitivo. Cada funcionalidad será fácil de entender y útil para el usuario.
- Ocupar la mínima memoria necesaria y consumirá de forma eficiente los recursos del sistema.

3.1.3. Requisitos de implementación

La aplicación deberá:

- Ser desarrollada para el sistema operativo Android, a partir de la versión 4.0 (Ice Cream Sandwich). La densidad y resolución de la pantalla no influirá en su correcto funcionamiento.
- Evitar ejecutarse en dispositivos Android que no dispongan de cámara o los sensores anteriormente mencionados.
- Ajustarse a los patrones de diseño de Android, presentes en el apartado de diseño de Android Developers.

3.2. Actores

En este apartado se verán los actores que interactuarán con la aplicación.

Actor	Usuario		Identificador	USU1	
Descripción	Persona usuaria de la aplicación desarrollada para terminales Android				
Características	Usuario de smartphone, tablet o cámara con sistema operativo Android 4.0 (Ice Cream Sandwich) o superior que disponga de la aplicación 'Picture Assistant'				
Relaciones					
Referencias					
Autor	Sergio Ferrer	Fecha	24/09/2014	Versión	v1

Atributos		
Nombre	Descripción	Tipo
Vacío	Vacío	Vacío

Comentarios
El usuario será genérico. No se tiene en cuenta su nivel de conocimiento acerca del sistema operativo ni sus diferentes funciones (cámara, sensores…). Tampoco se tendrá en cuenta su nivel de conocimientos de fotografía.

Actor	Usuario Básico		Identificador	USUBA1	
Descripción	Persona usuaria de la aplicación desarrollada para terminales Android, con conocimientos básicos				
Características	Usuario de smartphone, tablet o cámara con sistema operativo Android 4.0 (Ice Cream Sandwich) o superior que disponga de la aplicación 'Picture Assistant'				
Relaciones	Extiende de USU1				
Referencias					
Autor	Sergio Ferrer	Fecha	24/09/2014	Versión	v1

Atributos		
Nombre	Descripción	Tipo
Vacío	Vacío	Vacío

Comentarios
Se trata de un usuario que, por su nivel de conocimientos, no profundizará en todas las opciones de la aplicación. Empleará mayoritariamente el modo automático de la misma.

Actor	Usuario Avanzado		Identificador	USUAV1	
Descripción	Persona usuaria de la aplicación desarrollada para terminales Android, con conocimientos avanzados				
Características	Usuario de smartphone, tablet o cámara con sistema operativo Android 4.0 (Ice Cream Sandwich) o superior que disponga de la aplicación 'Picture Assistant'				
Relaciones	Extiende de USUBA1				
Referencias					
Autor	Sergio Ferrer	Fecha	24/09/2014	Versión	v1

Atributos		
Nombre	Descripción	Tipo
Vacío	Vacío	Vacío

Comentarios
Se trata de un usuario que, por su nivel de conocimientos, podrá emplear tanto el modo automático de la aplicación como el personal. Podrá variar los parámetros que considere adecuados para la obtención de mejores resultados.

3.3. Casos de uso

Caso de Uso	Arranque de la aplicación			ARAP1	
Actores	Usuario básico				
Tipo	Primario				
Referencias					
Precondición	La aplicación no debe estar iniciada				
Postcondición	La aplicación se habrá ejecutado y estará lista para su uso				
Autor	Sergio Ferrer	Fecha	25/09/2014	Versión	v1
Propósito					
Ejecutar la aplicación 'Picture Assistant'					
Resumen					
Cuando el usuario presiona el icono de la aplicación, ésta se inicia. Internamente, inicializará todas las variables necesarias y verificará que el terminal disponga de la cámara y los sensores adecuados. Aparece la pantalla inicial de la aplicación, dónde se pregunta al usuario si desea hacer una medición de la distancia antes de empezar el proceso de toma de fotografías.					
Curso Normal (Básico)					
1	El usuario presiona el icono para iniciar la aplicación				
2	Se muestra la pantalla inicial de la aplicación				
3	La aplicación queda a la espera de respuesta del usuario sobre si desea medir la distancia antes de empezar a capturar fotografías				
Curso Alternos					

Caso de Uso	Medición de distancia inicial			MEDDISIN1	
Actores	Usuario básico				
Tipo	Primario				
Referencias					
Precondición	La aplicación debe haberse ejecutado y el usuario haber presionado el botón de afirmación a la pregunta de si desea medir la distancia				
Postcondición	Se dispondrá de una medida interna de la distancia inicial				
Autor	Sergio Ferrer	Fecha	25/09/2014	Versión	v1
Propósito					
Medir la distancia inicial a la que el usuario se encuentra de la pila de madera.					
Resumen					
En primer lugar, la aplicación activa el registro de los datos de los sensores de manera transparente al usuario. A la vez, se muestra al usuario una pantalla con instrucciones para hacer una correcta medición de la distancia inicial (cómo colocarse, a dónde apuntar con el objetivo de la cámara, etc.). El usuario presionará OK para empezar con la medición y aparecerán unos ejes en el centro de la pantalla, los cuales tiene que alinear con la base de la pila de madera. En el momento en que considere adecuado, presionará el botón que aparece en pantalla para guardar la medida. La aplicación desconectará los sensores con la medida guardada y pasará al siguiente paso, la toma de fotografías.					
Curso Normal (Básico)					
1	Se muestran las instrucciones en la pantalla y se activa el registro de los datos provenientes de los sensores (giroscopio y acelerómetro)				
2	El usuario manifiesta su conformidad presionando el botón ‘OK’				
3	La aplicación muestra unos ejes en la pantalla y se empieza a mostrar en pantalla lo que ve la cámara				
4	El usuario alinea los ejes con la base de la pila de madera y presiona el botón de guardar medida				
5	La aplicación captura las medidas de los sensores y almacena estos datos internamente				
Curso Alternos					
1b	El usuario decide que prefiere no hacer una medición inicial de la distancia y presiona el botón ‘Atrás/Back’. La aplicación volverá al paso anterior donde se pregunta al usuario si desea medir la distancia				

Caso de Uso	Comprobación de distancia			COMPDIS1	
Actores	Usuario básico				
Tipo	Primario				
Referencias					
Precondición	Debe haberse medido la distancia inicial y tomado al menos una fotografía				
Postcondición	Se conocerá si se está a la distancia correcta				
Autor	Sergio Ferrer	Fecha	25/09/2014	Versión	v1
Propósito					
Comprobar si el usuario está a una distancia adecuada de la pila de madera con respecto a la distancia inicial.					
Resumen					
En primer lugar, la aplicación activa el registro de los datos de los sensores de manera transparente al usuario. Se muestra al usuario una pantalla con los ejes que debe ajustar a la base de la pila de madera. Habrá un botón en la esquina superior derecha. Éste será verde y contendrá la palabra 'OK' en el caso de encontrarse a la distancia correcta. Por el contrario, será rojo si la distancia no es la adecuada. El usuario podrá avanzar o retroceder para llegar al punto idóneo con respecto a la pila de madera. Al ser una función meramente informativa el usuario podrá continuar con el proceso de toma de fotografías en ambos casos presionando el botón mencionado (asumiendo el riesgo de que las mismas no sean válidas en el caso de no estar a la distancia correcta).					
Curso Normal (Básico)					
1	Se muestra la imagen que ve la cámara y sobre ella unos ejes en la pantalla para alinear con la base de la pila de madera				
2	El usuario comprueba si está a la distancia adecuada y, en caso negativo, lo soluciona modificando su posición respecto a los troncos				
3	El usuario presiona el botón cuando está conforme				
Curso Alternos					
1b	El usuario decide que prefiere no hacer una comprobación de la distancia y presiona el botón 'Atrás/Back'. La aplicación volverá a la toma de fotografías.				

Caso de Uso	Toma de la primera fotografía			TOFO1	
Actores	Usuario básico				
Tipo	Primario				
Referencias					
Precondición	Debe haberse arrancado la aplicación				
Postcondición	Se habrá capturado la primera imagen				
Autor	Sergio Ferrer	Fecha	15/04/2014	Versión	v1
Propósito					
Capturar la primera imagen del grupo con el que luego se realizará el stitching					
Resumen					
Se muestra al usuario una pantalla con la vista de la cámara y dos líneas rojas que delimitan el margen superior e inferior donde debe encontrarse la pila de madera (se comprobó que las zonas muy cercanas al borde de la imagen pueden quedar descartadas al hacer el stitching y esto evita ese problema). El usuario colocará la pila de madera en la posición adecuada y presionará el botón físico de captura de fotografías para capturar la primera instantánea.					
Curso Normal (Básico)					
1	Se muestra la imagen que ve la cámara y sobre ella unas líneas rojas delimitando el espacio correcto donde se encontrará la pila de madera				
2	El usuario coloca la pila de madera entre los márgenes				
3	El usuario presiona el botón para capturar la fotografía				
Curso Alternos					
1b	El usuario decide que prefiere no hacer una capturar la fotografía y presiona el botón ‘Atrás/Back’. La aplicación volverá al punto de partida.				

Caso de Uso	Toma de las demás fotografías			TOFO2	
Actores	Usuario básico				
Tipo	Primario				
Referencias					
Precondición	Debe haberse tomado la primera fotografía				
Postcondición	Se tendrán todas las imágenes con las que realizar el stitching				
Autor	Sergio Ferrer	Fecha	15/04/2014	Versión	v1
Propósito					
Capturar el resto de imágenes que compondrán el con el que luego se realizará el stitching					
Resumen					
Tal y como se hacía en el caso de la primera fotografía, se muestra al usuario una pantalla con la vista de la cámara y dos líneas rojas que delimitan el margen superior e inferior donde debe encontrarse la pila de madera. Además el usuario verá en la parte izquierda de la pantalla la parte derecha de la última fotografía tomada de forma translúcida (suponiendo que el desplazamiento que realiza el usuario para capturar la serie de fotografías es de izquierda a derecha). El usuario colocará la pila de madera en la posición adecuada con respecto a los márgenes rojos y a la fotografía inmediatamente anterior que aparece mostrada a la izquierda y presionará el botón físico de captura de fotografías para capturar la siguiente instantánea.					
Curso Normal (Básico)					
1	Se muestra la imagen que ve la cámara y sobre ella unas líneas rojas delimitando el espacio correcto donde se encontrará la pila de madera. También se ve una porción de la fotografía anterior.				
2	El usuario coloca la pila de madera entre los márgenes encuadrándola con la parte derecha de la fotografía anterior				
3	El usuario presiona el botón para capturar la fotografía				
Cursos Alternos					
1b	El usuario decide que prefiere no hacer una capturar la fotografía y presiona el botón ‘Atrás/Back’. La aplicación volverá al punto de partida.				

Caso de Uso	Fin de la toma de fotografías			FINFOT1	
Actores	Usuario básico				
Tipo	Primario				
Referencias					
Precondición	Deben haberse tomado todas las fotografías necesarias				
Postcondición	Se obtendrá la panorámica formada por la unión de las fotografías tomadas				
Autor	Sergio Ferrer	Fecha	15/04/2014	Versión	v1
Propósito					
Obtener una fotografía panorámica mediante el stitching de las fotos capturadas durante el proceso anteriormente comentado					
Resumen					
Una vez tomada la última fotografía de la serie el usuario presionará el botón Menú seguido de Finalizar Proyecto. La aplicación mostrará en pantalla el número total de fotografías tomadas por el usuario y pasará a realizar el stitching de forma transparente para éste, que puede comenzar con un nuevo proyecto si lo desea.					
Curso Normal (Básico)					
1	El usuario selecciona finalizar el proyecto				
2	La aplicación muestra por pantalla la cantidad de fotografías tomadas				
3	La aplicación comienza el proceso de Stitching				

Caso de Uso	Enfocar			ENF1	
Actores	Usuario básico				
Tipo	Primario				
Referencias					
Precondición	Estar en mitad de la toma de fotografías				
Postcondición	La cámara enfocará el punto deseado				
Autor	Sergio Ferrer	Fecha	15/04/2014	Versión	v1
Propósito					
Enfocar correctamente la imagen para facilitar el proceso de stitching					
Resumen					
Aprovechando la tecnología táctil se ha implementado esta función para que la cámara enfoque el punto de la imagen que toquemos con el dedo en la pantalla. Normalmente el usuario no necesitará enfocar ya que el autofocus funciona de manera bastante precisa. Pero si es necesario bastará con tocar el punto de la pantalla donde aparece la pila de troncos para que éste se oriente.					
Curso Normal (Básico)					
1	Se muestra la imagen que ve la cámara				
2	El usuario toca en la pantalla el punto que desea enfocar				
3	La cámara enfoca el punto elegido por el usuario				

Caso de Uso	Zoom			ZOOM1	
Actores	Usuario básico				
Tipo	Primario				
Referencias					
Precondición	Estar en mitad de la toma de fotografías				
Postcondición	Se dispondrá de una imagen más cerca o lejana (zoom in//zoom out)				
Autor	Sergio Ferrer	Fecha	15/04/2014	Versión	v1
Propósito					
Conseguir acercar o alejar la imagen					
Resumen					
En ocasiones, por circunstancias del terreno, tendremos que tomar las fotografías desde un punto más alejado. Por ello, la aplicación incluye la función de zoom, para poder acercar y conseguir capturar solo lo que necesitamos y no los alrededores. El usuario realizará el zoom con los botones físicos del dispositivo destinados a ello.					
Curso Normal (Básico)					
1	Se muestra la imagen que ve la cámara				
2	El usuario regula el zoom mediante los botones físicos del dispositivo				

Caso de Uso	Modificación trasparencia/preview			MODTRASP1	
Actores	Usuario básico				
Tipo	Primario				
Referencias					
Precondición	Haber tomado al menos una fotografía				
Postcondición	Variará el nivel y/o color de la transparencia/preview				
Autor	Sergio Ferrer	Fecha	15/04/2014	Versión	v1
Propósito					
Hacer más fácil para el usuario la colocación de la cámara para la correcta toma de fotografías					
Resumen					
Debido a la homogeneidad de las fotografías en ocasiones es complicado diferenciar lo que está viendo la cámara y la previsualización de un porcentaje de la última fotografía para cuadrar la nueva. Con esta función se permite al usuario dar mayor o menor nivel de transparencia a la porción de imagen anterior mostrada, así como mostrarla de un color diferente si se desea para poder diferenciarla más fácilmente del fondo.					
Curso Normal (Básico)					
1	Se muestra la imagen que ve la cámara				
2	El usuario desplaza la barra para ajustar el nivel de transparencia				
3	El usuario puede presionar el botón de cambio de color				
4	El usuario presiona ‘OK’ para confirmar los cambios				
5	La aplicación vuelve a la toma de fotografías con los parámetros cambiados				
Cursos Alternos					
1b	Si el usuario decide que prefiere no hacer ningún cambio y presiona el botón ‘Atrás/Back’ la aplicación volverá a la toma de fotografías sin cambiar ningún parámetro de la transparencia.				

Caso de Uso	Modificación del ISO			MODISO1	
Actores	Usuario avanzado				
Tipo	Primario				
Referencias					
Precondición	Estar en toma de fotografías				
Postcondición	Habrá cambiado el ISO para futuras fotografías				
Autor	Sergio Ferrer	Fecha	15/04/2014	Versión	v1
Propósito					
Permitir al usuario un mayor control sobre la luminosidad de la imagen					
Resumen					
El usuario podrá variar el ISO de las fotografías a tomar para conseguir el nivel idóneo si no está satisfecho con el ISO automático. Un ISO menor reducirá el ruido a costa de una menor luminosidad y viceversa. El usuario podrá moverse en el intervalo 100 – 1600.					
Curso Normal (Básico)					
1	Se muestra la imagen que ve la cámara				
2	El usuario desplaza la barra para ajustar el nivel del ISO				
3	El usuario presiona ‘OK’ para confirmar los cambios				
Cursos Alternos					
1b	Si el usuario decide que prefiere no hacer ningún cambio y presiona el botón ‘Atrás/Back’ la aplicación volverá a la toma de fotografías sin cambiar ningún parámetro				

Caso de Uso	Modificación de la apertura			MODAPER1	
Actores	Usuario avanzado				
Tipo	Primario				
Referencias					
Precondición	Estar en toma de fotografías				
Postcondición	Se habrá cambiado la apertura del objetivo para futuras fotografías				
Autor	Sergio Ferrer	Fecha	15/04/2014	Versión	v1
Propósito					
Permitir al usuario un mayor control sobre la cantidad de luz que capta la cámara					
Resumen					
El usuario podrá modificar la apertura del objetivo entre unos valores que dependerán del zoom en el momento de hacerlo. Números bajos permitirán una apertura mayor y más entrada de luz y números altos cerrarán el objetivo y conseguirán fotos más oscuras.					
Curso Normal (Básico)					
1	Se muestra la imagen que ve la cámara				
2	El usuario desplaza la barra para ajustar el nivel de la apertura				
3	El usuario presiona ‘OK’ para confirmar los cambios				
Cursos Alternos					
1b	Si el usuario decide que prefiere no hacer ningún cambio y presiona el botón ‘Atrás/Back’ la aplicación volverá a la toma de fotografías sin cambiar ningún parámetro				

Caso de Uso	Modificación de la velocidad de obturación			MODVEL1	
Actores	Usuario avanzado				
Tipo	Primario				
Referencias					
Precondición	Estar en toma de fotografías				
Postcondición	Se habrá cambiado la velocidad de obturación para futuras fotografías				
Autor	Sergio Ferrer	Fecha	15/04/2014	Versión	v1
Propósito					
Permitir al usuario un mayor control sobre la cantidad de luz que capta la cámara					
Resumen					
El usuario podrá modificar la velocidad de obturación para elegir que el objetivo permanezca más o menos tiempo abierto a la hora de capturar la instantánea. Cuanto más tiempo permanezca abierto, más luminosas serán las fotografías, pero también habrá un mayor riesgo de falta de nitidez si no se emplea trípode, ya que cualquier pequeño movimiento de la cámara será capturado por el sensor.					
Curso Normal (Básico)					
1	Se muestra la imagen que ve la cámara				
2	El usuario desplaza la barra para ajustar el nivel de la apertura				
3	El usuario presiona ‘OK’ para confirmar los cambios				
Cursos Alternos					
1b	Si el usuario decide que prefiere no hacer ningún cambio y presiona el botón ‘Atrás/Back’ la aplicación volverá a la toma de fotografías sin cambiar ningún parámetro				

3.3.1. Modelo de casos de uso

Una vez vistos los autores y casos de uso de la aplicación se puede mostrar el diagrama que representa este modelo de casos de uso.

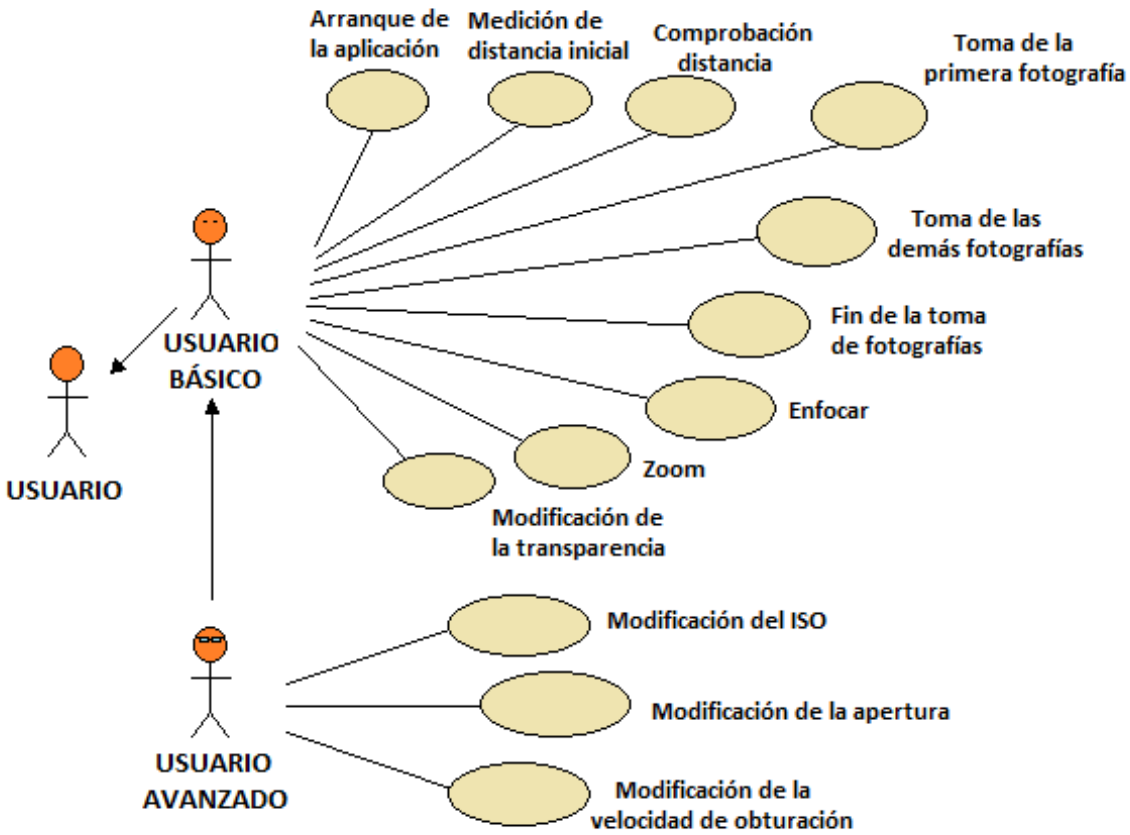

Ilustración 7 – Modelo de casos de uso

4. Aplicación desarrollada

En este apartado se analizará la estructura de la aplicación y su estructura de clases como paso posterior a la definición de requisitos y casos de uso. También se mostrarán capturas de pantalla de la aplicación y cómo se va de un punto a otro. Para ello se estructurará de aquí en adelante según las distintas carpetas del proyecto.

4.1. Carpeta ./src

Es la carpeta principal donde se incluirá todo el código funcional de la aplicación.

4.1.1. MainActivity.java

Es la primera actividad (pantalla) y se abre al lanzar la aplicación. En ella únicamente diremos si queremos medir la distancia a la pila de madera antes de empezar el proceso y se pasará a la siguiente actividad.

4.1.2. PicsAllower.java

Es el archivo más importante de la aplicación. En este fichero se incluye la mayor parte del código y es el encargado de llamar a las funciones externas escritas en otros archivos cuando sea necesario para hacer todas las operaciones.

Al comienzo se definirán todos los imports necesarios para el correcto funcionamiento de la aplicación, tanto los propios de Android y Java como los necesarios para poder utilizar OpenCV.

La clase principal, llamada PicsAllower como el fichero, es un extend de una Activity (necesario para todas las clases que se quieran visualizar por pantalla) y hace implement SensorEventListener (necesario para poder emplear los sensores a la hora de calcular inclinación de la cámara).

```
public class PicsAllower extends Activity implements SensorEventListener {
```

Dentro de esta clase encontramos distintos métodos principales:

- **onCreate**: se llamará a este método nada más arrancar la aplicación. Se encargará de definir algunas FLAG para el sistema, inicializar las variables y definir varios parámetros para todas las herramientas que se utilizarán.
- **onPause**: será llamado al salir de esta actividad. Será el encargado de liberar la cámara para que pueda ser utilizada por otras aplicaciones, así como finalizar ciertas funciones (toasts en pantalla, etc.)
- **onResume**: este método es llamado al entrar por primera vez a la actividad (después de pasar por onCreate) o si se vuelve a ella después de haber salido.

- **onCreateOptionsMenu** y **onOptionsItemSelected**: encargados de crear el menú y definir las acciones que se realizarán al seleccionar sus elementos respectivamente.

También se han implementado otros métodos con distintas funcionalidades para la aplicación (suelen ser llamados desde los métodos principales):

- **releaseCamera**: elimina la retención que realiza nuestra aplicación sobre el uso de la cámara para que pueda ser utilizada por otras.
- **checkCameraHardware**: comprueba que el dispositivo en el que se ha ejecutado la aplicación contiene una cámara.
- **getCameraInstance**: se encargará de abrir la cámara conseguir una instancia que le haga referencia para poder utilizarla posteriormente.
- **PictureCallback**: será la función que maneje los Bitmaps y se encargue de guardar las imágenes tomadas.
- **getOutputMediaFile**: se encargará de crear un fichero para guardar la imagen y los directorios previos a éste si son necesarios.
- **CameraPreview**: se dedicará a mostrar por pantalla lo que se va viendo por la cámara de fotos
- **initListeners**: se encarga de inicializar los listeners de los sensores del acelerómetro, giroscopio y campo magnético.
- **onSensorChange**: realiza la función indicada según el cambio que se haya producido en un determinado sensor.
- **displayToast** y **displayToastLong**: métodos destinados a mostrar algunos mensajes por pantalla. Se utilizará uno u otro según la duración que queramos que tengan los mismos.
- **changeColor**: cambia el color de lo que está captando la cámara por uno negativizado si el ángulo en el que está la cámara no es adecuado para la correcta toma de fotografías.
- **visibilityFunction**: se encarga de adecuar el aspecto de la pantalla según en qué estado se encuentre la aplicación.
- **ImageStitching**: método encargado de hacer la fusión de imágenes utilizando las librerías de OpenCV.
- **detectCircles**: se encarga de detectar la circunferencia de los troncos de las pilas de madera mediante el uso de diferentes filtros y la función de Hough.

Ilustración 8 – Ejemplo de detección de círculos

Además de los métodos anteriores hay unos cuantos que serán llamados cuando suceda un cierto evento (tocar la pantalla, presionar un botón, etc.). A continuación se explica los que están siendo usados en la aplicación:

- **onTouchEvent**: detecta cuando se toca la pantalla y si está en la fase de toma de fotografías llamará al método encargado de enfocar el punto tocado con el dedo (tal y como se vio en los casos de uso).
- **onKeyDown**: detecta cuando un botón es presionado. En este caso tenemos diferentes casos. Por un lado el botón de atrás, que permitirá volver a un punto anterior en la aplicación o no, según en qué estado se encuentre. Por ejemplo, cuando se han tomado varias fotografías hay que finalizar primero el proyecto antes de salir (por seguridad ya que a veces se perdía todo lo hecho por error). Luego tenemos el botón de cámara que capturará una instantánea al ser presionado. También encontramos los botones encargados del zoom.
- **onKeyUp**: detecta cuando se deja de presionar un botón. Es útil cuando hacemos un zoom largo, manteniendo mucho tiempo presionado el botón correspondiente.

4.1.3. AngleCalculator.java

Es la clase que contiene los métodos encargados de calcular la inclinación y rotación de la cámara a partir de la información proporcionada por los sensores. También tiene un método encargado de verificar con los datos si la posición de la cámara es la adecuada para un resultado óptimo a la hora de hacer el stitching de las fotografías.

4.1.4. AutoFocusAdjust.java

El propósito de esta clase es proveer de una función útil para ajustar el foco automáticamente en el punto que se toque de la pantalla.

4.1.5. CalculateDistance.java

Esta clase es la contenedora de los métodos encargados de calcular la distancia entre la cámara y la base de la pila de madera. Utiliza funciones trigonométricas para

posteriormente comparar la distancia actual con el valor de referencia calculado al principio de la captura de fotografías.

4.1.6. CannyConfig.java

Esta clase permite modificar la configuración del modo de visualización llamado Canny. Es un modo que permite ver en la imagen solamente los perfiles de los objetos capturados para la cámara. Finalmente se dejó oculto para el usuario ya que no le aporta ningún beneficio, pero ayudó a la hora de desarrollar la aplicación a entender mejor cómo funcionaban las librerías OpenCV a la hora de detectar los círculos de los troncos, pudiendo así conseguir mejores resultados.

4.1.7. FocalValues.java

Según el zoom aplicado a la hora de tomar una fotografía hay distintas distancias focales posibles. Esta clase es la encargada de asociar un determinado zoom con cierta distancia focal y cambiar de una a otra si el usuario lo desea.

4.1.8. ImageResizer.java

Es la clase encargada de ajustar los tamaños de las imágenes para tener al final una panorámica con las medidas apropiadas.

4.2. Carpeta ./res

Esta carpeta contiene los recursos que se utilizarán en la aplicación.

En primer lugar están las carpetas denominadas drawable, que contienen las imágenes utilizadas en la aplicación en sus distintas resoluciones.

En segundo lugar se encuentran los layouts que definirán el aspecto gráfico de cada pantalla (activity).

En tercer lugar aparece una carpeta destinada a guardar los ficheros XML de los menús.

Por último, hay unas carpetas denominadas values que contienen también ficheros XML con la información sobre el tema utilizado en la aplicación o, por ejemplo, las cadenas de texto que se utilizan en los distintos puntos de la misma (esto es una gran ventaja a la hora de traducir la aplicación a diversos idiomas, ya que bastará con tener una carpeta para cada idioma y la cadena traducida en ese idioma; evitando tener que modificar layouts o crear algunos innecesarios).

4.3. Carpeta ./libs

Es la carpeta que contiene las distintas librerías que usamos en la aplicación. Se incluyen aquí durante el desarrollo del proyecto las necesarias de OpenCV, java y el propio Android.

4.4. AndroidManifest.xml

Como ya se explicó en el apartado de conceptos teóricos, este fichero contiene toda la información necesaria para el arranque de la aplicación. En él se declaran los servicios, actividades y demás que vayamos a usar. También se especifica el hardware que debe disponer el dispositivo y los permisos que necesita la aplicación para realizar sus tareas (usar la cámara, leer y escribir en la tarjeta SD, etc.).

4.5. Diseño de la interfaz gráfica (capturas)

En este apartado se mostrará el diseño elegido para la interfaz gráfica, para ello analizaremos las distintas Activities y el paso de una a otra a través de capturas de pantalla. También se mostrarán algunos detalles implementados para facilitar el uso de la aplicación a sus usuarios.

4.5.1. Logo

Como logo para la aplicación se ha elegido uno que sea representativo. Puesto que la aplicación será empleada para capturar panorámicas de grandes pilas de madera y luego calcular su volumen se ha elegido una de estas pilas, junto con el nombre de la empresa, como logo de la app.

Ilustración 9 – Logo de la aplicación

4.5.2. Nuevo proyecto

Una vez arrancada la aplicación, se empezará un nuevo proyecto. Nada más comenzar se mostrará una pantalla preguntando al usuario si desea hacer una medida de la distancia inicial (a la pila de madera).

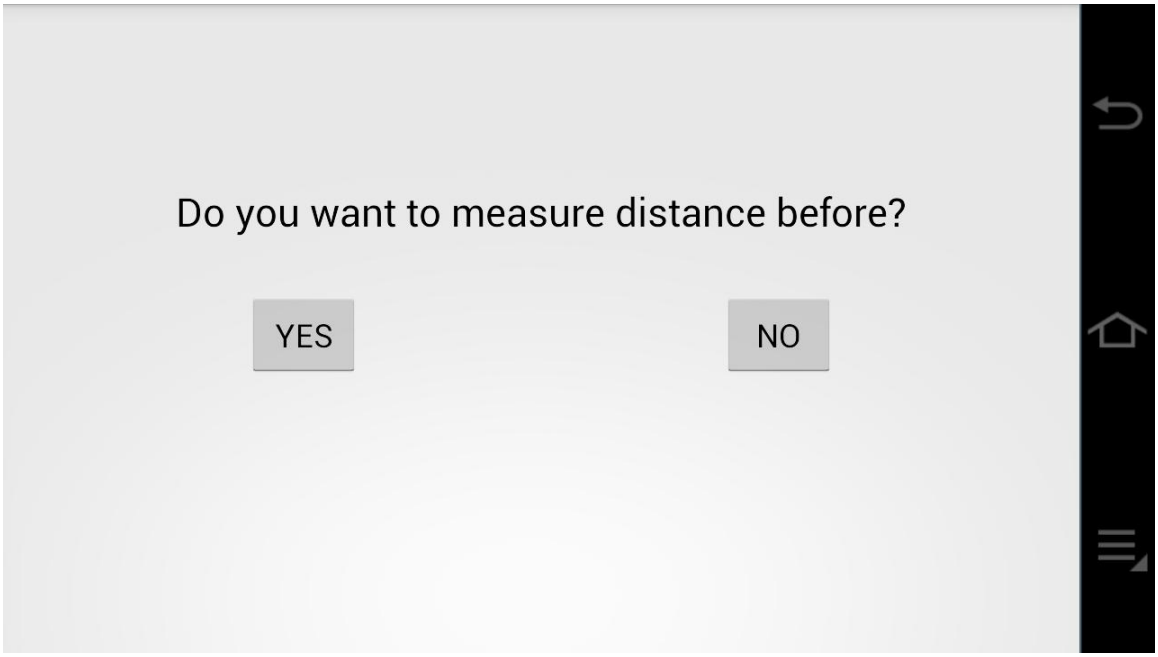

Ilustración 10 – Pantalla: Inicio de un nuevo proyecto

En caso afirmativo se pasará al siguiente punto (4.5.3), en caso negativo se irá a la pantalla del punto 4.5.5 para empezar con la toma de la primera fotografía.

4.5.3. Instrucciones para medir la distancia

Si el usuario ha decidido que quiere medir la distancia la aplicación mostrará unas instrucciones básicas para la correcta medición de ésta.

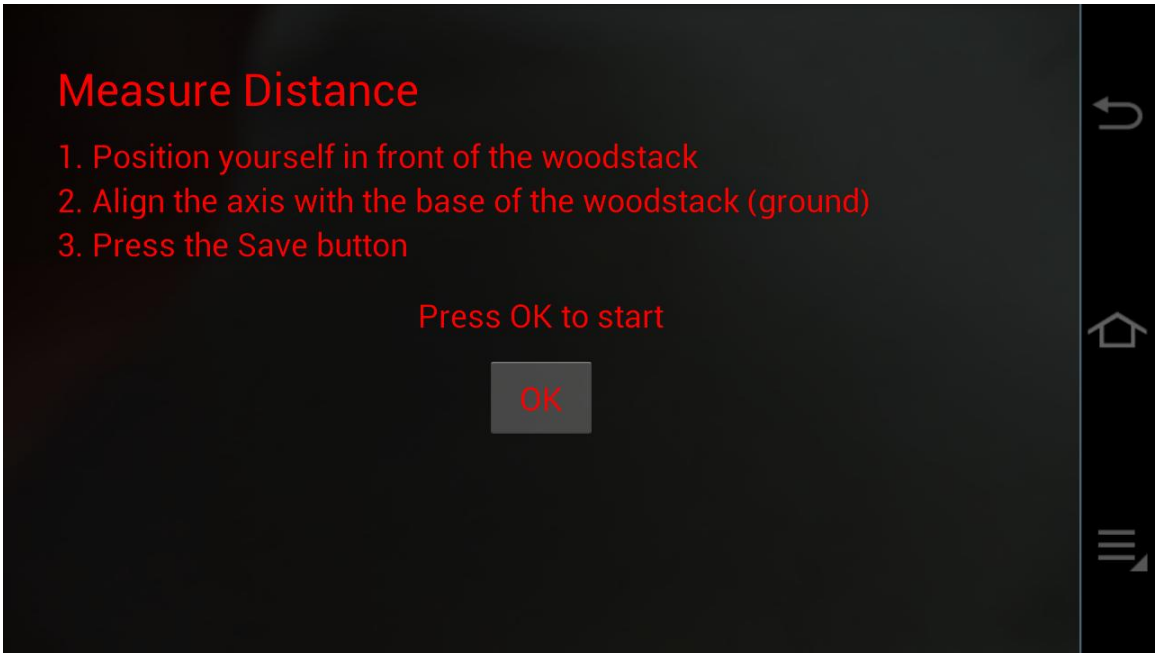

Ilustración 11 – Pantalla: Instrucciones para medir la distancia

El usuario deberá presionar OK para pasar al proceso de medición (4.5.4)

4.5.4. Midiendo la distancia

En esta pantalla el usuario podrá ver lo que está viendo la cámara y sobre ello unos ejes que debe alinear con la base de la pila de madera. Una vez estén correctamente posicionados el usuario deberá pulsar el botón “Save” para que la aplicación guarde la distancia actual.

Ilustración 12 – Midiendo la distancia

Una vez tomada la medida de la distancia se pasará a la captura de la primera fotografía (4.5.5).

Cuando el usuario quiera ver si sigue encontrándose a la misma distancia durante la toma de fotografías puede realizar una nueva medición. En ella se mostrará la misma pantalla con los ejes en el centro de la misma. El botón sin embargo cambiará siendo verde y mostrando la palabra OK si el usuario se encuentra a la distancia correcta o siendo rojo si el usuario no está a la distancia adecuada. Además puede que aparezca una transparencia de la última foto tomada en la zona izquierda de la pantalla (se hablará de ella más adelante).

A continuación podremos ver una foto de cada caso comentado, en primer lugar si la distancia es incorrecta y en segundo lugar con la distancia buena. El usuario tendrá que alinear los ejes de nuevo con la base de la pila de madera para realizar la medición.

Ilustración 13 – Comprobando la distancia (caso distancia incorrecta)

Ilustración 14 – Comprobando la distancia (caso distancia correcta)

4.5.5. Captura de la primera fotografía

En esta pantalla el usuario verá dos líneas que le delimitarán el margen superior e inferior donde debe encontrarse la pila de madera. Cuando esté en la posición correcta presionará el botón de la cámara para hacer la fotografía.

Ilustración 15 – Toma de la primera fotografía

Se ha comentado anteriormente que se puede elegir qué zona de la imagen capturada por la cámara enfocar tocando dicho punto en la pantalla táctil. A continuación podemos observar dos capturas de pantalla. Una con la imagen enfocada y otra desenfocada. El usuario debe recordar que a mayor nitidez mejor resultado del posterior stitching de imágenes.

Ilustración 16 – Ejemplo de imagen enfocada

Ilustración 17 – Ejemplo imagen desenfocada

La aplicación también detectará si la posición de la cámara es adecuada (paralela al suelo) o no y en caso negativo mostrará la imagen con "efecto negativo" para hacérselo saber al usuario.

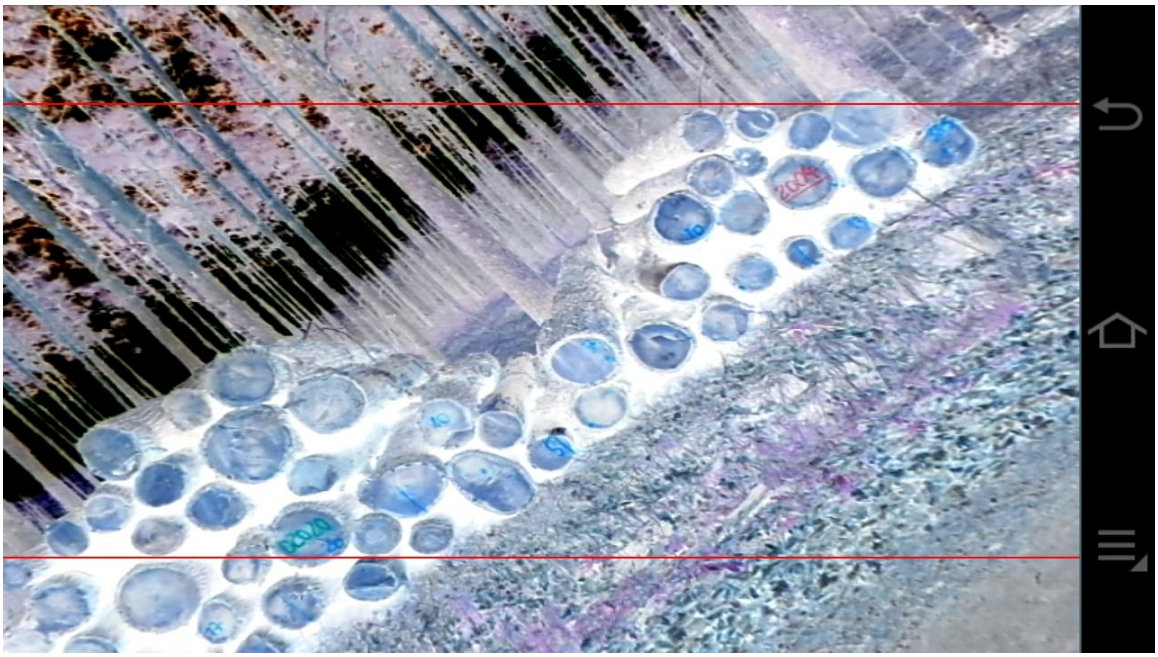

Ilustración 18 – Ejemplo ángulo de cámara incorrecto 1

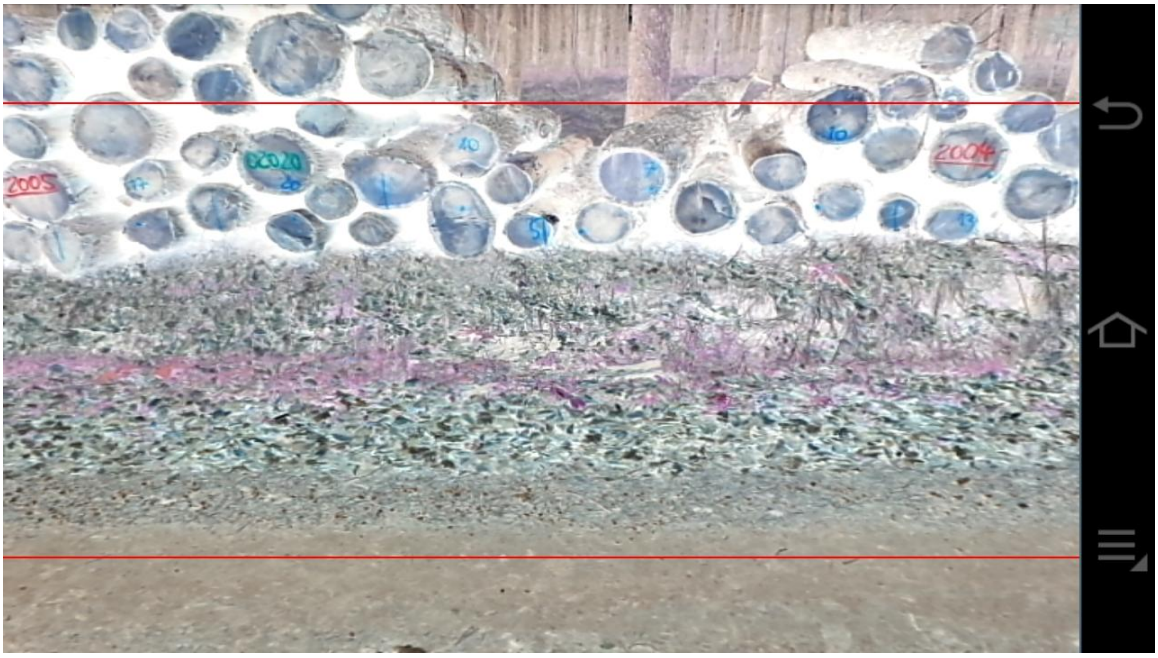

Ilustración 19 – Ejemplo ángulo cámara incorrecto 2

4.5.6. Captura de las siguientes fotografías

Una vez tomada la primera fotografía se seguirán tomando las siguientes. En esta pantalla veremos una imagen semitransparente con una parte de la última captura. El usuario utilizará esta imagen para colocarse de manera adecuada a la hora de fotografiar de nuevo.

Ilustración 20 – Captura de las siguientes fotografías (I)

Tras desplazarse de manera paralela a las pilas de madera colocará la transparencia de manera aproximada sobre lo que ve la cámara en ese momento y presionará el botón de hacer fotografía nuevamente.

Ilustración 21 – Captura de las siguientes fotografías (II)

El usuario tiene la opción de cambiar el color o el nivel de transparencia conforme sea más claro para él (durante las pruebas se observó que algunos usuarios tenían problema al diferenciar imagen real y transparencia si ésta tenía los colores verdaderos).

Ilustración 22 – Cambio del color de la transparencia (azul)

Ilustración 23 – Cambio del color de la transparencia (rojo)

4.5.7. ISO, apertura y velocidad de obturación

Para la modificación de estas tres variables se utiliza el mismo tipo de pantalla. En ella aparece una barra en la parte inferior para modificar el valor de la variable y el botón de OK en la esquina superior derecha para que el usuario indique cuando aplicar el cambio. Además, de fondo se mostrará lo que está viendo la cámara para ver el efecto de la modificación de los valores en la imagen.

En la siguiente página se pueden ver fotografías sobre cada caso en concreto.

ISO:

Ilustración 24 – Ejemplo control de ISO

Apertura:

Ilustración 25 – Ejemplo control apertura

Velocidad de obturación:

Ilustración 26 – Ejemplo control velocidad de obturación

5. Conclusiones

En este apartado se concluirá con la memoria repasando lo que se ha realizado, lo que finalmente no se ha hecho y la explicación de por qué no, posibles mejoras de cara al futuro y una conclusión personal acerca del proyecto.

5.1. ¿Qué se iba a hacer?

El objetivo del Proyecto Final de Carrera era automatizar el proceso de medición de volúmenes de pilas de madera. Para la consecución de este fin se propuso dividir el proyecto en varias partes.

En primer lugar, una aplicación que cumpliese las funciones de asistente fotográfico para guiar al usuario a la hora de hacer las fotografías para las posteriores medidas, obteniendo de este modo mejores resultados. Este asistente permitiría variar los distintos parámetros existentes (ISO, apertura, velocidad de obturación) y, además, contendría muchas más funcionalidades como por ejemplo medir la distancia a la pila de madera, avisar al usuario si la inclinación de la cámara no es correcta o ayudar a posicionar la cámara para conseguir mejores panorámicas.

En segundo lugar, se añadiría a la aplicación comentada la funcionalidad de *stitching*, para componer las panorámicas a partir de las capturas tomadas por el usuario. Esta parte se desarrollaba también de forma paralela para PC debido a que se experimenta una gran mejora en la velocidad de los cálculos con respecto al móvil.

Por último, se incluiría en la aplicación un método para detectar la cantidad de troncos en la fotografía final y calcular el área de sus cortes transversales. A partir de estos datos y la longitud de los troncos (introducida por el usuario manualmente y similar para todos los de la pila) se podría hallar el volumen total de madera existente.

5.2. ¿Qué se ha hecho?

Como se ha podido ir viendo a lo largo de la memoria se han ido desarrollando para la aplicación cada una de las partes anteriores. Se ha hecho especial hincapié en el aspecto más visual para el usuario, el asistente fotográfico y conseguir una panorámica tras el stitching buena para los posteriores cálculos.

El último punto, la detección de la cantidad de troncos y su área, se realiza de forma totalmente transparente al usuario, mostrándole únicamente los datos finales y dejándole al margen de todo el proceso interno que realiza la aplicación para la consecución de los números. Además, se comprobó que a pesar de optimizarlo al máximo, el proceso era muy lento en dispositivos móviles y convenía hacerlo siempre en PC, al menos hasta que estos sean más avanzados. Por todo esto, no se ha hecho tanto hincapié en esta parte del proyecto.

5.3. ¿Qué no se ha hecho y por qué no se ha hecho?

En la versión final para dispositivos móviles no se ha añadido la función de calcular volumen, es accesible sólo para los desarrolladores. Como se ha comentado anteriormente, los resultados se obtenían tras una larga espera, habitualmente sobrepasando los 10 minutos, en los cuales el dispositivo estaba 100% entregado a cálculos. Esto producía una sensación de insatisfacción a los clientes y por ello la funcionalidad se dejó exclusivamente para la versión de ordenador encargada de calcular el volumen a partir de las panorámicas.

Así que, como parte finalmente no hecha del proyecto, se tiene conseguir que el cálculo de volúmenes funcione de forma óptima también en el propio terminal. En cualquier caso, con la rápida evolución de las tecnologías y teniendo en cuenta de que cada día se lanzan al mercado terminales más potentes, en un tiempo se podrá llevar al móvil de manera exitosa. A esto hay que añadir los avances de los desarrolladores de OpenCV para hacer que sus funciones sean más efectivas, haciendo que cada vez se tarde menos tiempo en obtener resultados satisfactorios.

5.4. ¿Qué más se podría haber hecho sin estar previsto al principio?

La aplicación se encuentra en tres idiomas: español, alemán e inglés. Al principio no se planteó siquiera la cuestión de las traducciones por la facilidad con la que se puede realizar a posteriori.

Conforme avanzaba el proyecto se pensó en añadir algún idioma más a la lista de citados anteriormente, pero dado que el mercado al que estaba enfocada la aplicación era principalmente alemán y latinoamericano, se decidió que con esos tres idiomas era suficiente por el momento. No obstante, la traducción futura no se descarta si se amplían las zonas de comercialización.

5.5. Conclusiones a nivel personal

Al comenzar el proyecto, lo primero que percibí es que lo estudiado en la universidad se quedaba muy corto comparado con lo que se me iba a exigir en los futuros meses. También me di cuenta pronto de que cuando te dedicas a la programación cuarenta horas semanales rindes mucho más de lo que esperas.

Empecé con muchas ganas de aprender, con ansias de conocimiento y de descubrir hasta dónde podía llegar y, al finalizar, sigo con esas ganas de seguir instruyéndome y mejorar como ingeniero. Creo que en las dos primeras semanas aprendí mucho más que en todo el curso de programación para Android que estudié en la universidad. Eso sí, éste fue imprescindible para ir con unas bases a la hora de afrontar esta aventura mayor.

Ahora miro atrás y me siento orgulloso de haber superado esta prueba, que puede considerarse la primera en cuanto a trabajo especializado se refiere. He aprendido no sólo sobre programación y sobre Android, sino el funcionamiento de una empresa que se dedica a realizar aplicaciones bajo demanda, la organización dentro de esta y como de imprescindible es la colaboración entre miembros de un mismo equipo por un fin común.

Aparte de todo lo anterior, he trabajado en un ambiente internacional, donde no podía comunicarme en mi lengua materna y esto me ha ayudado a conseguir más fluidez en inglés y a romper barreras de cara al futuro.

5.6. Agradecimientos

Me gustaría dar las gracias en primer lugar a la empresa GIS-Dienst GmbH y, especialmente, a Marion Pause y Michael Gessel, la confianza puesta en mí y la acogida recibida en la empresa para llevar a cabo el proyecto; así como la ayuda que me han ofrecido guiándome y asesorándome a la vez que me daban libertad para desarrollar mis ideas e inquietudes.

En segundo lugar, agradecer también a mi tutor, Miguel Mateo Pla, su apoyo desde España para conseguir que el proyecto llegase a buen término.

6. Bibliografía

[1]. Wikipedia. *Android*. Disponible en: http://es.wikipedia.org/wiki/Android

[2]. Scoello12. Características de Android. Disponible en: https://scoello12.wordpress.com/caracteristicas/

[3]. Xataka. *Android 4.4. Máquina virtual dalvik vs art.* Disponible en:http://www.xatakandroid.com/sistema-operativo/android-4-4-kitkat-experimenta-sustituir-dalvik-por-art-para-mejorar-el-rendimiento

[4]. Androideity. *Arquitectura de Android*. Disponible en: http://androideity.com/2011/07/04/arquitectura-de-android/

[5]. Google. *Arquitectura de Android*. Disponible en: https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android

[6]. Tuprogramacion.com. Estructura *de una aplicación en Android*. Disponible en: http://www.tuprogramacion.com/programacion/estructura-de-una-aplicacion-android/

[7]. AndroidCurso. *Ciclo de vida de una actividad.* Disponible en: http://www.androidcurso.com/index.php/tutoriales-android/37-unidad-6-multimedia-y-ciclo-de-vida/158-ciclo-de-vida-de-una-actividad

[8]. Wikipedia. *OpenCV*. Disponible en: http://es.wikipedia.org/wiki/OpenCV

[9]. GitHub. *JavaCV*. Disponible en: https://github.com/bytedeco/javacv

[10]. Animus Project. *Introducción a OpenCV*. Disponible en: http://animusproject.wix.com/web/apps/blog/opencv-i

[11]. OpenCV. Feature Finder and Feature Matcher. Disponible en: http://docs.opencv.org/modules/stitching/doc/matching.html

[12]. Wikipedia. *Circle Hough Transform.* Disponible en: https://en.wikipedia.org/wiki/Circle_Hough_Transform

[13]. OpenCV. *Hough Circle Transform*. Disponible en: http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html

7. Anexo I – Código fuente de la aplicación móvil

7.1. PicsAllower.java (El núcleo de la app)

```
package com.GDsergio.picsallower;

import static com.googlecode.javacv.cpp.opencv_highgui.cvLoadImage;
import static com.googlecode.javacv.cpp.opencv_highgui.cvSaveImage;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.math.RoundingMode;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.List;

import org.opencv.android.OpenCVLoader;
import org.opencv.android.Utils;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.PorterDuff;
import android.graphics.Rect;
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.Display;
```

```
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.FrameLayout;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;

import com.googlecode.javacv.cpp.opencv_core.IplImage;
import com.googlecode.javacv.cpp.opencv_core.MatVector;
import com.googlecode.javacv.cpp.opencv_stitching.Stitcher;
//import org.opencv.core.Point;

public class PicsAllower extends Activity implements
SensorEventListener{

	/**VARIABLES*/
	//Visibility
	public static final int CANNY_CALL = 0;
	public static final int DISTANCE_INSTRUCTIONS = 101;
	public static final int MEASURE_DISTANCE = 102;
	public static final int TAKE_PICTURE = 103;
	public static final int FINISHED = 104;
	public static final int CHANGE_ALFA = 105;
	public static final int PARAMETER_ALFA = 200;
	public static final int PARAMETER_ISO = 201;
	public static final int PARAMETER_EXPOSURE = 202;
	public static final int PARAMETER_FOCAL = 203;
	public static final int PARAMETER_TIME = 204;
	public static final int T_NORMAL = 0;
	public static final int T_RED = 1;
	public static final int T_BLUE = 2;

	//General variables
	public static final int PREVIEW_WIDTH = 640;
	public static final int PREVIEW_HEIGHT = 480;
	public static final int PICTURE_WIDTH = 1984;  //3Mpx (1984x1488)
	public static final int PICTURE_HEIGHT = 1488;
	private static final int maxHeight = 200; //is in cm -> 2 meters
	private static final int startingPoint = 100; //1 meter
	public static final float UP_DOWN_MARGINS = 0.15f;
	public static final float NOT_OVERLAPING_AREA = 0.6f;

	private int currentStage, currentParameter,    isoValue,
	            timeValue = 0, transparencyMode = T_NORMAL;
	private float alfaValue, angleXFB = 0, savedAngleXFB,
```

```
        angleXLR = 0,heightProgress = 1,
              /*EstDistance = 0,*/ GndDistance = 0, rotAngle = 0,
        stabilityAngle=0, savedGroundDistance = 0, focalValue,
        firstPictureAngle;
  private boolean firstTime, stitching, loading, takingPicture,
        firstChain, measureDistance, captureButtonNeeded;
  private String alfaValueString;
  private String appName;

  private ArrayList <String> timeList =
            new ArrayList<String> (Arrays.asList(
            "1/100","1/125","1/160","1/200","1/250","1/320","1/400
            ","1/500","1/640","1/800","1/1000","1/1250","1/1600","
            1/2000"));

  private ArrayList <Integer> isoList =
            new ArrayList<Integer> (Arrays.asList(
                        100, 200, 400, 800, 1600, 3200));

  private FocalValues fv = new FocalValues();

  private ImageView topLine, bottomLine, horizontalRedLine,
        verticalRedLine, blackScreen;
  private TextView picsToStitch, tvEstimatedDistance,
        tvGroundDistance, tvHeight,
        tvInstructionsTitle, tvInstruction1, tvInstruction2,
        tvInstruction3;
  private LinearLayout valuesColumn, distanceInstructions;
  private Toast toast;
  private SeekBar sb;
  private Bitmap myBitmap, blackScreenBitmap;
  private Canvas blackScreenCanvas;
  private Paint blackScreenPaint;
  private Rect blackScreenRect;

//Angle stuff variables
  AngleCalculator ac;
  private SensorManager mSensorManager = null;
  private boolean validAux;
  DecimalFormat d = new DecimalFormat("#.##");


  //Camera variables
  public static final int MEDIA_TYPE_IMAGE = 1;
  public static final int MIN_ZOOM = 1; //1.2, it works with intgrs
  private static final String TAG = "CameraPreview";

  private Camera myCamera;
  private CameraPreview myPreview;
  private Camera.Parameters params;
  private List<Integer> zoom_list;
  private FrameLayout preview;
  private TextView tvZoomValue;
  private ImageView fullPic;
  private Button captureButton, distanceInstructionsOKButton,
```

```
            saveDistanceButton, startButton, finishButton;
      private int zoomValue, realZoomValue, maxZoom, picturesTaken = 0;

    static String dirPath, imageNameAux;
    private String [] imageNames = new String[20];
      //For the image names (max 20)

    Thread timerOF, timerMP;
    private static Handler uiCallback;
    private AlertDialog dialog;
    private Intent myIntent;

    //Stitching variables
    private MatVector images;
    private Stitcher stitcher;
    private IplImage result;

    private static File mediaStorageDir;

    //Canny & HoughCircles
    private Mat mRgba, mGray, mIntermediateMat;
    private double iKernelSize1, iCannyLowerThreshold1,
CannyUpperThreshold1;

    /**START OPENCV*/
    static {
          if (!OpenCVLoader.initDebug()) {
              // Handle initialization error
          }
      }

/**ON CREATE*/
@Override
protected void onCreate(Bundle savedInstanceState)
{super.onCreate(savedInstanceState);

      /**FLAGS FOR THE SYSTEM*/
this.requestWindowFeature(Window.FEATURE_NO_TITLE);
this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
this.getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_O
N);

setContentView(R.layout.pics_allower);
Bundle extras = getIntent().getExtras();
if (extras != null) {
            measureDistance = extras.getBoolean("measureDistance");
            }

picsToStitch = (TextView)findViewById(R.id.textView5);

/**CHECKING CAMERA*/
if(checkCameraHardware(this)){
// Create an instance of Camera
      myCamera = getCameraInstance();
```

```
        if(myCamera!=null){

              //Initialize camera
              params = myCamera.getParameters();
              params.set("mode", "m");
            //Down to 3Mpx (1984x1488)
            params.setPictureSize(PICTURE_WIDTH, PICTURE_HEIGHT); //4:3

            //This can fail in other devices if they don't support it
            params.setPreviewSize (PREVIEW_WIDTH, PREVIEW_HEIGHT); //4:3

            zoom_list = params.getZoomRatios();
            zoomValue = params.getZoom();
            realZoomValue = zoom_list.get(zoomValue);
            maxZoom = params.getMaxZoom();
            Log.i("MaxZoom", String.valueOf(maxZoom));

            if (params.isAutoExposureLockSupported()){
              params.setAutoExposureLock(true);}
            if (params.isAutoWhiteBalanceLockSupported()){
              params.setAutoWhiteBalanceLock(true);}

            isoValue = isoList.get(0);
            params.set("iso", isoValue);
            myCamera.setParameters(params);
            params = myCamera.getParameters();
            focalValue = params.getFocalLength();
            Log.i("FocalValueIni", String.valueOf(focalValue));

    // Create our Preview view and set it as the content of our activity.
        myPreview = new CameraPreview(this, myCamera);
        preview = (FrameLayout) findViewById(R.id.camera_preview);
        preview.addView(myPreview);

        //Decimal Rounding
        d.setRoundingMode(RoundingMode.HALF_UP);
        d.setMaximumFractionDigits(3);
        d.setMinimumFractionDigits(3);

        //LinearLayouts
        valuesColumn = (LinearLayout)findViewById(R.id.values_column);
        distanceInstructions =
              (LinearLayout)findViewById(R.id.distance_instructions);

        //Add TextViews
        tvZoomValue = (TextView)findViewById(R.id.textView6);
        tvZoomValue.setText(" ");

        tvEstimatedDistance = (TextView)findViewById(R.id.textView7);
        tvGroundDistance = (TextView)findViewById(R.id.textView8);
        tvHeight = (TextView)findViewById(R.id.show_height);

        tvInstructionsTitle = (TextView)findViewById(R.id.textView9);
        tvInstruction1 = (TextView)findViewById(R.id.textView10);
        tvInstruction2 = (TextView)findViewById(R.id.textView11);
```

```
        tvInstruction3 = (TextView)findViewById(R.id.textView12);

        //Add ImageViews, RedLine Axis and RedLine Margins
        fullPic = (ImageView)findViewById(R.id.imageView1);
        topLine = (ImageView)findViewById(R.id.top_line);
        bottomLine = (ImageView)findViewById(R.id.bottom_line);
        horizontalRedLine = (ImageView)findViewById(R.id.axis_line_h);
        verticalRedLine = (ImageView)findViewById(R.id.axis_line_v);
        blackScreen = (ImageView)findViewById(R.id.black_screen);

        Display display = getWindowManager().getDefaultDisplay();
        Point size = new Point();
        display.getSize(size);
        int width = size.x;
        int height = size.y;
        topLine.setY(UP_DOWN_MARGINS*height);
        bottomLine.setY((1-UP_DOWN_MARGINS)*height);

        //Sidebar to change the parameters
        sb = (SeekBar) findViewById(R.id.select_height);
        sb.setMax(maxHeight); //In centimeters
        sb.incrementProgressBy(startingPoint);
        heightProgress = (float) startingPoint/100;
        tvHeight.setText("h: " + String.format("%.2f", heightProgress)
                        + "m"); //Beginning

        sb.setOnSeekBarChangeListener(
              new SeekBar.OnSeekBarChangeListener(){
                    @Override
                    public void onProgressChanged(
                          SeekBar seekBar, int progress,
                               boolean fromUser) {
// TODO Auto-generated method stub
              if (currentStage == MEASURE_DISTANCE){
                    heightProgress = (float)progress/100;
                    tvHeight.setText("h: " + String.format("%.2f",
                          heightProgress)+ "m");

              }else if (currentStage == CHANGE_ALFA){
                    switch(currentParameter){
                    case PARAMETER_ALFA:
                    alfaValue = (float)progress/100;
                    alfaValueString =
                    Integer.toString(Math.round(alfaValue*0.5f*255), 16);
                    if (alfaValueString.length()==1){
                          alfaValueString = "0" + alfaValueString;
                    }
                    fullPic.setAlpha(alfaValue);
                    if (transparencyMode == T_NORMAL){
                          fullPic.clearColorFilter();
                    }else if (transparencyMode == T_RED){
                          fullPic.setColorFilter(Color.parseColor(
                          "#"+alfaValueString+"AA0000"),
                           PorterDuff.Mode.OVERLAY);
                    }else if (transparencyMode == T_BLUE){
```

```
                        fullPic.setColorFilter(Color.parseColor(
                        "#"+alfaValueString+"0000AA"),
                        PorterDuff.Mode.OVERLAY);
                    }
                    tvHeight.setText(
                        String.valueOf((int)(alfaValue*100)));
                    break;


                    case PARAMETER_ISO:
                    params.set("mode", "m");
                    isoValue = isoList.get(progress);
                    params.set("iso", isoValue);
                    myCamera.setParameters(params);
                    params = myCamera.getParameters();
                    tvHeight.setText(String.valueOf(isoValue));
                    Log.i("iso", String.valueOf(isoValue));
                    break;

                    case PARAMETER_FOCAL:
                    params.set("mode", "m");
                    focalValue = fv.getFocalValue(zoomValue, progress);
                    params.set("aperture", Math.round(focalValue*10));
                    myCamera.setParameters(params);
                    tvHeight.setText(String.valueOf(focalValue));
                    Log.i("FocalValueBar", String.valueOf(focalValue));
                    break;

                    case PARAMETER_TIME:
                    params.set("mode", "m");
                    timeValue = progress;
                    params.set("shutter-speed", timeValue+32);
                    myCamera.setParameters(params);
                    params = myCamera.getParameters();
                    tvHeight.setText(timeList.get(progress));
                    Log.i("time", timeList.get(progress));
                    break;
                    }
                }
        }

        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {
        // TODO Auto-generated method stub
        }

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {
        // TODO Auto-generated method stub
        }
});

//Angle
ac = new AngleCalculator();
mSensorManager = (SensorManager) this.getSystemService(SENSOR_SERVICE);
```

```
alfaValue = 0.58f;
alfaValueString = Integer.toString(Math.round(alfaValue*0.5f*255), 16);

takingPicture = false;
stitching = false;
firstChain = true;

//Buttons actions
distanceInstructionsOKButton = (Button)
findViewById(R.id.distance_instructions_ok_button);
distanceInstructionsOKButton.setOnClickListener(
      new View.OnClickListener() {
            @Override
            public void onClick(View v) {
            visibilityFunction(MEASURE_DISTANCE);
            params.set("zoom", 1);
            params.set("curr_zoom_level", 1);
            myCamera.setParameters(params);
            params = myCamera.getParameters();
            zoomValue = params.getInt("zoom");
            realZoomValue = zoom_list.get(zoomValue);
            }
      });

saveDistanceButton = (Button) findViewById(R.id.button_save_distance);
saveDistanceButton.setOnClickListener(
      new View.OnClickListener() {
            @Override
            public void onClick(View v) {
            if(currentStage == MEASURE_DISTANCE){
                  if(picturesTaken == 0){
                  //savedGroundDistance = GndDistance;
                  savedAngleXFB = angleXFB;
                  }
            }
            visibilityFunction(TAKE_PICTURE);
      }
});

startButton = (Button) findViewById(R.id.button_start);
startButton.setOnClickListener(
      new View.OnClickListener() {
            @Override
            public void onClick(View v) {
            if (currentStage == FINISHED){
                  if(stitching == false){
                        fullPic.clearColorFilter();
                        picturesTaken = 0;
                        myIntent = new Intent(PicsAllower.this,
                              MainActivity.class);
                        startActivity(myIntent);
                  }else{
                        displayToastLong(getString(R.string.wait));
                  }
            }else if (currentStage == CHANGE_ALFA &&
```

```
                        currentParameter == PARAMETER_ALFA){
                        if (transparencyMode == T_NORMAL){
                              transparencyMode = T_RED;
                              fullPic.setColorFilter(Color.parseColor(
                              "#"+alfaValueString+"AA0000"),
                              PorterDuff.Mode.OVERLAY);
                        }else if(transparencyMode == T_RED){
                              transparencyMode = T_BLUE;
                              fullPic.setColorFilter(Color.parseColor(
                              "#"+alfaValueString+"0000AA"),
                              PorterDuff.Mode.OVERLAY);
                        }else if(transparencyMode == T_BLUE){
                              transparencyMode =T_NORMAL;
                              fullPic.clearColorFilter();
                  }
               }
         }
});
startButton.setBackgroundColor(Color.GRAY);
finishButton = (Button) findViewById(R.id.button_finish);
finishButton.setOnClickListener(
      new View.OnClickListener() {
      @Override
            public void onClick(View v) {
                  finishButton.setVisibility(View.GONE);
                  finishButton.setEnabled(false);
                  picsToStitch.setText(String.valueOf(picturesTaken)
                  + " " + getString(R.string.pictures_taken));
                  fullPic.setAlpha((float)0);
                  dialog.show();
                  visibilityFunction(FINISHED);
            }
      });
finishButton.setVisibility(View.GONE);

//It appears only if we don't have a physical button to take pictures
in our device
captureButton = (Button) findViewById(R.id.button_capture);
captureButton.setOnClickListener(
new View.OnClickListener() {
      @Override
      public void onClick(View v) {
            if (currentStage == TAKE_PICTURE){
                  if (validAux==true && loading==false){
                        picsToStitch.setText(" ");
                        loading = true;
                        takingPicture = true;
                        captureButton.setEnabled(false);
                        if (picturesTaken == 0){
                              firstPictureAngle = angleXFB;
                        }
                        myCamera.takePicture(null, null, jpegCallBack);
                  }
            }
      }
```

```
});

startButton.setBackgroundColor(Color.GRAY);

if(android.view.KeyCharacterMap.deviceHasKey(KeyEvent.KEYCODE_CAMERA)){
      captureButtonNeeded = false;
      captureButton.setVisibility(View.GONE);
}else{
      captureButtonNeeded = true;
}

//Painting the Black Screen (Background)
blackScreenBitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888);

blackScreenCanvas = new Canvas(blackScreenBitmap);
blackScreenPaint = new Paint();
blackScreenPaint.setColor(Color.BLACK);
blackScreenPaint.setStyle(Paint.Style.FILL);
blackScreenRect = new Rect(0, 0, width, height);
blackScreenCanvas.drawRect(blackScreenRect, blackScreenPaint);
blackScreen.setImageBitmap(blackScreenBitmap);


/* Disable and make invisible all Buttons, TextViews, etc., but the
ones related to the Distance_Instructions. It depends of our first
choice (measure or not measure the distance)*/

if (measureDistance){
      visibilityFunction(DISTANCE_INSTRUCTIONS)
}else{
      visibilityFunction(TAKE_PICTURE);
      try{
            Thread.sleep(700);
      }catch(Exception e){
            Log.i("Error Inicial", e.getMessage());
      }
      params.set("zoom", 1);
      params.set("curr_zoom_level", 1);
      myCamera.setParameters(params);
      params = myCamera.getParameters();
      zoomValue = params.getInt("zoom");
      realZoomValue = zoom_list.get(zoomValue);
}
initListeners();

//Canny & HoughCircles variable init
mRgba = new Mat();
mGray = new Mat();
mIntermediateMat = new Mat();

iCannyLowerThreshold1 = 45;  //35 - 100
iCannyUpperThreshold1 = 105;  //75 - 250
iKernelSize1 = 5;
```

```
} //Closes if(myCamera!=null)
} //Closes if(checkCameraHardware(this))
} //Closes onCreate()

@Override
public void onResume() {
super.onResume();
initListeners();
/** Camera resume*/
if(myCamera==null){
      //Start the camera again if it wasn't
      myCamera = getCameraInstance(); //Camera.open inside a try-catch
      myCamera.setParameters(params);
      params = myCamera.getParameters();

      if(myCamera!=null){
            //Once the camera is started, set our parameters again
            myPreview = new CameraPreview(this, myCamera);
            preview.addView(myPreview);

            params.set("mode", "m");
            //Down to 3Mpx (1984x1488)
            params.setPictureSize(PICTURE_WIDTH, PICTURE_HEIGHT); //4:3
            //This can fail in other devices if they don't support it
            params.setPreviewSize (PREVIEW_WIDTH, PREVIEW_HEIGHT); //4:3
            myCamera.setParameters(params);
            params = myCamera.getParameters();

            firstChain = false;
            }
      }

      firstTime = true; //Check changeColor to understand
      loading = false;
      stitching = false;
      savedAngleXFB = 0;
      picturesTaken = 0;
      appName = getString(R.string.app_name);
      createDialog();

    }

    @Override
    protected void onPause() {
      super.onPause();

      //Remove possible toasts remaining
      if(toast != null){
            toast.cancel();
      }
      visibilityFunction (FINISHED);
      fullPic.clearColorFilter();
      fullPic.setAlpha((float)0);

      //Unregister sensor listeners to save battery.
```

```
  mSensorManager.unregisterListener(this);
  releaseCamera();

  finish();

}

private void releaseCamera(){
    if (myCamera != null){
        myCamera.release();
        myCamera = null;
    }
    preview.removeView(myPreview);
    myPreview=null;
}
```

```
private boolean checkCameraHardware(Context context) {
        if (context.getPackageManager().hasSystemFeature(
            PackageManager.FEATURE_CAMERA)){
            // this device has a camera
            return true;
        } else {
            // no camera on this device
            return false;
        }
    }

    /** A safe way to get an instance of the Camera object. */
    public static Camera getCameraInstance(){
        Camera c = null;
        try {
            c = Camera.open(); // attempt to get a Camera instance
        }
        catch (Exception e){
            // Camera is not available (in use or does not exist)
        }
        return c; // returns null if camera is unavailable
    }

    /**Creates a new Picture Callback for when the picture is done*/
    @SuppressLint("HandlerLeak")
      private PictureCallback jpegCallBack = new PictureCallback() {

        @Override
        public void onPictureTaken(byte[] data, Camera camera) {
            camera.startPreview(); //Start Preview Again
            File pictureFile = getOutputMediaFile(
                  MEDIA_TYPE_IMAGE, picturesTaken, appName);
                  Log.i("jpegCallback", "INICIO");

            Display display = getWindowManager().getDefaultDisplay();
            Point size = new Point();
            display.getSize(size);
            int width = size.x;
            int height = size.y;

            if (pictureFile == null){
                  displayToast(
                  "Error creating media file,
                  check storage permissions");
                return;
            }


            try {
                FileOutputStream fos =
                        new FileOutputStream(pictureFile);
                fos.write(data);
                fos.close();
                displayToast("Stored: " + imageNameAux);
```

```
            if (picturesTaken<20){
              imageNames [picturesTaken]= new String (imageNameAux);
                    picturesTaken += 1;
              }else if (picturesTaken >= 20){

              //After 20pics it won't save more, it's the max
              value of our vector, we can change it.
              displayToast("20 PICS LIMIT REACHED, PLEASE FINISH");
              }

              //decode a Bitmap without memory problem
              (due to high expenses of big pics)

              //Move the image to cover only a part of the screen
              fullPic.setX(-width*NOT_OVERLAPING_AREA);

              // NORMAL_PREVIEW:
              myBitmap = Bitmap.createScaledBitmap
              (ImageResizer.decodeSampledBitmapFromResource
              (imageNameAux, 1280, 960), width, height, true);
              fullPic.setImageBitmap(myBitmap);

              if (transparencyMode == T_NORMAL){
                    fullPic.clearColorFilter();
              }else if (transparencyMode == T_RED){

              fullPic.setColorFilter(Color.parseColor(
              "#"+alfaValueString+"AA0000"),
              PorterDuff.Mode.OVERLAY);
              }else if (transparencyMode == T_BLUE){
              fullPic.setColorFilter(Color.parseColor(
              "#"+alfaValueString+"0000AA"),
              PorterDuff.Mode.OVERLAY);
              }
              fullPic.setAlpha(alfaValue);
              stitching = false;

        } catch (FileNotFoundException e) {
              displayToast("File not found");
        } catch (IOException e) {
            displayToast("Error accessing file");
        }
        loading = false;
        if(captureButtonNeeded){
              captureButton.setEnabled(true);
        }
        takingPicture = false;
        //visibilityFunction(DISTANCE_INSTRUCTIONS);
        visibilityFunction(TAKE_PICTURE);
        //Again if user doesn't select Measure Distance
        Log.i("jpegCallback", "FINAL");

    }
};
```

```
    /** Creates a File for saving an image*/
    @SuppressLint("SimpleDateFormat")
      private static File getOutputMediaFile(int type, int picsTaken,
String appName){
// To be safe, you should check that the SDCard is mounted
// using Environment.getExternalStorageState() before doing this.
      if (picsTaken == 0){
            mediaStorageDir = new
            File(Environment.getExternalStoragePublicDirectory(
                              Environment.DIRECTORY_PICTURES), appName);
// This location works best if you want the created images to be shared
// between applications and persist after your app has been
uninstalled.
              // Create the storage directory if it does not exist
              if (! mediaStorageDir.exists()){
                  if (! mediaStorageDir.mkdirs()){
                      Log.d(appName, "failed to create directory");
                      return null;
                  }
              }
      }


// Create a media file name
String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
Date());

//Create a new folder for each project-------
if (picsTaken == 0){
      mediaStorageDir = new File(mediaStorageDir, "P_" + timeStamp);
             // Create the storage directory if it does not exist
             if (! mediaStorageDir.exists()){
                if (! mediaStorageDir.mkdirs()){
                    Log.d("Project_Dir", "failed to create directory");
                    return null;
                }
             }
        }
        //-----------------------------------------

        File mediaFile;
        dirPath = mediaStorageDir.getPath();
        if (type == MEDIA_TYPE_IMAGE){
            imageNameAux = dirPath + File.separator +
            "IMG_"+ timeStamp + ".jpg";
            mediaFile = new File(imageNameAux);

            /*mediaFile = new File(mediaStorageDir.getPath() +
File.separator +
            "IMG_"+ timeStamp + ".jpg");*/
        } else {
            return null;
        }

        return mediaFile;
```

```
    }

    /** A basic Camera preview class */
    public class CameraPreview extends SurfaceView implements
SurfaceHolder.Callback {
        private SurfaceHolder mHolder;
        private Camera mCamera;

        @SuppressWarnings("deprecation")
            public CameraPreview(Context context, Camera camera) {
            super(context);
            mCamera = camera;

            // Install a SurfaceHolder.Callback so we get notified when
the
            // underlying surface is created and destroyed.
            mHolder = getHolder();
            mHolder.addCallback(this);
            // deprecated setting, but required on Android versions
prior to 3.0
            mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        }

        public void surfaceCreated(SurfaceHolder holder) {
            // The Surface has been created, now tell the camera where
to draw the preview.
            try {
                mCamera.setPreviewDisplay(holder);
                mCamera.startPreview();

            } catch (IOException e) {
                Log.d(TAG, "Error setting camera preview: " +
e.getMessage());
            }
        }

        public void surfaceDestroyed(SurfaceHolder holder) {
            // empty. Take care of releasing the Camera preview in your
activity.
        }

        public void surfaceChanged(SurfaceHolder holder, int format,
int w, int h) {
            // If your preview can change or rotate, take care of those
events here.
            // Make sure to stop the preview before resizing or
reformatting it.

            if (mHolder.getSurface() == null){
              // preview surface does not exist
              return;
            }

            // stop preview before making changes
            try {
```

```
                mCamera.stopPreview();
            } catch (Exception e){
              // ignore: tried to stop a non-existent preview
            }

            // set preview size and make any resize, rotate or
            // reformatting changes here

            // start preview with new settings
            try {
                mCamera.setPreviewDisplay(mHolder);
                mCamera.startPreview();

            } catch (Exception e){
                Log.d(TAG, "Error starting camera preview: " +
e.getMessage());
            }
        } //Closes surface changed
    } //Closes CameraPreview

      /**Touch the Screen to Focus*/
      @Override
      public boolean onTouchEvent(MotionEvent event) {
            if(currentStage==TAKE_PICTURE || currentStage ==
MEASURE_DISTANCE){ //Focus only when needed
                  if(takingPicture == false){
                        AutoFocusAdjust autoF = new AutoFocusAdjust();
                        autoF.adjust(event, myCamera, preview);
                  }
          }
      return false; //default, it doesn't matter
      }

      /**Shot with the Camera Button and make zoom*/
      @Override
      public boolean onKeyDown(int keyCode, KeyEvent event) {
            //AutoFocusAdjust autoF = new AutoFocusAdjust();

            if(event.getAction() == KeyEvent.ACTION_DOWN){
                  switch(keyCode){
                  case KeyEvent.KEYCODE_BACK: //Don't allow to go back
                                              without finishing first
                  if(currentStage != FINISHED){
                        displayToast(getString(R.string.first_finish));
                        return true;
                  }
                  break;
                  case KeyEvent.KEYCODE_CAMERA: //Camera Button to pic
                  if (currentStage == TAKE_PICTURE){
                        if (validAux==true && loading==false){
                              picsToStitch.setText(" ");
                              loading = true;
                              takingPicture = true;
                              if (picturesTaken == 0){
                                    firstPictureAngle = angleXFB;
```

```
                        }
                        myCamera.takePicture(null, null, jpegCallBack);
                        }
                  }
                  return true;

                  case KeyEvent.KEYCODE_ZOOM_IN:
                  if((currentStage == MEASURE_DISTANCE ||
                        currentStage == TAKE_PICTURE) &&
                        picturesTaken == 0){
      params.set("mode", "smart-auto");
      params.set("zoom-speed", 1);
      params.set("zoom-action", "optical-tele-start");
      myCamera.setParameters(params);
      params = myCamera.getParameters();
      zoomValue = params.getInt("curr_zoom_level");
      realZoomValue = zoom_list.get(zoomValue);
      tvZoomValue.setText("Zoom: " +
      String.valueOf(((float)realZoomValue)/100));
      params.set("mode", "m");
      myCamera.setParameters(params);
}else if ((currentStage == MEASURE_DISTANCE ||
      currentStage == TAKE_PICTURE) && picturesTaken > 0){
      displayToast(getString(R.string.cannot_zoom));
}

if (params.getInt("curr_zoom_level")==0){
      displayToastLong(getString(R.string.min_zoom_alert));
}
return true;

case KeyEvent.KEYCODE_ZOOM_OUT:
if((currentStage == MEASURE_DISTANCE || currentStage == TAKE_PICTURE)
&& picturesTaken == 0){
      params.set("mode", "smart-auto");
      params.set("zoom-speed", 1);
      params.set("zoom-action", "optical-wide-start");
      myCamera.setParameters(params);
      params = myCamera.getParameters();
      zoomValue = params.getInt("curr_zoom_level");
      realZoomValue = zoom_list.get(zoomValue);
      tvZoomValue.setText("Zoom: " +
            String.valueOf(((float)realZoomValue)/100));
      params.set("mode", "m");
      myCamera.setParameters(params);
      }else if ((currentStage == MEASURE_DISTANCE || currentStage ==
            TAKE_PICTURE) && picturesTaken > 0){
            displayToast(getString(R.string.cannot_zoom));
      }

      if (params.getInt("curr_zoom_level")==0){
            displayToastLong(getString(R.string.min_zoom_alert));
      }
      return true;
```

```
case KeyEvent.KEYCODE_VOLUME_UP: //Some devices use the volume buttons
                                    for the zoom
if((currentStage == MEASURE_DISTANCE ||
      currentStage == TAKE_PICTURE) && picturesTaken == 0){
      params.set("mode", "smart-auto");
      params.set("zoom-speed", 1);
      params.set("zoom-action", "optical-tele-start");
      myCamera.setParameters(params);
      params = myCamera.getParameters();
      zoomValue = params.getZoom();
      realZoomValue = zoom_list.get(zoomValue);
      tvZoomValue.setText("Zoom: " +
            String.valueOf(((float)realZoomValue)/100));
      params.set("mode", "m");
      myCamera.setParameters(params);
      }else if ((currentStage == MEASURE_DISTANCE || currentStage ==
TAKE_PICTURE) && picturesTaken > 0){

displayToast(getString(R.string.cannot_zoom));
                              }

if (params.getZoom()==0){

displayToastLong(getString(R.string.min_zoom_alert));
}

return true;

case KeyEvent.KEYCODE_VOLUME_DOWN: //Some devices use the volume
buttons for the zoom
if((currentStage == MEASURE_DISTANCE || currentStage == TAKE_PICTURE)
&& picturesTaken == 0){
      params.set("mode", "smart-auto");
      params.set("zoom-speed", 1);
      params.set("zoom-action", "optical-wide-start");
      myCamera.setParameters(params);
      params = myCamera.getParameters();
      //zoomValue = params.getInt("curr_zoom_level");
      zoomValue = params.getZoom();
      realZoomValue = zoom_list.get(zoomValue);
      tvZoomValue.setText("Zoom: " +
      String.valueOf(((float)realZoomValue)/100));
      params.set("mode", "m");
      myCamera.setParameters(params);
}else if ((currentStage == MEASURE_DISTANCE || currentStage ==
      TAKE_PICTURE) && picturesTaken > 0){

      displayToast(getString(R.string.cannot_zoom));
}
if (params.getZoom()==0){

      displayToastLong(getString(R.string.min_zoom_alert));
}
return true;
}
```

```
}
return super.onKeyDown(keyCode, event);
}

@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {
//AutoFocusAdjust autoF = new AutoFocusAdjust();
      if(event.getAction() == KeyEvent.ACTION_UP){
      switch(keyCode){ //This is necessary because when you release the
KeyDown, you go through KeyUp
            case KeyEvent.KEYCODE_ZOOM_IN:
            if((currentStage == MEASURE_DISTANCE || currentStage ==
            TAKE_PICTURE) && picturesTaken == 0){
            params.set("mode", "smart-auto");
            params.set("zoom-action", "zoom-stop");
            myCamera.setParameters(params);
            params.set("mode", "m");
            myCamera.setParameters(params);
            params = myCamera.getParameters();
            if (params.getInt("curr_zoom_level")==0){
                  displayToastLong(getString(R.string.min_zoom_alert));
            }

            zoomValue = params.getInt("curr_zoom_level");
            realZoomValue = zoom_list.get(zoomValue);
            if(" ".compareTo(String.valueOf(tvZoomValue.getText()))!=0){
            tvZoomValue.setText("Zoom: " +
                  String.valueOf(((float)realZoomValue)/100));
                                    }
            tvZoomValue.setText(" ");
            }
            return true;

            case KeyEvent.KEYCODE_ZOOM_OUT:
            if((currentStage == MEASURE_DISTANCE || currentStage ==
                  TAKE_PICTURE) && picturesTaken == 0){
            params.set("mode", "smart-auto");
            params.set("zoom-action", "zoom-stop");
            myCamera.setParameters(params);
            params.set("mode", "m");
            myCamera.setParameters(params);
            params = myCamera.getParameters();

            if (params.getInt("curr_zoom_level")==0){
                  displayToastLong(getString(R.string.min_zoom_alert));
            }

            zoomValue = params.getInt("curr_zoom_level");
            realZoomValue = zoom_list.get(zoomValue);
            if(" ".compareTo(String.valueOf(tvZoomValue.getText()))!=0){
                  tvZoomValue.setText("Zoom: " +
            String.valueOf(((float)realZoomValue)/100));
            }
                  tvZoomValue.setText(" ");
            }
```

```
return true;

case KeyEvent.KEYCODE_VOLUME_UP:
      if((currentStage == MEASURE_DISTANCE ||
      currentStage == TAKE_PICTURE) && picturesTaken == 0){
      params.set("mode", "smart-auto");
      params.set("zoom-action", "zoom-stop");
      myCamera.setParameters(params);
      params.set("mode", "m");
      myCamera.setParameters(params);
      params = myCamera.getParameters();

      if (params.getZoom()==0){
      displayToastLong(getString(R.string.min_zoom_alert));
      }

      zoomValue = params.getZoom();
      realZoomValue = zoom_list.get(zoomValue);
      if(" ".compareTo(String.valueOf(
            tvZoomValue.getText()))!=0){
            tvZoomValue.setText("Zoom: " +
            String.valueOf(((float)realZoomValue)/100));
            }
            tvZoomValue.setText(" ");
      }
      return true;

      case KeyEvent.KEYCODE_VOLUME_DOWN:
            if((currentStage == MEASURE_DISTANCE ||
            currentStage == TAKE_PICTURE)
            && picturesTaken == 0){
            params.set("mode", "smart-auto");
            params.set("zoom-action", "zoom-stop");
            myCamera.setParameters(params);
            params.set("mode", "m");
            myCamera.setParameters(params);
            params = myCamera.getParameters();

            if (params.getZoom()==0){
      displayToastLong(getString(R.string.min_zoom_alert));
            }

            zoomValue = params.getZoom();
            realZoomValue = zoom_list.get(zoomValue);
            if(" ".compareTo(
            String.valueOf(
                  tvZoomValue.getText()))!=0){
                  tvZoomValue.setText("Zoom: " +
                  String.valueOf((
                  (float)realZoomValue)/100));
                        }
                  tvZoomValue.setText(" ");
            }
                return true;
            }
```

```
                }

            return super.onKeyUp(keyCode, event);
      }

      /**Sensors and Angle Calulator*/
public void initListeners(){
mSensorManager.registerListener(this,
mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
SensorManager.SENSOR_DELAY_FASTEST);

mSensorManager.registerListener(this,
mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE),
SensorManager.SENSOR_DELAY_FASTEST);

mSensorManager.registerListener(this,

mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
    SensorManager.SENSOR_DELAY_FASTEST);
}

@Override
public void onSensorChanged(SensorEvent event) {
      switch(event.sensor.getType()) {
      case Sensor.TYPE_ACCELEROMETER:
      angleXFB = (float) (Math.atan2(event.values[0],
      event.values[2])/(Math.PI/180)); //In degrees
      angleXLR = (float) (Math.atan2(event.values[0],
      event.values[1])/(Math.PI/180));

      //IMAGE ROTATION: to move the help axis
      if (currentStage == MEASURE_DISTANCE){
            rotAngle=angleXLR-90;
            if((Math.abs(stabilityAngle-rotAngle))>(0.5)){
            //Al menos 1/2 grado de dif para que rote
                  horizontalRedLine.setRotation(rotAngle);
                  verticalRedLine.setRotation(rotAngle);
                  stabilityAngle=rotAngle;
            }
      }
break;

case Sensor.TYPE_GYROSCOPE:
break;

case Sensor.TYPE_MAGNETIC_FIELD:
break;
}

if(currentStage == TAKE_PICTURE){
if(takingPicture==false){
if (picturesTaken == 0){
if (Math.abs(angleXLR-90)<5 && Math.abs(angleXFB-87.8)<10){
      changeColor(true);
}else{
```

```
      changeColor(false);
}
} else if(picturesTaken > 0){
if ((Math.abs(angleXLR-90)<5)&&(Math.abs(Math.abs(firstPictureAngle)-
Math.abs(angleXFB)))<5){
      changeColor(true);
}else{
      changeColor(false);
}
}}}else if (currentStage == MEASURE_DISTANCE){
      if(picturesTaken > 0){
      if(Math.abs(Math.abs(savedAngleXFB)-
            Math.abs(angleXFB))<2.5){

      saveDistanceButton.setBackgroundColor(Color.GREEN);
                        saveDistanceButton.setTextColor(Color.BLUE);
                  }else{

      saveDistanceButton.setBackgroundColor(Color.RED);
                        saveDistanceButton.setTextColor(Color.RED);

                  }
            }else{
                  saveDistanceButton.setBackgroundColor(Color.GRAY);
                  saveDistanceButton.setTextColor(Color.RED);
            }
        }
            //tvZoomValue.setText(String.valueOf(angleXFB-87.8)); Parece
que el plano no lo hace exactamente 90
    }

      @Override
      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }

      /**Functions to display Toasts efficiently*/
      public void displayToast(String message) {
          if(toast != null){
              toast.cancel();
          }
          toast = Toast.makeText(getApplicationContext(),
            message, Toast.LENGTH_SHORT);
          toast.show();
      }

      public void displayToastLong(String message) {
            //They last more time than the normal toasts
          if(toast != null){
              toast.cancel();
          }
          toast = Toast.makeText(getApplicationContext(),
            message, Toast.LENGTH_LONG);
          toast.show();
      }
```

```
	/**Function to change the color effect to NONE or NEGATIVE*/
	public void changeColor(boolean valid){

		if (firstTime){ //We have to come in at least once;
			validAux=!(valid);
			firstTime = false;
		}

		//To avoid change camera parameters all the time
		if(valid!=validAux){
			if (myCamera!=null){
				params = myCamera.getParameters();
			if (valid){

	params.setColorEffect(Camera.Parameters.EFFECT_NONE);
				}else{

	params.setColorEffect(Camera.Parameters.EFFECT_NEGATIVE);

				} //Close if-else
				myCamera.setParameters(params);
				validAux = valid;
				if(captureButtonNeeded){
					if (validAux == false){
						captureButton.setEnabled(false);
					}else{
						captureButton.setEnabled(true);
					}
				}
			}
		}
	} //Close changeColor

	/**MENUS*/
	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if it
is present.
		getMenuInflater().inflate(R.menu.pics_allower, menu);
		return true;
	}

	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
	    // Handle item selection
	    switch (item.getItemId()) {

		case R.id.menu_iso:
			if(currentStage != TAKE_PICTURE){

			displayToast(getString(R.string.finish_the_step));
			}else{
				currentParameter = PARAMETER_ISO;
				sb.setMax(4);
```

```
                    //number of values - 1 (3200 excluded)
                    visibilityFunction(CHANGE_ALFA);
              }
              return true;

        case R.id.menu_focal:
              if(currentStage != TAKE_PICTURE){

              displayToast(getString(R.string.finish_the_step));
              }else{
                    currentParameter = PARAMETER_FOCAL;

                        Log.i("FocalValueMenu",
                          String.valueOf(focalValue));

                    sb.setMax(fv.getLength(zoomValue)-1);
                    //number of values - 1
                    tvHeight.setText(String.valueOf(focalValue));
                    visibilityFunction(CHANGE_ALFA);
              }
              return true;
        case R.id.menu_time:
              if(currentStage != TAKE_PICTURE){

displayToast(getString(R.string.finish_the_step));
              }else{
                    currentParameter = PARAMETER_TIME;
                    sb.setMax(13);
                    //number of values - 1 [>1/100 excluded]
                    visibilityFunction(CHANGE_ALFA);
              }
              return true;

          case R.id.change_alfa_menu:
           if(currentStage != TAKE_PICTURE){
                 displayToast(getString(R.string.finish_the_step));
              }else{
                    if (picturesTaken == 0){

              displayToast(getString(R.string.no_pictures_yet));
                    }else{
                          sb.setMax(100);
                          currentParameter = PARAMETER_ALFA;
                          visibilityFunction(CHANGE_ALFA);
                    }
              }
           return true;

          case R.id.measure_distance_menu:
              if(measureDistance){
                    if(currentStage != TAKE_PICTURE){

              displayToast(getString(R.string.finish_the_step));
                    }else{
                          sb.setMax(maxHeight);
```

```
                        visibilityFunction(MEASURE_DISTANCE);
                    }
                }else{

                displayToastLong(
                        getString(R.string.no_distance_selected));
                }
                return true;

            case R.id.finish_menu:
              picsToStitch.setText(String.valueOf(picturesTaken) + " "
                + getString(R.string.pictures_taken));
                fullPic.setAlpha((float)0);
                dialog.show();

                /*params.set("zoom", 1);
                    params.set("curr_zoom_level", 1);
                    myCamera.setParameters(params);*/
                visibilityFunction(FINISHED);
                return true;

            case R.id.take_all:
              takeAllPictures();
                return true;

            default:
                return super.onOptionsItemSelected(item);
        }
    }

    protected void onActivityResult(int requestCode, int resultCode,
            Intent data) {

            if (requestCode == CANNY_CALL) {
            if (resultCode == RESULT_OK) {
                    Bundle extras = data.getExtras();
                    if (extras!= null) {
                        iCannyLowerThreshold1 =
                        extras.getInt("newLower");
                        iCannyUpperThreshold1 =
                        extras.getInt("newUpper");
                        iKernelSize1 = extras.getInt("newKernelSize");
                    }
            }
          }
    }

    /**GUI for doing everything step by step*/
    public void visibilityFunction(int code){
            currentStage = code;
            switch(code){
            case DISTANCE_INSTRUCTIONS:
                    Log.i("Stage", "DISTANCE INSTRUCTIONS");
                    //TextViews
                    valuesColumn.setVisibility(View.INVISIBLE);
```

```
                picsToStitch.setText(" ");
                picsToStitch.setVisibility(View.GONE);
                tvHeight.setVisibility(View.INVISIBLE);
                distanceInstructions.setVisibility(View.VISIBLE);

        tvInstructionsTitle.setText(getString(R.string.title_distance));

        tvInstruction1.setText(getString(R.string.distance_instruction_1))
;

        tvInstruction2.setText(getString(R.string.distance_instruction_2))
;

        tvInstruction3.setText(getString(R.string.distance_instruction_3))
;

        //Button

        distanceInstructionsOKButton.setVisibility(View.VISIBLE);
        distanceInstructionsOKButton.setEnabled(true);
        saveDistanceButton.setVisibility(View.GONE);
        saveDistanceButton.setEnabled(false);
        startButton.setBackgroundColor(Color.GRAY);
        startButton.setVisibility(View.INVISIBLE);
        startButton.setEnabled(false);
        if(captureButtonNeeded){
              captureButton.setVisibility(View.GONE);
              captureButton.setEnabled(false);
              }

        //ImageViews
        fullPic.setVisibility(View.INVISIBLE);
        topLine.setVisibility(View.INVISIBLE);
        bottomLine.setVisibility(View.INVISIBLE);
        horizontalRedLine.setVisibility(View.INVISIBLE);
        verticalRedLine.setVisibility(View.INVISIBLE);
        blackScreen.setVisibility(View.VISIBLE);
        blackScreen.setAlpha(0.8f);
        //Too see a bit of the camera (I think it's a cool effect)


        //SeekBar
        sb.setVisibility(View.INVISIBLE);
        sb.setEnabled(false);
        break;

        case MEASURE_DISTANCE:
                    Log.i("Stage", "MEASURE DISTANCE");
                    //Always with real color, not negative
                    changeColor(true);

                    //TextViews
                    valuesColumn.setVisibility(View.VISIBLE);
                tvZoomValue.setVisibility(View.VISIBLE);
                if (picturesTaken>0){
```

```
            tvEstimatedDistance.setText(
                  getString(R.string.previous_gnd_dist) +
                  String.format("%.2f", savedGroundDistance) + "
                  m");
            tvGroundDistance.setText(
            getString(R.string.current_gnd_dist) +
            String.format("%.2f", GndDistance) + " m");
            saveDistanceButton.setText(getString(R.string.ok));
        }else if (picturesTaken == 0){
                //Initial Zoom x1.2

saveDistanceButton.setText(getString(R.string.save_distance));


        }

        tvEstimatedDistance.setVisibility(View.GONE);
        tvGroundDistance.setVisibility(View.GONE);

        tvHeight.setVisibility(View.INVISIBLE);
            picsToStitch.setVisibility(View.GONE);
            distanceInstructions.setVisibility(View.INVISIBLE);

            //Button
            distanceInstructionsOKButton.setVisibility(View.GONE);
            distanceInstructionsOKButton.setEnabled(false);
            saveDistanceButton.setVisibility(View.VISIBLE);
            saveDistanceButton.setEnabled(true);
            startButton.setVisibility(View.INVISIBLE);
            startButton.setEnabled(false);
            if(captureButtonNeeded){
                  captureButton.setVisibility(View.GONE);
                  captureButton.setEnabled(false);
            }
            //ImageViews
            fullPic.setVisibility(View.VISIBLE);
        topLine.setVisibility(View.INVISIBLE);
        bottomLine.setVisibility(View.INVISIBLE);
        horizontalRedLine.setVisibility(View.VISIBLE);
        verticalRedLine.setVisibility(View.VISIBLE);
        blackScreen.setVisibility(View.INVISIBLE);

        //SeekBar
        sb.setVisibility(View.GONE);

        break;

      case TAKE_PICTURE:
            Log.i("Stage", "TAKE PICTURE");
            //TextViews
            valuesColumn.setVisibility(View.VISIBLE);
            if (picturesTaken == 0){

saveDistanceButton.setText(getString(R.string.save_distance));
```

```
                    if (!firstChain){

displayToastLong(getString(R.string.min_zoom_alert));
                    }
            }

            tvZoomValue.setVisibility(View.VISIBLE);
            tvEstimatedDistance.setVisibility(View.GONE);
            tvGroundDistance.setVisibility(View.GONE);
            tvHeight.setVisibility(View.INVISIBLE);
                picsToStitch.setVisibility(View.VISIBLE);
                distanceInstructions.setVisibility(View.INVISIBLE);

                //Button
                distanceInstructionsOKButton.setVisibility(View.GONE);
                distanceInstructionsOKButton.setEnabled(false);
                saveDistanceButton.setVisibility(View.GONE);
                saveDistanceButton.setEnabled(false);
                startButton.setVisibility(View.INVISIBLE);
                startButton.setEnabled(false);
                if(captureButtonNeeded){
                      captureButton.setVisibility(View.VISIBLE);
                      captureButton.setEnabled(true);
                }

                //ImageViews
                fullPic.setVisibility(View.VISIBLE);
            topLine.setVisibility(View.VISIBLE);
            bottomLine.setVisibility(View.VISIBLE);
            horizontalRedLine.setVisibility(View.INVISIBLE);
            verticalRedLine.setVisibility(View.INVISIBLE);
            blackScreen.setVisibility(View.INVISIBLE);

            //SeekBar
            sb.setVisibility(View.INVISIBLE);
            sb.setEnabled(false);
            if (picturesTaken == 0){
                displayToastLong(getString(R.string.touch_to_focus));
            }

            break;

          case FINISHED:
                Log.i("Stage", "FINISHED");
                //Always with real color, not negative
                changeColor(true);

                //TextViews
                valuesColumn.setVisibility(View.INVISIBLE);
                tvHeight.setVisibility(View.INVISIBLE);
                picsToStitch.setVisibility(View.VISIBLE);
                distanceInstructions.setVisibility(View.INVISIBLE);

                //Button
                distanceInstructionsOKButton.setVisibility(View.GONE);
```

```
            distanceInstructionsOKButton.setEnabled(false);
            saveDistanceButton.setVisibility(View.GONE);
            saveDistanceButton.setEnabled(false);
            startButton.setVisibility(View.VISIBLE);
            startButton.setEnabled(true);
            startButton.setText(getString(R.string.start_again));
            if(captureButtonNeeded){
                  captureButton.setVisibility(View.GONE);
                  captureButton.setEnabled(false);
            }

            //ImageViews
            fullPic.setVisibility(View.INVISIBLE);
        topLine.setVisibility(View.INVISIBLE);
        bottomLine.setVisibility(View.INVISIBLE);
        horizontalRedLine.setVisibility(View.INVISIBLE);
        verticalRedLine.setVisibility(View.INVISIBLE);
        blackScreen.setVisibility(View.INVISIBLE);


        //SeekBar
        sb.setVisibility(View.INVISIBLE);
        sb.setEnabled(false);
        break;

    case CHANGE_ALFA:
            Log.i("Stage", "CHANGE ALFA");
            //Always with real color, not negative
            changeColor(true);

            //TextViews
            valuesColumn.setVisibility(View.INVISIBLE);
            tvHeight.setVisibility(View.VISIBLE);
            picsToStitch.setVisibility(View.VISIBLE);
            distanceInstructions.setVisibility(View.INVISIBLE);

            //Button
            distanceInstructionsOKButton.setVisibility(View.GONE);
            distanceInstructionsOKButton.setEnabled(false);
            saveDistanceButton.setVisibility(View.VISIBLE);
            saveDistanceButton.setEnabled(true);
            saveDistanceButton.setBackgroundColor(Color.GRAY);
            saveDistanceButton.setTextColor(Color.RED);
            saveDistanceButton.setText(getString(R.string.ok));
            startButton.setVisibility(View.GONE);
            startButton.setEnabled(false);
            if(captureButtonNeeded){
                  captureButton.setVisibility(View.GONE);
                  captureButton.setEnabled(false);
            }

            //ImageViews
            fullPic.setVisibility(View.VISIBLE);
            topLine.setVisibility(View.INVISIBLE);
            bottomLine.setVisibility(View.INVISIBLE);
```

```
            horizontalRedLine.setVisibility(View.INVISIBLE);
            verticalRedLine.setVisibility(View.INVISIBLE);
            blackScreen.setVisibility(View.INVISIBLE);


        //SeekBar

        sb.setVisibility(View.VISIBLE);
        sb.setEnabled(true);
        switch(currentParameter){
        case PARAMETER_ALFA:
            sb.setProgress((int)(alfaValue*100f));
            tvHeight.setText(
            String.valueOf((int)(alfaValue*100)));
            startButton.setVisibility(View.VISIBLE);
                  startButton.setEnabled(true);
                  startButton.setText(
            getString(R.string.change_color));
            break;
        case PARAMETER_ISO:
            sb.setProgress(isoList.indexOf(isoValue));
            tvHeight.setText(String.valueOf(isoValue));
            break;
        /*
        case PARAMETER_EXPOSURE:
            sb.setProgress(exposureValue+6);
            tvHeight.setText(exposureList.get(exposureValue+6));
            break;
        */
        case PARAMETER_FOCAL:
            Log.i("Zoom", String.valueOf(zoomValue));
            focalValue = fv.getNewFocalValue(
            zoomValue, focalValue);
            sb.setProgress(fv.getBarProgress(
            zoomValue, focalValue));
            tvHeight.setText(String.valueOf(focalValue));
            Log.i("FocalValueVis", String.valueOf(focalValue));
            //params.set("aperture", Math.round(focalValue*10));
            break;
        case PARAMETER_TIME:
            sb.setProgress(timeValue);
            tvHeight.setText(timeList.get(timeValue));
            break;
        }
        break;
      }
}

public void updateZoom(int value){
      params = myCamera.getParameters();
      params.set("zoom", value);
      params.set("current-zoom", value);
      myCamera.setParameters(params);
    params = myCamera.getParameters();
}
```

```
@SuppressLint("SimpleDateFormat")
public void ImageStitching(int numberOfPics){

      final int numOfPics = numberOfPics;
      Log.i("ImageStitching", "INIT");
      stitcher = Stitcher.createDefault(false);
      images = new MatVector(numOfPics);
      //images = new MatVector(4);

      result = new IplImage(null);

      uiCallback = new Handler () {
          public void handleMessage (Message msg) {
            if (msg.what == 0){
                  displayToastLong(getString(R.string.stitching));
            }else if (msg.what == 1){
            displayToastLong(getString(R.string.stitched_saved));
            }else if (msg.what == 2){
            displayToastLong(getString(R.string.stitch_problem));
            }
          }
      };

      timerOF = new Thread(){
            public void run () {
                  try{
                 // do stuff in a separate thread
                        uiCallback.sendEmptyMessage(0);
                        Log.i("ImageStitching", "LOAD " +
                        String.valueOf(numOfPics) +" IMAGES");

                        for (int i=0; i<numOfPics; i++){
                        images.put(i,cvLoadImage(imageNames[i]));
                        }


Log.i("ImageStitching", "STITCH");

Log.i("VALUES",
                                          "registrationResol: " +
String.valueOf(stitcher.registrationResol())
                                        + "\nseamEstimationResol: " +
String.valueOf(stitcher.seamEstimationResol())
                                        + "\ncompositingResol: " +
String.valueOf(stitcher.compositingResol())
                                        + "\nwaveCorrection: " +
String.valueOf(stitcher.waveCorrection())
                                        + "\nwaveCorrectKind: " +
String.valueOf(stitcher.waveCorrectKind())
                                        + "\npanoConfidenceThresh: " +
String.valueOf(stitcher.panoConfidenceThresh()));
                              int status =
stitcher.stitch(images,result);
```

```
if( status == Stitcher.OK ){
      Log.i("ImageStitching", "SAVING");
      String timeStamp =
            new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
      imageNameAux = dirPath + File.separator + "PANO_"
            + timeStamp + ".JPG";
      cvSaveImage(imageNameAux, result);
      uiCallback.sendEmptyMessage(1);
}else{
      Log.i("ImageStitching", "NOT STITCHED");
      uiCallback.sendEmptyMessage(2);
}

Log.i("ImageStitching", "FINISHED");
stitching = false;
}catch(Exception e){
Log.i("Exception", String.valueOf(e));
                  }
              }
       };
       timerOF.start();
}

public void createDialog(){
//Dialog menu to choose if stitch or not
// 1. Instantiate an AlertDialog.Builder with its constructor
AlertDialog.Builder builder = new AlertDialog.Builder(this);
// 2. Chain together various setter methods to set the dialog
characteristics
builder.setMessage(R.string.stitch_question)
  .setTitle(R.string.stitching_title);
// Add the buttons
builder.setPositiveButton(R.string.yes, new
DialogInterface.OnClickListener() {
```

```
public void onClick(DialogInterface dialog, int id) {
      // User clicked YES button
      if(picturesTaken > 1){
            stitching = true;
            ImageStitching(picturesTaken);

      }else{
            displayToast(getString(R.string.cannot_stitch) + " "
            + String.valueOf(picturesTaken) + " "
            + getString(R.string.pictures_taken));
      }
      }
      });

builder.setNegativeButton(R.string.no, new
DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int id) {
      // User cancelled the dialog
}
});

// 3. Get the AlertDialog from create()
dialog = builder.create();
}

public void takeAllPictures(){
            params = myCamera.getParameters();
            updateZoom(1);
            zoomValue = 1;

            uiCallback = new Handler () {
                public void handleMessage (Message msg) {
                  if(myCamera != null){

      params.setColorEffect(Camera.Parameters.EFFECT_NONE);
                        myCamera.setParameters(params);
                        myCamera.takePicture(null, null, jpegCallBack);

      params.setColorEffect(Camera.Parameters.EFFECT_NONE);
                        myCamera.setParameters(params);
                  }
                }
            };

            timerOF = new Thread(){
                  int i, j, k, cuenta = 0;
                  public void run () {
                        try{
                              for(k=0; k< fv.getListSize(zoomValue);
                                    k++){
                                    for(j=0; j< timeList.size(); j++){
                                          for(i=0;
                                          i< (isoList.size()-1); i++){
                                                params.set(
```

```
                                                "mode", "m");
                                    isoValue = isoList.get(i);
                        params.set("iso", isoValue);
                        params.set("shutter-speed", j+32);
                        focalValue = fv.getFocalValue(zoomValue, k);
                        params.set("aperture",
                              Math.round(focalValue*10));

                        myCamera.setParameters(params);
                        params = myCamera.getParameters();

                        Log.i("iso", String.valueOf(isoValue));
                        Log.i("time", timeList.get(j));
                        Log.i("FocalValueBar",
                              String.valueOf(focalValue));

                        Thread.sleep(2500);
                        uiCallback.sendEmptyMessage(0);
                        cuenta = cuenta + 1;
                  Log.i("FOTO", String.valueOf(cuenta) + " de 700");
                  }
            }
      }
}catch(Exception e){
      Log.i("Exception", String.valueOf(e));
      }
}
};

timerOF.start();
}

public void detectCircles(){
      Imgproc.GaussianBlur(mGray, mGray, new Size(iKernelSize2,
            iKernelSize2), 6, 6);
            Imgproc.Canny(mGray, mGray, iCannyLowerThreshold2,
            iCannyUpperThreshold2);

            //Works better if we give the opposite Mat RGB instead of
                  BGR or viceversa
            Imgproc.HoughCircles(mGray, mIntermediateMat,
            Imgproc.CV_HOUGH_GRADIENT, 1.0,
            mGray.rows() / 32, iCannyUpperThreshold2,
            iAccumulator, iMinRadius, iMaxRadius);

          if (mIntermediateMat.cols() > 0)
              for (int x = 0; x < Math.min(mIntermediateMat.cols(),
                  1000); x++)
                  {
                  double vCircle[] = mIntermediateMat.get(0,x);

                  if (vCircle == null)
                      break;
```

```
if(Math.round(vCircle[1])>(mRgba.height()*UP_DOWN_MARGINS) &&
Math.round(vCircle[1])<(mRgba.height()*(1-UP_DOWN_MARGINS))){
pt = new org.opencv.core.Point(Math.round(vCircle[0]),
Math.round(vCircle[1]));
radius = (int)Math.round(vCircle[2]);
// draw the found circle
Core.circle(mRgba, pt, radius, colorRed, iLineThickness);

// draw a cross on the centre of the circle
Core.circle(mRgba, pt, 0, colorRed, 10);
}}}

public void makeCanny (){
// doing a gaussian blur prevents getting a lot of false hits (only for
Canny)

Imgproc.GaussianBlur(mGray, mGray, new Size(iKernelSize1,
      iKernelSize1), 8, 8);
Imgproc.Canny(mGray, mIntermediateMat, iCannyLowerThreshold1,
      iCannyUpperThreshold1);
}
}//Close Activity
```

7.2. MainActivity.java (Pantalla de inicio de la app)

```
package com.GDsergio.picsallower;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;

public class MainActivity extends Activity{
      private Button bYes, bNo;
      private Intent myIntent;

      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            /**FLAGS FOR THE SYSTEM*/
            this.requestWindowFeature(Window.FEATURE_NO_TITLE);

            this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FU
            LLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);

            this.getWindow().addFlags(WindowManager.LayoutParams.FLAG_KE
            EP_SCREEN_ON);
            setContentView(R.layout.main_activity);

            bYes = (Button)findViewById(R.id.button_yes);
            bYes.setOnClickListener(
            new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                  bYes.setEnabled(false);
                  bNo.setEnabled(false);

                  myIntent = new Intent(MainActivity.this ,
                  PicsAllower.class);
                  myIntent.putExtra("measureDistance", true);

                  startActivity(myIntent);
            }
      }
    );

    bNo = (Button)findViewById(R.id.button_no);
    bNo.setOnClickListener(
        new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                  bYes.setEnabled(false);
                  bNo.setEnabled(false);
                  myIntent = new Intent(MainActivity.this ,
```

```
                    PicsAllower.class);
                    myIntent.putExtra("measureDistance", false);

                    startActivity(myIntent);

                }
            }
        );


      }

       @Override
          public void onResume() {
            super.onResume();
            if (bYes.isEnabled()==false){bYes.setEnabled(true);}
            if (bNo.isEnabled()==false){bNo.setEnabled(true);}
       }

       @Override
            protected void onPause() {
                  super.onPause();
                  finish();
            // We are trying to kill the old activity here ...
            }
}
```

7.3. AngleCalculator.java (Función para calcular el ángulo)

```
package com.GDsergio.picsallower;

import java.math.RoundingMode;
import java.text.DecimalFormat;
import java.util.Timer;
import java.util.TimerTask;

import android.hardware.SensorEvent;
import android.hardware.SensorManager;
import android.os.Handler;

public class AngleCalculator {

      //private SensorManager mSensorManager = null;

    // angular speeds from gyro
    private float[] gyro = new float[3];
     // rotation matrix from gyro data
    private float[] gyroMatrix = new float[9];
     // orientation angles from gyro matrix
    private float[] gyroOrientation = new float[3];
     // magnetic field vector
    private float[] magnet = new float[3];
     // accelerometer vector
    private float[] accel = new float[3];
    // orientation angles from accel and magnet
    private float[] accMagOrientation = new float[3];
     // final orientation angles from sensor fusion
    private float[] fusedOrientation = new float[3];
    private float[] fusedOrientation2 = new float[3];
     // accelerometer and magnetometer based rotation matrix
    private float[] rotationMatrix = new float[9];

    public static final float EPSILON = 0.000000001f;
    private static final float NS2S = 1.0f / 1000000000.0f;
      private float timestamp;
      private boolean initState = true;

      public static final int TIME_CONSTANT = 30;
      public static final float FILTER_COEFFICIENT = 0.98f;
      private Timer fuseTimer = new Timer();

      // The following members are only for displaying the sensor
output.
      public Handler mHandler;
      DecimalFormat d = new DecimalFormat("#.##");


    AngleCalculator(){

        gyroOrientation[0] = 0.0f;
        gyroOrientation[1] = 0.0f;
        gyroOrientation[2] = 0.0f;
```

```
    // initialise gyroMatrix with identity matrix
    gyroMatrix[0] = 1.0f; gyroMatrix[1] = 0.0f; gyroMatrix[2] =
        0.0f;
    gyroMatrix[3] = 0.0f; gyroMatrix[4] = 1.0f; gyroMatrix[5] =
        0.0f;
    gyroMatrix[6] = 0.0f; gyroMatrix[7] = 0.0f; gyroMatrix[8] =
        1.0f;


    // wait for one second until gyroscope and
        magnetometer/accelerometer
    // data is initialised then scedule the complementary filter
        task
    fuseTimer.scheduleAtFixedRate(
        new calculateFusedOrientationTask(),
                                    1000, TIME_CONSTANT);

    // GUI stuff
    mHandler = new Handler();
    d.setRoundingMode(RoundingMode.HALF_UP);
    d.setMaximumFractionDigits(3);
    d.setMinimumFractionDigits(3);
}



  // calculates orientation angles from accelerometer and
        magnetometer output
  public void calculateAccMagOrientation(SensorEvent event) {
        System.arraycopy(event.values, 0, accel, 0, 3);
      if(SensorManager.getRotationMatrix(rotationMatrix, null,
        accel, magnet)) {
      SensorManager.getOrientation(rotationMatrix,
        accMagOrientation);
      }
  }

  // This function is borrowed from the Android reference
  // at
  http://developer.android.com/reference/android/hardware/SensorEven
  t.html#values
  // It calculates a rotation vector from the gyroscope angular
        speed values.
private void getRotationVectorFromGyro(float[] gyroValues,
        float[] deltaRotationVector,
        float timeFactor)
  {
        float[] normValues = new float[3];

        // Calculate the angular speed of the sample
        float omegaMagnitude =
        (float)Math.sqrt(gyroValues[0] * gyroValues[0] +
        gyroValues[1] * gyroValues[1] +
```

```
            gyroValues[2] * gyroValues[2]);

            // Normalize the rotation vector if it's big enough to get
            the axis
            if(omegaMagnitude > EPSILON) {
            normValues[0] = gyroValues[0] / omegaMagnitude;
            normValues[1] = gyroValues[1] / omegaMagnitude;
            normValues[2] = gyroValues[2] / omegaMagnitude;
            }

            // Integrate around this axis with the angular speed by the
            timestep
            // in order to get a delta rotation from this sample over
            the timestep
            // We will convert this axis-angle representation of the
            delta rotation
            // into a quaternion before turning it into the rotation
            matrix.
            float thetaOverTwo = omegaMagnitude * timeFactor;
            float sinThetaOverTwo = (float)Math.sin(thetaOverTwo);
            float cosThetaOverTwo = (float)Math.cos(thetaOverTwo);
            deltaRotationVector[0] = sinThetaOverTwo * normValues[0];
            deltaRotationVector[1] = sinThetaOverTwo * normValues[1];
            deltaRotationVector[2] = sinThetaOverTwo * normValues[2];
            deltaRotationVector[3] = cosThetaOverTwo;
        }

    // This function performs the integration of the gyroscope data.
    // It writes the gyroscope based orientation into gyroOrientation.
    public void gyroFunction(SensorEvent event) {
        // don't start until first accelerometer/magnetometer
orientation has been acquired
        if (accMagOrientation == null)
            return;

        // initialisation of the gyroscope based rotation matrix
        if(initState) {
            float[] initMatrix = new float[9];
            initMatrix =
getRotationMatrixFromOrientation(accMagOrientation);
            float[] test = new float[3];
            SensorManager.getOrientation(initMatrix, test);
            gyroMatrix = matrixMultiplication(gyroMatrix, initMatrix);
            initState = false;
        }

        // copy the new gyro values into the gyro array
        // convert the raw gyro data into a rotation vector
        float[] deltaVector = new float[4];
        if(timestamp != 0) {
            final float dT = (event.timestamp - timestamp) * NS2S;
        System.arraycopy(event.values, 0, gyro, 0, 3);
        getRotationVectorFromGyro(gyro, deltaVector, dT / 2.0f);
        }
```

```
            // measurement done, save current time for next interval
            timestamp = event.timestamp;

            // convert rotation vector into rotation matrix
            float[] deltaMatrix = new float[9];
            SensorManager.getRotationMatrixFromVector(deltaMatrix,
deltaVector);

            // apply the new rotation interval on the gyroscope based
rotation matrix
            gyroMatrix = matrixMultiplication(gyroMatrix, deltaMatrix);

            // get the gyroscope based orientation from the rotation matrix
            SensorManager.getOrientation(gyroMatrix, gyroOrientation);
        }

        private float[] getRotationMatrixFromOrientation(float[] o) {
            float[] xM = new float[9];
            float[] yM = new float[9];
            float[] zM = new float[9];

            float sinX = (float)Math.sin(o[1]);
            float cosX = (float)Math.cos(o[1]);
            float sinY = (float)Math.sin(o[2]);
            float cosY = (float)Math.cos(o[2]);
            float sinZ = (float)Math.sin(o[0]);
            float cosZ = (float)Math.cos(o[0]);

            // rotation about x-axis (pitch)
            xM[0] = 1.0f; xM[1] = 0.0f; xM[2] = 0.0f;
            xM[3] = 0.0f; xM[4] = cosX; xM[5] = sinX;
            xM[6] = 0.0f; xM[7] = -sinX; xM[8] = cosX;

            // rotation about y-axis (roll)
            yM[0] = cosY; yM[1] = 0.0f; yM[2] = sinY;
            yM[3] = 0.0f; yM[4] = 1.0f; yM[5] = 0.0f;
            yM[6] = -sinY; yM[7] = 0.0f; yM[8] = cosY;

            // rotation about z-axis (azimuth)
            zM[0] = cosZ; zM[1] = sinZ; zM[2] = 0.0f;
            zM[3] = -sinZ; zM[4] = cosZ; zM[5] = 0.0f;
            zM[6] = 0.0f; zM[7] = 0.0f; zM[8] = 1.0f;

            // rotation order is y, x, z (roll, pitch, azimuth)
            float[] resultMatrix = matrixMultiplication(xM, yM);
            resultMatrix = matrixMultiplication(zM, resultMatrix);
            return resultMatrix;
        }

        private float[] matrixMultiplication(float[] A, float[] B) {
            float[] result = new float[9];

            result[0] = A[0] * B[0] + A[1] * B[3] + A[2] * B[6];
            result[1] = A[0] * B[1] + A[1] * B[4] + A[2] * B[7];
            result[2] = A[0] * B[2] + A[1] * B[5] + A[2] * B[8];
```

```
        result[3] = A[3] * B[0] + A[4] * B[3] + A[5] * B[6];
        result[4] = A[3] * B[1] + A[4] * B[4] + A[5] * B[7];
        result[5] = A[3] * B[2] + A[4] * B[5] + A[5] * B[8];

        result[6] = A[6] * B[0] + A[7] * B[3] + A[8] * B[6];
        result[7] = A[6] * B[1] + A[7] * B[4] + A[8] * B[7];
        result[8] = A[6] * B[2] + A[7] * B[5] + A[8] * B[8];

        return result;
    }

    class calculateFusedOrientationTask extends TimerTask {
        public void run() {
            float oneMinusCoeff = 1.0f - FILTER_COEFFICIENT;

            /*
             * Fix for 179° <--> -179° transition problem:
             * Check whether one of the two orientation angles (gyro or
accMag) is negative while the other one is positive.
             * If so, add 360° (2 * math.PI) to the negative value,
perform the sensor fusion, and remove the 360° from the result
             * if it is greater than 180°. This stabilizes the output
in positive-to-negative-transition cases.
             */

            // azimuth
            if (gyroOrientation[0] < -0.5 * Math.PI &&
                  accMagOrientation[0] > 0.0) {
                  fusedOrientation[0] = (float) (FILTER_COEFFICIENT *
                  (gyroOrientation[0] + 2.0 * Math.PI) + oneMinusCoeff *
                  accMagOrientation[0]);
                  fusedOrientation[0] -= (fusedOrientation[0] > Math.PI)
                  ? 2.0 * Math.PI : 0;
            }
            else if (accMagOrientation[0] < -0.5 * Math.PI &&
                  gyroOrientation[0] > 0.0) {
                  fusedOrientation[0] = (float) (FILTER_COEFFICIENT *
                  gyroOrientation[0] + oneMinusCoeff *
                  (accMagOrientation[0] + 2.0 * Math.PI));
                  fusedOrientation[0] -= (fusedOrientation[0] >
                  Math.PI)? 2.0 * Math.PI : 0;
            }
            else {
                  fusedOrientation[0] = FILTER_COEFFICIENT *
                  gyroOrientation[0] + oneMinusCoeff *
                  accMagOrientation[0];
            }

            // pitch
            if (gyroOrientation[1] < -0.5 * Math.PI &&
                  accMagOrientation[1] > 0.0) {
                  fusedOrientation[1] = (float) (FILTER_COEFFICIENT *
                  (gyroOrientation[1] + 2.0 * Math.PI) + oneMinusCoeff *
                  accMagOrientation[1]);
```

```
                fusedOrientation[1] -= (fusedOrientation[1] > Math.PI)
                ? 2.0 * Math.PI : 0;
            }
            else if (accMagOrientation[1] < -0.5 * Math.PI &&
                gyroOrientation[1] > 0.0) {
                fusedOrientation[1] = (float) (FILTER_COEFFICIENT *
                gyroOrientation[1] + oneMinusCoeff *
                (accMagOrientation[1] + 2.0 * Math.PI));
                fusedOrientation[1] -= (fusedOrientation[1] >
            Math.PI)? 2.0 * Math.PI : 0;
            }
            else {
                fusedOrientation[1] = FILTER_COEFFICIENT *
                gyroOrientation[1] + oneMinusCoeff *
                accMagOrientation[1];
            }

            // roll
            if (gyroOrientation[2] < -0.5 * Math.PI &&
                accMagOrientation[2] > 0.0) {
                fusedOrientation[2] = (float) (FILTER_COEFFICIENT *
                (gyroOrientation[2] + 2.0 * Math.PI) + oneMinusCoeff *
                accMagOrientation[2]);
                fusedOrientation[2] -= (fusedOrientation[2] > Math.PI)
                ? 2.0 * Math.PI : 0;
            }
            else if (accMagOrientation[2] < -0.5 * Math.PI &&
            gyroOrientation[2] > 0.0) {
            fusedOrientation[2] = (float) (FILTER_COEFFICIENT *
            gyroOrientation[2] + oneMinusCoeff * (accMagOrientation[2] +
            2.0 * Math.PI));
            fusedOrientation[2] -= (fusedOrientation[2] > Math.PI)? 2.0
            * Math.PI : 0;
            }
            else {
            fusedOrientation[2] = FILTER_COEFFICIENT *
            gyroOrientation[2] + oneMinusCoeff * accMagOrientation[2];
            }

            // overwrite gyro matrix and orientation with fused
            orientation
            // to comensate gyro drift
            gyroMatrix =
            getRotationMatrixFromOrientation(fusedOrientation);
            System.arraycopy(fusedOrientation, 0, gyroOrientation, 0,
                3);


            // update sensor output in GUI
            //mHandler.post(updateOreintationDisplayTask);
        }
    }

    public boolean isValid(int picturesTaken, double
firstPictureAngle){
```

```
      //double angleDiff=0;
      double fused1 = Math.abs(fusedOrientation[1]* 180/Math.PI);
      double fused2 = Math.abs(fusedOrientation[2]* 180/Math.PI);



      if (picturesTaken>0){
            if((fused1<5)&&((Math.abs(firstPictureAngle)-
            Math.abs(fused2))<5){
                  return true;
            }else{
                  return false;
            }
      }else { //before the first picture
            if((fused1<5)){
                  return true;
            }else{
                  return false;
            }
      }
      }

    public double getAzimuth(){
      return Math.abs(fusedOrientation[0]* 180/Math.PI);
    }

    public float [] getFusedOrientation(){
      System.arraycopy(fusedOrientation, 0, fusedOrientation2, 0,
fusedOrientation.length);
      return fusedOrientation2;
    }

    public void setMagnet(SensorEvent event){
        System.arraycopy(event.values, 0, magnet, 0, 3);
    }
}
```

7.4. AutoFocusAdjust.java (Función para el enfoque automático)

```
package com.GDsergio.picsallower;

import java.util.ArrayList;
import java.util.List;

import android.graphics.Rect;
import android.hardware.Camera;
import android.hardware.Camera.AutoFocusCallback;
import android.view.MotionEvent;
import android.widget.FrameLayout;

public class AutoFocusAdjust {
    float xTouch=0, yTouch=0; //TO AVOID integer division
    float xnew=0, ynew=0;
    int left=0, top=0, right=0, bottom=0;
    ImageFocusCallback autoFocusCallBack = new ImageFocusCallback();
//Constructor
      AutoFocusAdjust(){
      }

      public void adjust(MotionEvent event, Camera camera, FrameLayout
            preview){
            Camera.Parameters params = camera.getParameters();
            xTouch = event.getX();
            yTouch = event.getY();
            switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:

//Convert from View's width and height to +/- 1000 "matrix"

                  xnew = (xTouch*2000/preview.getWidth())-1000;
                  ynew = (yTouch*2000/preview.getHeight())-1000;


                  /**CAMERA ADJUST TO FOCUS WHERE TOUCH*/
                  //Metering areas not supported :(
                  //Focus areas
                if (params.getMaxNumFocusAreas() > 0){ // check that
                                          focus areas are supported


                  /**Values for focus area rectangle*/
                  left = (int) xnew-200;
                  if(left<-1000){
                        left=-1000;
                  }
                    top = (int)ynew-200;
                    if(top<-1000){
                        top=-1000;
                  }
                    right = (int)xnew+200;
                    if(right>1000){
                        right=1000;
```

```
                    }
                      bottom = (int)ynew+200;
                      if(bottom>1000){
                          bottom=1000;
                      }

                      /**Creating focus area*/
                      Rect areaRect1 = new Rect(left, top, right, bottom);

                      List<Camera.Area> focusAreas =
                          new ArrayList<Camera.Area>();

                      focusAreas.add(new Camera.Area(areaRect1, 1000));
//100% weight
                          params.setFocusAreas(focusAreas);
                      camera.setParameters(params);
                      camera.autoFocus(autoFocusCallBack);

                  }



              case MotionEvent.ACTION_MOVE:
                case MotionEvent.ACTION_UP:

            }
        }

        public void adjustByButton(Camera camera){
          camera.autoFocus(autoFocusCallBack);
        }
        class ImageFocusCallback implements AutoFocusCallback {
            @Override
            public void onAutoFocus(boolean success, Camera camera) {
              //We can put here the green square when it is focused
            }
        }
}
```

7.5. CalculateDistance.java (Función que calcula la distancia)

```
package com.GDsergio.picsallower;

public class CalculateDistance {

	private float distance=0, stability=0, height, angle, angle2,
anglerad, anglerad2;
	private float coseno, tan, sin, diff = 0, diff2 = 0;
	private float gnddistance=0, gndstability=0, gndFlagStability=0,
ohStability=0;
	private float gndSavedDistance = 0, newDistance= 0, oh=0;
	private final float conversion = (float)(Math.PI/180);

	public CalculateDistance(){
	}

	public float getDistance(float h, float alfa) {
		height=h;
		angle=alfa;
		anglerad = angle*conversion;

		if(alfa<90f){
			coseno=(float)Math.cos(anglerad);
			distance=height/coseno;
		}else if(alfa>=90f){
			distance=100000;
		}

		if (distance<=2.8&&(Math.abs(stability-
distance)>(0.01*distance))){ //1% errormax
			stability=distance;
		}else if(distance>2.8&&(Math.abs(stability-
distance)>(0.03*distance))){
			stability=distance;
		}
		return stability;
	}

	public float getGndDistance(float h, float alfa) {
		height=h;
		angle=alfa;
		anglerad = angle*conversion;

		if(angle<90f){
			tan=(float)Math.tan(anglerad);
			gnddistance=height*tan;
		}else if(angle>=90f){
			gnddistance=100000;
		}


		if (gnddistance<=2.8&&(Math.abs(gndstability-
gnddistance)>(0.01*gnddistance))){ //1% errormax
			gndstability=gnddistance;
```

```
            }else if(gnddistance>2.8&&(Math.abs(gndstability-
gnddistance)>(0.03*gnddistance))){
                  gndstability=gnddistance;
            }
            return gndstability;
      }

      public float getFlagDistance(float gndDist, float alfa) {
            gndSavedDistance=gndDist;
            angle=alfa;
            angle2=angle-90;
            anglerad = angle*conversion;
            anglerad2 = angle2*conversion;

            if(angle<=90f){
                  sin=(float)Math.sin(anglerad);
                  newDistance=gndSavedDistance/sin;
            }else if(angle>90f){
                  coseno = (float)Math.cos(anglerad2);
                  newDistance = gndSavedDistance/coseno;
            }

            if (newDistance<=5&&(Math.abs(gndFlagStability-
            newDistance)>(0.01*newDistance))){ //1% errormax
                  gndFlagStability=newDistance;
            }else if(newDistance>5&&(Math.abs(gndFlagStability-
                  newDistance)>(0.03*newDistance))){
                  gndFlagStability=newDistance;
            }
            return gndFlagStability;
      }

      public float getObjectHeight(float gndDist, float h, float alfa) {
            gndSavedDistance=gndDist;
            angle=alfa;
            height = h;
            angle2=angle-90;
            anglerad = angle*conversion;
            anglerad2 = angle2*conversion;

            if(angle<90f){
                  tan = (float)Math.tan(anglerad);
                  diff = gndSavedDistance/tan;
                  oh = height-diff;
            }else if(angle==90f){
                  oh = height;
            }else if(angle>90f){
                  tan = (float)Math.tan(anglerad2);
                  diff2 = tan*gndSavedDistance;
                  oh = height + diff2;
            }

            if (oh<=2&&(Math.abs(ohStability-oh)>=(0.01))){
//1% errormax
                  ohStability=oh;
```

```
            }else if(oh>2&&(Math.abs(ohStability-oh)>(0.01*oh))){
                  ohStability=oh;
            }
            return ohStability;
      }
}
```

7.6. FocalValues.java (Calcula distancias focales según zoom)

```
package com.GDsergio.picsallower;

import java.util.ArrayList;
import java.util.Arrays;

public class FocalValues {

	/**Focal Distances to the different zoom Values*/
	private ArrayList <Float> focalList0 = //Zoom x1.0
			new ArrayList<Float> (Arrays.asList(
					2.8f, 3.2f, 3.5f, 4.0f, 4.5f, 5.0f, 5.6f,
					6.3f, 7.1f, 8.0f));
	private ArrayList <Float> focalList1 = //Zoom x1.2
			new ArrayList<Float> (Arrays.asList(
					2.9f, 3.3f, 3.7f, 4.1f, 4.6f, 5.2f, 5.8f,
					6.5f, 7.3f, 8.3f));
	private ArrayList <Float> focalList2 = //Zoom x1.5
			new ArrayList<Float> (Arrays.asList(
					3.0f, 3.4f, 3.8f, 4.3f, 4.8f, 5.4f, 6.1f,
					6.8f, 7.6f, 8.6f));
	private ArrayList <Float> focalList3 = //Zoom x1.8
			new ArrayList<Float> (Arrays.asList(
					3.2f, 3.6f, 4.0f, 4.5f, 5.0f, 5.6f, 6.3f,
					7.1f, 8.0f));
	private ArrayList <Float> focalList4 = //Zoom x2.2
			new ArrayList<Float> (Arrays.asList(
					3.4f, 3.8f, 4.2f, 4.7f, 5.3f, 5.9f, 6.7f,
					7.5f, 8.4f));
	private ArrayList <Float> focalList5 = //Zoom x2.8
			new ArrayList<Float> (Arrays.asList(
					3.6f, 4.0f, 4.5f, 5.0f, 5.6f, 6.3f, 7.1f,
					8.0f));
	private ArrayList <Float> focalList6 = //Zoom x3.4
			new ArrayList<Float> (Arrays.asList(
					3.8f, 4.2f, 4.7f, 5.3f, 5.9f, 6.6f, 7.5f,
					8.4f));
	private ArrayList <Float> focalList7 = //Zoom x4.0
			new ArrayList<Float> (Arrays.asList(
					3.9f, 4.3f, 4.9f, 5.4f, 6.1f, 6.9f, 7.7f,
					8.6f));
	private ArrayList <Float> focalList8 = //Zoom x5.0
			new ArrayList<Float> (Arrays.asList(
					4.0f, 4.5f, 5.1f, 5.7f, 6.4f, 7.2f,
					8.0f));
	private ArrayList <Float> focalList9 = //Zoom x6.1
			new ArrayList<Float> (Arrays.asList(
					4.2f, 4.8f, 5.3f, 6.0f, 6.7f, 7.5f,
					8.4f));
	private ArrayList <Float> focalList10 = //Zoom x7.5
			new ArrayList<Float> (Arrays.asList(
					4.5f, 5.0f, 5.6f, 6.3f, 7.1f, 7.9f,
					8.9f));
	private ArrayList <Float> focalList11 = //Zoom x9.4
```

```
            new ArrayList<Float> (Arrays.asList(
                        4.8f, 5.4f, 6.0f, 6.8f, 7.6f, 8.5f));
private ArrayList <Float> focalList12 = //Zoom x11.4
            new ArrayList<Float> (Arrays.asList(
                        5.1f, 5.7f, 6.4f, 7.2f, 8.1f));
private ArrayList <Float> focalList13 = //Zoom x13.9
            new ArrayList<Float> (Arrays.asList(
                        5.5f, 6.2f, 6.9f, 7.7f, 8.7f));
private ArrayList <Float> focalList14 = //Zoom x17.9
            new ArrayList<Float> (Arrays.asList(
                        5.8f, 6.7f, 7.4f, 8.4f));
private ArrayList <Float> focalList15 = //Zoom x21.0
            new ArrayList<Float> (Arrays.asList(
                        5.9f, 6.9f, 7.5f, 8.5f));

FocalValues(){
}

public int getLength(int zoomValue){
      switch(zoomValue){
      case 0:
            return focalList0.size();
      case 1:
            return focalList1.size();
      case 2:
            return focalList2.size();
      case 3:
            return focalList3.size();
      case 4:
            return focalList4.size();
      case 5:
            return focalList5.size();
      case 6:
            return focalList6.size();
      case 7:
            return focalList7.size();
      case 8:
            return focalList8.size();
      case 9:
            return focalList9.size();
      case 10:
            return focalList10.size();
      case 11:
            return focalList11.size();
      case 12:
            return focalList12.size();
      case 13:
            return focalList13.size();
      case 14:
            return focalList14.size();
      case 15:
            return focalList15.size();
      default:
            return 0;
      }
```

```
}

public int getBarProgress(int zoomValue, float focalValue){
	switch(zoomValue){
	case 0:
		return focalList0.indexOf(focalValue);
	case 1:
		return focalList1.indexOf(focalValue);
	case 2:
		return focalList2.indexOf(focalValue);
	case 3:
		return focalList3.indexOf(focalValue);
	case 4:
		return focalList4.indexOf(focalValue);
	case 5:
		return focalList5.indexOf(focalValue);
	case 6:
		return focalList6.indexOf(focalValue);
	case 7:
		return focalList7.indexOf(focalValue);
	case 8:
		return focalList8.indexOf(focalValue);
	case 9:
		return focalList9.indexOf(focalValue);
	case 10:
		return focalList10.indexOf(focalValue);
	case 11:
		return focalList11.indexOf(focalValue);
	case 12:
		return focalList12.indexOf(focalValue);
	case 13:
		return focalList13.indexOf(focalValue);
	case 14:
		return focalList14.indexOf(focalValue);
	case 15:
		return focalList15.indexOf(focalValue);
	default:
		return 0;
	}
}

public float getFocalValue(int zoomValue, int progress){
	switch(zoomValue){
	case 0:
		return focalList0.get(progress);
	case 1:
		return focalList1.get(progress);
	case 2:
		return focalList2.get(progress);
	case 3:
		return focalList3.get(progress);
	case 4:
		return focalList4.get(progress);
	case 5:
		return focalList5.get(progress);
```

```
            case 6:
                  return focalList6.get(progress);
            case 7:
                  return focalList7.get(progress);
            case 8:
                  return focalList8.get(progress);
            case 9:
                  return focalList9.get(progress);
            case 10:
                  return focalList10.get(progress);
            case 11:
                  return focalList11.get(progress);
            case 12:
                  return focalList12.get(progress);
            case 13:
                  return focalList13.get(progress);
            case 14:
                  return focalList14.get(progress);
            case 15:
                  return focalList15.get(progress);
            default:
                  return 0;
            }
      }

      public float getNewFocalValue(int zoomValue, float focalValue){

            switch(zoomValue){
            case 0:
                  for (int i=0; i<(focalList0.size()-1); i++){
                        if (focalValue <= focalList0.get(i)){
                              return focalList0.get(i);
                        }else if((focalValue > focalList0.get(i)) &&
(focalValue <= focalList0.get(i+1))){
                              return focalList0.get(i+1);
                        }else if (i == (focalList0.size()-2) &&
(focalValue > focalList0.get(i+1))){
                              return focalList0.get(i+1);
                        }
                  }
            case 1:
                  for (int i=0; i<(focalList1.size()-1); i++){
                        if (focalValue <= focalList1.get(i)){
                              return focalList1.get(i);
                        }else if((focalValue > focalList1.get(i)) &&
(focalValue <= focalList1.get(i+1))){
                              return focalList1.get(i+1);
                        }else if (i == (focalList1.size()-2) &&
(focalValue > focalList1.get(i+1))){
                              return focalList1.get(i+1);
                        }
                  }
            case 2:
                  for (int i=0; i<(focalList2.size()-1); i++){
                        if (focalValue <= focalList2.get(i)){
```

```
                        return focalList2.get(i);
                    }else if((focalValue > focalList2.get(i)) &&
(focalValue <= focalList2.get(i+1))){
                        return focalList2.get(i+1);
                    }else if (i == (focalList2.size()-2) &&
(focalValue > focalList2.get(i+1))){
                        return focalList2.get(i+1);
                    }
                }
            case 3:
                for (int i=0; i<(focalList3.size()-1); i++){
                    if (focalValue <= focalList3.get(i)){
                        return focalList3.get(i);
                    }else if((focalValue > focalList3.get(i)) &&
(focalValue <= focalList3.get(i+1))){
                        return focalList3.get(i+1);
                    }else if (i == (focalList3.size()-2) &&
(focalValue > focalList3.get(i+1))){
                        return focalList3.get(i+1);
                    }
                }
            case 4:
                for (int i=0; i<(focalList4.size()-1); i++){
                    if (focalValue <= focalList4.get(i)){
                        return focalList4.get(i);
                    }else if((focalValue > focalList4.get(i)) &&
(focalValue <= focalList4.get(i+1))){
                        return focalList4.get(i+1);
                    }else if (i == (focalList4.size()-2) &&
(focalValue > focalList4.get(i+1))){
                        return focalList4.get(i+1);
                    }
                }
            case 5:
                for (int i=0; i<(focalList5.size()-1); i++){
                    if (focalValue <= focalList5.get(i)){
                        return focalList5.get(i);
                    }else if((focalValue > focalList5.get(i)) &&
(focalValue <= focalList5.get(i+1))){
                        return focalList5.get(i+1);
                    }else if (i == (focalList5.size()-2) &&
(focalValue > focalList5.get(i+1))){
                        return focalList5.get(i+1);
                    }
                }
            case 6:
                for (int i=0; i<(focalList6.size()-1); i++){
                    if (focalValue <= focalList6.get(i)){
                        return focalList6.get(i);
                    }else if((focalValue > focalList6.get(i)) &&
(focalValue <= focalList6.get(i+1))){
                        return focalList6.get(i+1);
                    }else if (i == (focalList6.size()-2) &&
(focalValue > focalList6.get(i+1))){
                        return focalList6.get(i+1);
```

```
                        }
                  }
            case 7:
                  for (int i=0; i<(focalList7.size()-1); i++){
                        if (focalValue <= focalList7.get(i)){
                              return focalList7.get(i);
                        }else if((focalValue > focalList7.get(i)) &&
(focalValue <= focalList7.get(i+1))){
                              return focalList7.get(i+1);
                        }else if (i == (focalList7.size()-2) &&
(focalValue > focalList7.get(i+1))){
                              return focalList7.get(i+1);
                        }
                  }
            case 8:
                  for (int i=0; i<(focalList8.size()-1); i++){
                        if (focalValue <= focalList8.get(i)){
                              return focalList8.get(i);
                        }else if((focalValue > focalList8.get(i)) &&
(focalValue <= focalList8.get(i+1))){
                              return focalList8.get(i+1);
                        }else if (i == (focalList8.size()-2) &&
(focalValue > focalList8.get(i+1))){
                              return focalList8.get(i+1);
                        }
                  }
            case 9:
                  for (int i=0; i<(focalList9.size()-1); i++){
                        if (focalValue <= focalList9.get(i)){
                              return focalList9.get(i);
                        }else if((focalValue > focalList9.get(i)) &&
(focalValue <= focalList9.get(i+1))){
                              return focalList9.get(i+1);
                        }else if (i == (focalList9.size()-2) &&
(focalValue > focalList9.get(i+1))){
                              return focalList9.get(i+1);
                        }
                  }
            case 10:
                  for (int i=0; i<(focalList10.size()-1); i++){
                        if (focalValue <= focalList10.get(i)){
                              return focalList10.get(i);
                        }else if((focalValue > focalList10.get(i)) &&
(focalValue <= focalList10.get(i+1))){
                              return focalList10.get(i+1);
                        }else if (i == (focalList10.size()-2) &&
(focalValue > focalList10.get(i+1))){
                              return focalList10.get(i+1);
                        }
                  }
            case 11:
                  for (int i=0; i<(focalList11.size()-1); i++){
                        if (focalValue <= focalList11.get(i)){
                              return focalList11.get(i);
```

```
                        }else if((focalValue > focalList11.get(i)) &&
(focalValue <= focalList11.get(i+1))){
                              return focalList11.get(i+1);
                        }else if (i == (focalList11.size()-2) &&
(focalValue > focalList11.get(i+1))){
                              return focalList11.get(i+1);
                        }
                  }
            case 12:
                  for (int i=0; i<(focalList12.size()-1); i++){
                        if (focalValue <= focalList12.get(i)){
                              return focalList12.get(i);
                        }else if((focalValue > focalList12.get(i)) &&
(focalValue <= focalList12.get(i+1))){
                              return focalList12.get(i+1);
                        }else if (i == (focalList12.size()-2) &&
(focalValue > focalList12.get(i+1))){
                              return focalList12.get(i+1);
                        }
                  }
            case 13:
                  for (int i=0; i<(focalList13.size()-1); i++){
                        if (focalValue <= focalList13.get(i)){
                              return focalList13.get(i);
                        }else if((focalValue > focalList13.get(i)) &&
(focalValue <= focalList13.get(i+1))){
                              return focalList13.get(i+1);
                        }else if (i == (focalList13.size()-2) &&
(focalValue > focalList13.get(i+1))){
                              return focalList13.get(i+1);
                        }
                  }
            case 14:
                  for (int i=0; i<(focalList14.size()-1); i++){
                        if (focalValue <= focalList14.get(i)){
                              return focalList14.get(i);
                        }else if((focalValue > focalList14.get(i)) &&
(focalValue <= focalList14.get(i+1))){
                              return focalList14.get(i+1);
                        }else if (i == (focalList14.size()-2) &&
(focalValue > focalList14.get(i+1))){
                              return focalList14.get(i+1);
                        }
                  }
            case 15:
                  for (int i=0; i<(focalList15.size()-1); i++){
                        if (focalValue <= focalList15.get(i)){
                              return focalList15.get(i);
                        }else if((focalValue > focalList15.get(i)) &&
(focalValue <= focalList15.get(i+1))){
                              return focalList15.get(i+1);
                        }else if (i == (focalList15.size()-2) &&
(focalValue > focalList15.get(i+1))){
                              return focalList15.get(i+1);
                        }
```

```
                }
            default:
                return 0;
            }
        }

        public float getListSize(int zoomValue){
            switch(zoomValue){
            case 0:
                return focalList0.size();
            case 1:
                return focalList1.size();
            case 2:
                return focalList2.size();
            case 3:
                return focalList3.size();
            case 4:
                return focalList4.size();
            case 5:
                return focalList5.size();
            case 6:
                return focalList6.size();
            case 7:
                return focalList7.size();
            case 8:
                return focalList8.size();
            case 9:
                return focalList9.size();
            case 10:
                return focalList10.size();
            case 11:
                return focalList11.size();
            case 12:
                return focalList12.size();
            case 13:
                return focalList13.size();
            case 14:
                return focalList14.size();
            case 15:
                return focalList15.size();
            default:
                return 0;
            }
        }
}
```

7.7. ImageResizer.java

```
package com.GDsergio.picsallower;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;

public class ImageResizer {

	ImageResizer(){

	}

	public static Bitmap decodeSampledBitmapFromResource(
		String nameAux,int reqWidth, int reqHeight) {

	    // First decode with inJustDecodeBounds=true to check
dimensions
	    final BitmapFactory.Options options =
	      new BitmapFactory.Options();
	    options.inJustDecodeBounds = true;
	    BitmapFactory.decodeFile(nameAux, options);

	    // Calculate inSampleSize
	    options.inSampleSize = calculateInSampleSize(
	      options, reqWidth, reqHeight);

	    // Decode bitmap with inSampleSize set
	    options.inJustDecodeBounds = false;
	    //options.outHeight = 720;
	    //options.outWidth = 960;
	    options.outHeight = 480;
	    options.outWidth = 640;
	    return BitmapFactory.decodeFile(nameAux, options);
	}


	public static int calculateInSampleSize(
		BitmapFactory.Options options, int reqWidth, int reqHeight)
{
    // Raw height and width of image
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;
    if (height > reqHeight || width > reqWidth) {

        // Calculate ratios of height and width to requested height and
width
        final int heightRatio = Math.round((float) height / (float)
reqHeight);
        final int widthRatio = Math.round((float) width / (float)
reqWidth);

        // Choose the smallest ratio as inSampleSize value, this will
guarantee
```

```
        // a final image with both dimensions larger than or equal to
the
        // requested height and width.
        inSampleSize = heightRatio < widthRatio ? heightRatio :
widthRatio;
    }

    return inSampleSize;
      }
}
```