UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Aplicación en Android para la medida de la velocidad en el entrenamiento deportivo con peso

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Aragó Palero, Joan Francesc

Tutor/a: Mossi García, José Manuel

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

PORTADA GENERADA AUTOMÁTICAMENTE EN EBRON

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Agradecimientos

El camino que he recorrido durante estos últimos meses no siempre ha sido plácido, pero gracias a las personas que me han acompañado, puedo decir que ha valido la pena cada día invertido.

En primer lugar, agradecer profundamente a mi tutor Jose Manuel Mossi todo el tiempo que me ha dedicado, gracias por hacer que disfrute programando a pesar de que en un primer momento te dije no sería capaz de hacerlo; gracias por hacerme sentir respaldado y haberme apoyado en todos los baches que he ido encontrando. El primer día que me reuní contigo te dije que yo no me planteaba otra opción que no fuera hacer el trabajo contigo y cinco meses después puedo decirte que ha sido la mejor decisión académica que he tomado en mi vida, ¡gracias!

En segundo lugar, quiero agradecer a mis amigos, mi otra familia, todo el apoyo incondicional que me han ofrecido desde el primer momento. Gracias a ti Marc por haberme apoyado desde la distancia y confiar ciegamente en que lo conseguiría. Gracias a ti Josep por abrazarme y decirme lo orgulloso que estabas de mí. Gracias a ti Pau por ser el primero que se ofreció a probar la aplicación y ayudarme en la fase final. Gracias a ti Aaron por estar siempre ahí, siempre. Gracias a todos, de corazón.

Por último, agradecer a los pilares de mi vida que siempre me sustentan cuando me tambaleo y nunca me dejan caer. Gracias a mis abuelos que sin entender nada de lo que hecho siempre me sonríen y presumen orgullosos de que su nieto va a convertirse en ingeniero. Gracias a mi hermano que sé que está muy orgulloso de su hermano mayor, tanto como yo lo estoy de él, y, por encima de todo, gracias a mis padres, gracias a ti papa, gracias a ti mamá, sin vosotros nunca lo hubiera conseguido. Gracias por vuestros consejos, gracias por hacerme sentir libre de llorar y de reír y gracias por haberme acompañado en ambas situaciones. Esto en gran parte es vuestro, ¡gracias!

Si desde allí arriba podéis leer este mensaje, gracias a vosotros dos también. Ojalá que veáis lo grande que me hecho. Siempre os llevaré conmigo.

“Los grandes logros de cualquier persona generalmente dependen de muchas manos, corazones y mentes.”

-Walt Disney

Resumen

Las técnicas más recientes de entrenamiento deportivo y de ejercicios de rehabilitación con levantamiento de peso están tomando como parámetro de referencia la velocidad a la que se ejecuta el movimiento. El objetivo principal de este proyecto es realizar esta medida mediante un sistema de visión artificial utilizando la cámara de un teléfono móvil.

Para ello se ha diseñado una aplicación utilizando el sistema operativo Android, que incluye el código necesario para poder hacer el seguimiento de la pesa, con el fin de obtener una medida en verdadera magnitud del desplazamiento y derivar la velocidad y aceleración instantánea presentando los datos en modo gráfico.

Resum

Les tècniques mes recents de entrenament deportiu i de exercicis de rehabilitament amb alçament de pes están prenent com a paràmetres de referencia la velocitat a la que realitza el moviment. Aquest projecte proposa realizar aquesta medida mitjançant un sistema de visió artificial utilitzant la cámara d'un telèfon móvil.

La aplicació es desarrollarà mitjansant el sistema operatiu Android. Incloura el còdic necessari per al seguiment de la pesa per a obtindre una medida en magnitut verdadera del desplaçament i obtindre la velocitat i la acceleració instantànea i presentant les dades en mode gràfic.

Abstract

The most recent techniques in sports training and rehabilitation exercises with weightlifting are taking the speed at which the movement is executed as a reference parameter. This project proposes to carry out this measurement by means of an artificial vision system using the camera of a mobile phone.

The application will be developed on the Android operating system. It will include the necessary code to track the weight, normalize to obtain a true magnitude measurement of the displacement, and derive the instantaneous velocity and acceleration and present the data in graphical mode.

Índice

Contenido

Índice de ilustraciones

Contenido

Índice de flujogramas

Contenido

Índice de partes de código

Contenido

Capítulo 1. Introducción

La aplicación desarrollada en este trabajo forma parte de un proyecto más amplio, en torno al análisis con visión artificial de ejercicios deportivos, en el que participan en colaboración el departamento de comunicaciones de la Universitat Politècnica de València y la Facultad de Educación Física de la Universidad de Alicante. El responsable en la UPV es el Dr. D. José Manuel Mossi.

En este trabajo en concreto, se aporta al proyecto global el desarrollo de una aplicación para dispositivos móviles Android, que permita grabar en tiempo real, a una persona realizando un determinado ejercicio con pesas. La aplicación, permitirá mostrar mediante gráficas el movimiento del disco, extrayendo la velocidad, la aceleración y la fuerza ejercida durante los movimientos.

Esta aplicación móvil pretende servir de guía para el deportista, ya que podrá aportarle información valiosa en tiempo real sobre la realización del ejercicio de manera efectiva y adecuada, al poder mostrarle, entre otros datos, si la velocidad de las repeticiones se mantiene constante, aumenta o disminuye y si la fuerza empleada va disminuyendo conforme va realizando el entrenamiento.

1.1 Deporte y salud: entrenamiento de fuerza

El deporte hoy en día se puede considerar un fenómeno social que va en aumento y que se ha visto reflejado en la realidad de lo cotidiano. Millones de personas en todo el mundo practican deporte diariamente, en espacios naturales, calles de ciudades y entornos específicos de entrenamiento. Deporte y salud van interrelacionados conformando dos ámbitos de sumo interés para las personas.

Dentro de las diversas actividades físicas asociadas a una mejora de la salud y a una prevención en el deterioro físico marcado por la edad, encontramos el entrenamiento de fuerza, que se considera el más importante para evitar la pérdida de masa muscular que viene acompañando al envejecimiento [1]. Los estudios científicos afirman que el entrenamiento de fuerza es el que mayor impacto tiene sobre el sistema musculoesquelético, siendo el más eficaz frente a la debilidad y fragilidad ósea. Entre sus múltiples beneficios se destacan la prevención del envejecimiento, reducción de estrés y ansiedad, mejora en la movilidad articular y reducción del colesterol [2].

Existen diferentes ejercicios en entrenamientos de fuerza, que pueden realizarse tanto con pesas como con máquinas. Aunque a primera vista las máquinas de gimnasio pueden ser más atractivas, los estudios afirman que el entrenamiento con pesas puede tener más beneficios y, sobre todo, aportan una mejor accesibilidad a dicho entrenamiento, ya que las pesas son más asequibles y ofrecen al usuario tanto la posibilidad de poder transportar el kit de entrenamiento allá donde se desplace, o, realizar una gran cantidad de ejercicios en casa.

En el entrenamiento con pesas, los movimientos pueden realizarse en tres dimensiones, por tanto, en todos los planos que existen, pudiendo realizar movilizaciones laterales, verticales o diagonales, solicitando mayor cantidad de grupos musculares, implicando tanto a los músculos sinergistas[1] como a los fijadores[2], haciendo un trabajo muscular mucho más completo. Sin embargo, los movimientos de las máquinas están acotados en uno o dos planos de movimiento, reduciendo la posibilidad de ejercitar distintos grupos musculares [3]. Independientemente de si se utilizan pesas o máquinas de gimnasio, en un entrenamiento con pesas es fundamental el control sobre el ejercicio y que este se ejecute adecuadamente con una buena técnica, con el fin de minimizar las posibles lesiones al tiempo que se consigue un ejercicio lo más eficiente posible.

[1] Músculos sinergistas: Grupo muscular encargado del movimiento.
[2] Músculos fijadores: Grupo muscular encargado de la estabilización

Capítulo 2. Objetivos

2.1 Objetivo general

Desarrollar una aplicación móvil funcional que consiga detectar, seguir y analizar el movimiento realizado en un ejercicio de pesas, permitiendo al usuario obtener información a tiempo real, sobre la idoneidad del ejercicio que realiza, mediante la visualización de las gráficas de datos de velocidad, aceleración y fuerza empleadas.

2.2 Objetivos secundarios

- Detección, seguimiento y análisis del movimiento con una única cámara.
- Diseñar una interfaz gráfica atractiva para el usuario.
- Lograr una navegación entre pantallas lógica e intuitiva.
- Diseñar alertas y mensajes que ayuden al usuario.
- Introducir de manera correcta los datos relacionados con la pesa.
- Mantener el buen rendimiento de la aplicación.
- Obtener medidas reales del movimiento.
- Comprobar su correcto funcionamiento con distintas pesas.

Capítulo 3. Estado del arte

El enorme avance y desarrollo en el campo de las tecnologías experimentadas en los últimos años, se ha materializado (entre otros aspectos) en el diseño de miles de aplicaciones móviles. Estas aplicaciones abarcan gran multitud de temáticas.

Uno de los mercados de oferta de aplicaciones que más interés despierta en el usuario es el relacionado con la salud y el deporte, existiendo un amplio catálogo de aplicaciones distintas en función del objetivo perseguido.

Ilustración 1. Aplicaciones "Salud" en *AppleStore*

Hoy en día el uso de aplicaciones relacionadas con la salud y el deporte son habituales, monitorización de ejercicios, aplicaciones que cuentan pasos y calorías consumidas durante el ejercicio, entrenadores virtuales, etc. son algunas de las aplicaciones más descargadas y utilizadas [4].

En nuestro caso, el tipo de aplicaciones que nos interesa, son aquellas relacionadas directamente con el ejercicio con pesas. Estas aplicaciones permiten al usuario conocer cuál ha sido su eficacia durante la realización de un ejercicio con pesas, permitiendo extraer información del movimiento realizado, como por ejemplo puede ser la velocidad a la que ha levantado el peso en cada repetición.

Una de las aplicaciones en la que nos hemos basado ha sido *WL Analysis* [5] pero esta no realiza una segmentación automática de las repeticiones.

3.1 Contexto tecnológico general de aplicaciones relacionadas con detección de objetos.

Las aplicaciones móviles utilizan librerías basadas en tecnologías que consiguen dotar a los móviles de funciones avanzadas más allá del campo de la telefonía. Las que mayor relación tienen con el desarrollo de este proyecto son las siguientes:

3.1.1 Machine Learning

El *Machine Learning* [6] o aprendizaje automático es una disciplina del campo de la Inteligencia Artificial que, gracias a distintos algoritmos, proporciona a los ordenadores la capacidad de identificar patrones y realizar análisis predictivo, esto permite que las computadoras puedan realizar tareas de manera autónoma.

Esta técnica se ha convertido en una herramienta necesaria para multitud de compañías de diferentes sectores. Las aplicaciones que han desarrollado no dejan de evolucionar, buscando siempre conseguir la mejor experiencia posible para el usuario.

Existen distintos métodos de *Machine Learning*, aunque son dos de ellos los que más se han utilizado, a saber, el aprendizaje supervisado y el aprendizaje no supervisado [7].

3.1.2 Aprendizaje Supervisado

Los algoritmos son entrenados usando muestras de las que se conoce la clase a la que pertenecen. Por ejemplo, un conjunto de piezas que van etiquetadas con una "V" (de válida) o "NV" (de no válida). De esta manera, el algoritmo recibe esta información, con la que luego comparará el siguiente conjunto de datos con los resultados correctos almacenados, aprendiendo las diferencias entre las etiquetas que hacen que un cierto dato sea correcto o no. Usando ciertos métodos distintos como el aumento del gradiente o la clasificación entre ellos el aprendizaje supervisado utilizará patrones para predecir los valores de las etiquetas en datos que no estén etiquetados.

Este tipo de aprendizaje es usado en aplicaciones donde los datos históricos predecirán eventos futuros probables.

3.1.3 Aprendizaje No Supervisado

Este método de aprendizaje se usa con datos que no están etiquetados, de esta manera no se le da al sistema cual es la respuesta correcta por lo que es el algoritmo quien debe descubrir de que tipo es el dato mostrado. El objetivo consiste en encontrar una estructura interna entre los conjuntos de datos que permita clasificarlos en función de sus atributos.

Este método necesitará de una posterior revisión humana para asegurarse que la clasificación se ha realizado de manera correcta.

Uno de los usos más comunes es en el campo del marketing donde se extraen patrones masivos provenientes de aplicaciones externas, como por ejemplo las redes sociales, para crear campañas de publicidad.

3.2 Visión artificial

Los seres humanos usamos nuestros ojos para entender nuestro entorno. La visión artificial [8] es una disciplina científica que busca que los ordenadores obtengan la misma capacidad, es decir, que las máquinas puedan percibir y entender imágenes para actuar de una manera u otra.

La visión artificial engloba todas las aplicaciones industriales y no industriales que combinan software y hardware dando orientación operativa a distintos dispositivos a la hora de ejecutar sus funciones, basándose en captura y procesamiento de imágenes.

Para que un sistema de visión artificial funcione de manera correcta necesita que se cumplan los siguientes requisitos:

- Buena y constante iluminación.
- Colocación correcta del objeto a examinar (puesta en escena).
- Uso de cámaras adecuadas.
- Uso de un procesador de imagen (modelo inteligente).
- Comunicación entre los dispositivos que conformen el sistema.

3.2.1 TensorFlow

TensorFlow [9] es una plataforma de código abierto de extremo a extremo usado para el aprendizaje automático disponible para los distintos lenguajes de programación como C++, Java o Python.

Esta tecnología se usa principalmente para el desarrollo de aplicaciones de visión artificial usando el aprendizaje automático. De esta manera se permite entrenar y crear modelos neuronales de manera sencilla para la detección de objetos.

Algunos de los ejemplos de aplicación de *TensorFlow* son los siguientes [10]:

1. Mejorar la fotografía de smartphones: En los móviles de Google se incluyen efectos como el *bokeh.* Se crea un modo retrato que consigue separar la persona del fondo con el uso de una sola cámara. Hasta el momento, este mismo efecto era imposible de realizar sin usar mínimo dos cámaras simultáneamente.

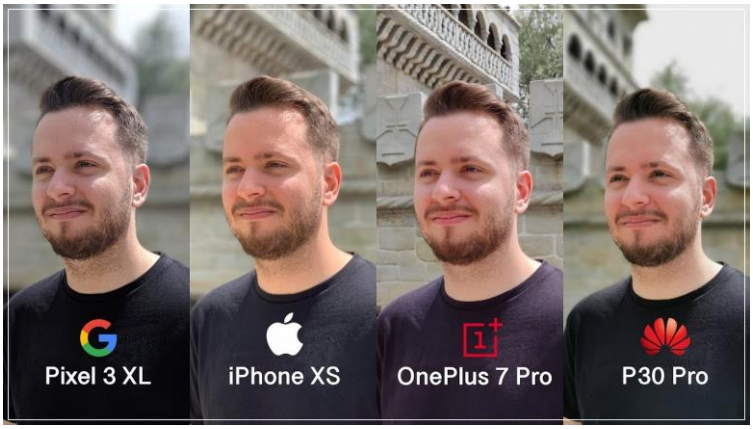


Ilustración 2. Comparativa modo Retrato de distintos dispositivos móviles.

Fuente: https://www.movilzona.es/2019/06/21/movil-mejor-modo-retrato/

2. Procesar imágenes: El procesamiento de imágenes es una de las aplicaciones más conocidas de TensorFlow.
   Uno de los más conocidos es *DeepDream*, creado por un ingeniero de Google llamado Alexander Mordvintsev, que usa una red neuronal convolucional para encontrar y mejorar patrones en imágenes mediante la pareidolia algorítmica, creando imágenes sobre procesadas similares a un sueño.

Ilustración 3. Imagen creada con DeepDream.

Fuente: Pinterest.es

3. Mejorar rendimiento de un dispositivo móvil: Google ha decidido integrar TensorFlow en los servicios de Google Play directamente para reducir el tamaño de las aplicaciones mejorando el rendimiento óptimo del dispositivo. El aprendizaje automático permite la aceleración del hardware cuando se use inteligencia artificial en las aplicaciones descargadas [11].

3.2.2 Amazon Rekognition

Amazon Rekognition [12] ofrece el uso de visión artificial previamente entrenada y personalizable para extraer información de imágenes y videos. Este procesamiento de imágenes permite detectar caras, objetos, movimientos e incluso gestos o contenido inapropiado para su censura.

Esta herramienta ofrece al usuario infinidad de opciones que permite una gran accesibilidad, permitiendo que cualquier tipo de usuario pueda usar esta tecnología.

Algunas de las aplicaciones más conocidas que ofrece *Amazon Rekognition* son las siguientes:

1. Detección y análisis de rostros: Se detectan los rostros que aparecen en imágenes o videos y se reconocen sus atributos, como por ejemplo si los ojos están abiertos o la apertura de la boca, para extraer información sobre el estado anímico de la persona detectada.

Ilustración 4. Análisis de rostro.

Fuente: https://aws.amazon.com/es/rekognition/

2. Moderación de contenido: Se analiza el contenido de la imagen mostrada por un dispositivo que incluya esta tecnología para detectar contenido potencialmente inseguro o inadecuado para ciertos usuarios.

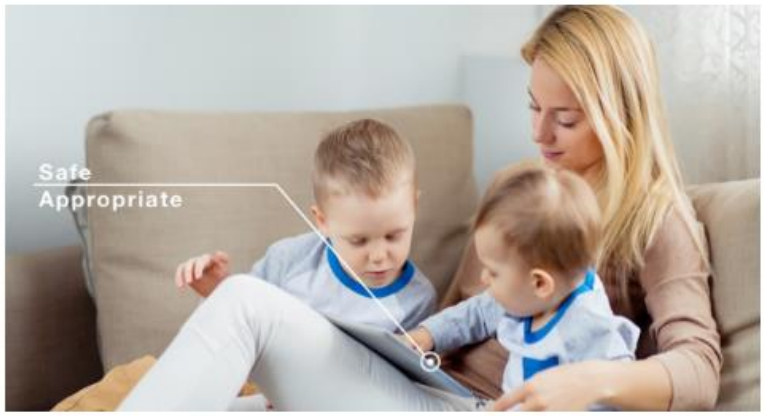


Ilustración 5. Moderación de contenido.

Fuente: https://aws.amazon.com/es/rekognition/

3. Reconocimiento de famosos: Se identifica a personas famosas para catalogar las fotos y videos en los que sale en los medios de comunicación. Esto se usa en el ámbito del marketing y la publicidad.

Ilustración 6. Reconocimiento de famosos.

Fuente: https://aws.amazon.com/es/rekognition/

3.2.3 OpenCV

OpenCV [13] es una biblioteca disponible para los distintos tipos de lenguajes como son C++, Java y Python entre otros, compatibles a la vez con los distintos sistemas operativos como Windows, Linux, Mac, iOS y Android.

El procesamiento de imágenes y videos es uno de los puntos fuertes de las librerías que contiene *OpenCV*, especialmente las relacionadas con el procesamiento de imágenes en tiempo real.

Algunas de las aplicaciones más conocidas que ofrece *OpenCV* son las siguientes:

1. Reconocimiento facial: Estas librerías han sido instaladas en redes sociales como Facebook permitiendo reconocer a las personas en las fotos que comparten los usuarios.

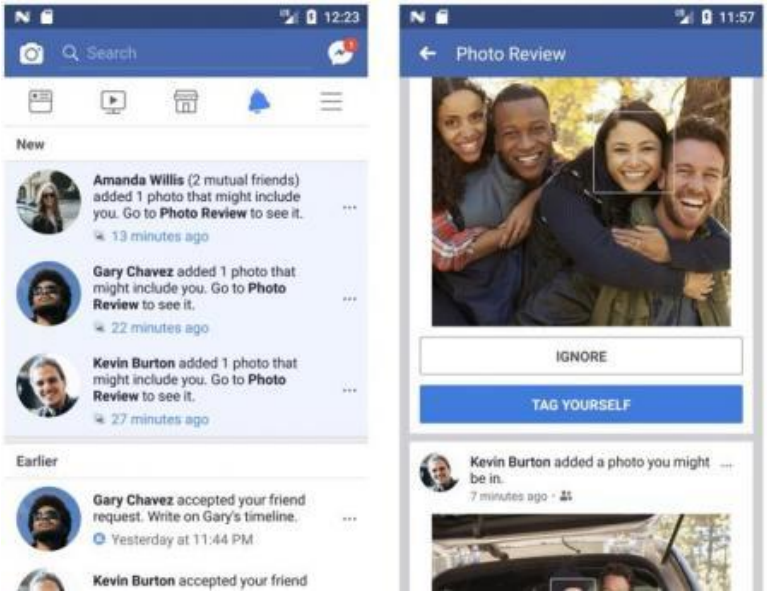


Ilustración 7. Reconocimiento facial Facebook.

Fuente: LaVanguardia.es

2. Reconocimiento de objetos: *OpenCV* contiene más de 500 funciones que abarcan el área de proceso de visión como lo es el reconocimiento de objetos. Este ofrece la posibilidad de buscar imágenes que contengan objetos parecidos mediante el uso de aprendizaje automático.

Ilustración 8. Reconocimiento de personas *OpenCV*.

Fuente: ***https://blog.desdelinux.net/opencv-una-biblioteca-para-el-reconocimiento-de-objetos-en-imagenes-y-camaras/***

3. Tracking: El primer paso para llevar a cabo el tracking es la detección de un objeto con cierta fiabilidad. Posteriormente existen muchas técnicas con las que detectar contornos o colores que permitirán seguir el movimiento del objeto.

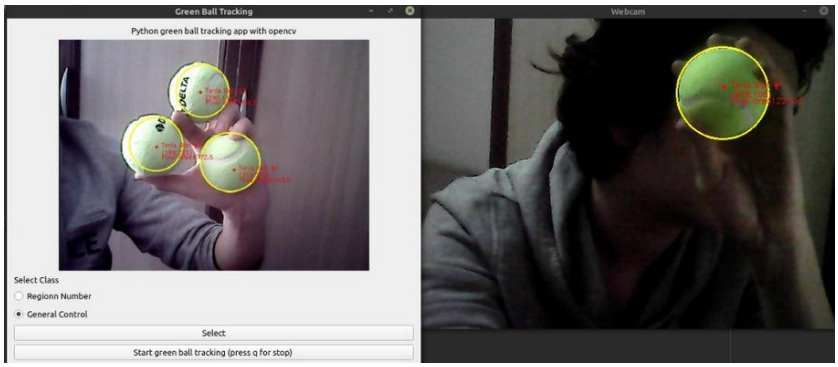


Ilustración 9. Tracking de objetos *OpenCV*.

Fuente: Youtube.es

3.3 Contexto tecnológico de la aplicación desarrollada.

3.3.1 Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para usarse en dispositivos Android. Está basado en IntelliJ DEA, que ofrece un editor potente de códigos y herramientas para desarrolladores [14].

Este entorno ofrece funciones que aumentan la productividad del usuario al desarrollar aplicaciones, como las siguientes:

- Sistema de compilación flexible (*Gradle*).
- Ayudas y consejos de optimización.
- Emulador rápido y con infinidad de funciones.
- Entorno unificado que permite desarrollar para todo tipo de dispositivos Android.

- Se permite aplicar cambios mientras se ejecuta la aplicación.
- Integración con *Github* y plantillas de código
- Compatibilidad con C++ y NDK[3]

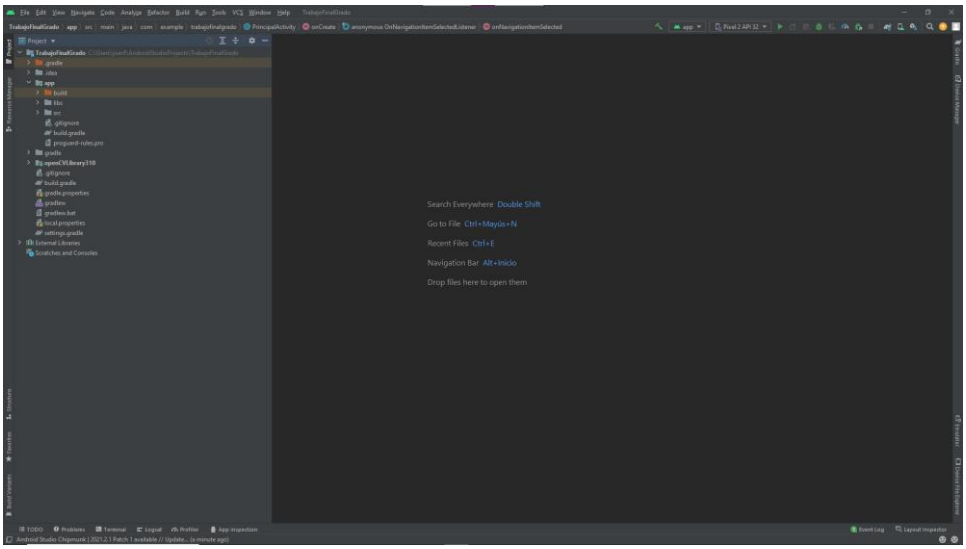


Ilustración 10. Pantalla principal de Android Studio

3.3.2 Java

Java es un lenguaje de programación orientado a objetos incorporado en el ámbito informático en los noventa. Actualmente, es uno de los lenguajes de programación más demandado y el escogido en esta ocasión para el diseño de la aplicación [15].

Este lenguaje ofrece versatilidad e intuición, ya que puede ser ejecutado desde cualquier plataforma creando programas complejos y usando métodos básicos. Una de las mayores ventajas que ofrece es la compatibilidad con Google y la cantidad de ayuda e información que se puede obtener a raíz de las herramientas proporcionadas por dicha empresa.

3.3.3 Librerías externas

En el desarrollo de esta aplicación ha sido necesaria la introducción de algoritmos externos a Android Studio, que han permitido crear una actividad donde se detectará un objeto y se pueda realizar el seguimiento de éste para obtener sus coordenadas.

Se investigó acerca de distintas librerías que pudieran ayudar con dicho propósito y se eligió *OpenCV* por su fácil manejo. Se probó *TensorFlow Lite* entre otras.

TensorFlow es un conjunto de herramientas que permite implementar el aprendizaje automático integrado en el dispositivo. En el desarrollo de aplicación se descargó un modelo existente de TensorFlow (que se ofrecía en la página web oficial) para comprobar que se podían detectar y seguir objetos.

Este modelo estaba totalmente cerrado y no permitía ajustarlo a nuestras necesidades por lo que se decidió solo tomarlo como referencia y utilizar la librería de *OpenCV* para desarrollar nuestros propios algoritmos.

[3] NDK: Kit de desarrollo nativo de Android que permite incorporar código de C++.

Ilustración 11. Icono de *OpenCV*.

***Fuente:* https://opencv.org/**

Tras probar la librería nativa de Android Studio para el diseño de gráficas y ver la escasez de recursos estéticos y técnicos, y descubrir la poca flexibilidad disponible respecto a diferentes formatos de datos con los que se podía trabajar y representar los valores obtenidos, se decidió investigar otras bibliotecas de algoritmos para llevar a cabo esta parte de la aplicación.

MPCharts fue la librería escogida. Esta fue creada por Phil Jay y se proporciona su código de uso libre en *Github.*

Ilustración 12. Icono de *MpCharts* para Android.

Fuente: https://github.com/PhilJay/MPAndroidChart

3.3.4 OpenCV

Tras muchos años de desarrollo, la inteligencia artificial se ha convertido en una tecnología imprescindible en la actualidad. Cada vez son más las empresas que optan por incluir dicha técnica en sus productos.

En la aplicación desarrollada, el mecanismo que se implementa es la visión artificial.

OpenCV es una biblioteca de código abierto [16] que contiene implementaciones con más de 2500 algoritmos especializados en visión artificial y *machine learning*[4]. Actualmente se podría decir que es la biblioteca de visión artificial más grande del mundo en cuanto a funciones, además de ser completamente gratuita.

Unas cuantas acciones que es capaz de realizar *OpenCV* son:

- Identificar caras u objetos (utilizado en esta aplicación).
- Relacionar imágenes similares.
- Eliminar ojos rojos.
- Clasificar acciones humanas.
- Realidad aumentada.
- Extraer modelos 3D.

Esta biblioteca de librerías ofrece todas las funciones que nuestra aplicación demanda y aunque *OpenCV* no es la única que encaja con el desarrollo del proyecto, sí que brinda ventajas respecto a sus rivales:

- Es multiplataforma, capaz de ejecutarse en los principales sistemas operativos, para poder ejecutarlo en Windows y Android.
- Se puede escribir en Java, así como en C++ y Python.

[4] Machine Learning: Disciplina del campo de la visión artificial que dota a los ordenadores de análisis predictivo.

- Se ofrecen nuevas actualizaciones que mejoran y añaden nuevos algoritmos.
- Es completamente gratuita.

Una vez decidido el uso de *OpenCV* se instaló en Android Studio.

Tras mucho ensayo y error, se escogió la versión 3.10, aunque luego se actualizaron a mano algunos de los códigos [17].

Siguiendo las instrucciones de instalación de la página web de *OpenCV* no se logró su correcta implementación, así que se optó por seguir otras instrucciones con las que se consiguió instalar.

Los pasos para su instalación fueron los siguientes:

1. Descargar el *sdk* [5]oficial de la página opencv.org

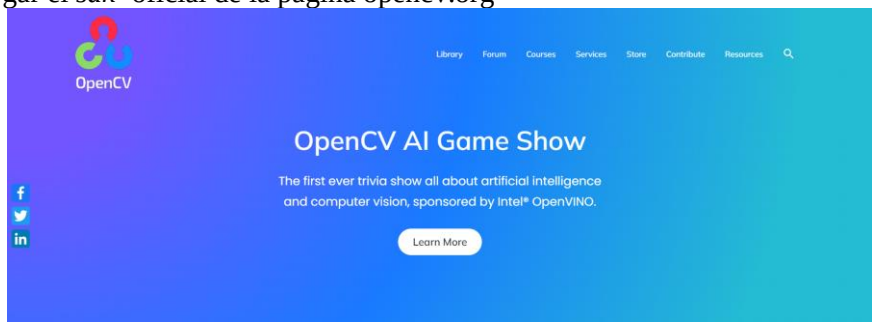


Ilustración 13. Página web de *OpenCV*.

***Fuente:* https://opencv.org/**

2. Extraer la carpeta con los algoritmos y copiarla a mano en la dirección que contiene el proyecto de la aplicación.

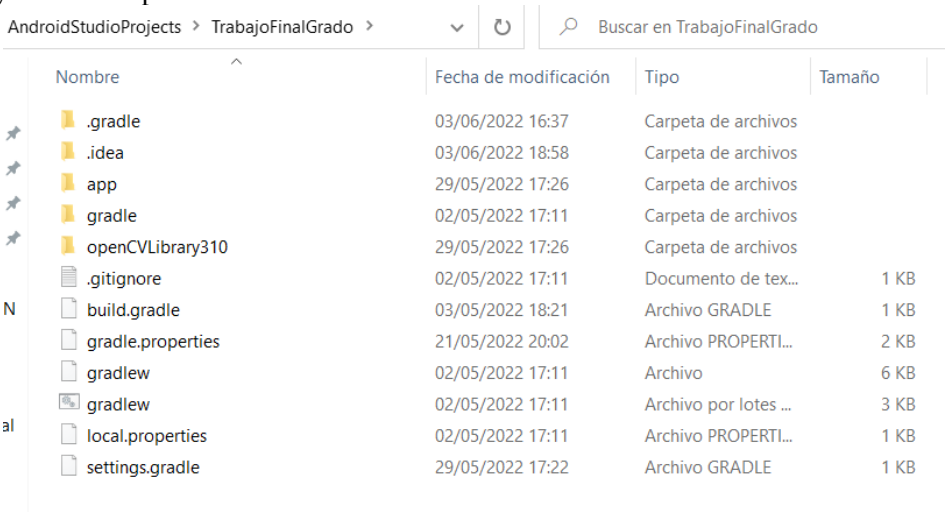


Ilustración 14. Carpeta de *OpenCV* en el proyecto

[5] SDK: Kit de desarrollo de software. Conjunto de herramientas que ayudan a desarrollar aplicaciones.

3. Con Android Studio abierto, incluir a mano el nombre de la carpeta en el fichero *settings.gradle* y actualizar el *Gradle*.

```
settings.gradle (TrabajoFinalGrado)
pluginManagement { PluginManagementSpec it ->
    repositories { RepositoryHandler it ->
        gradlePluginPortal()
        google()
        mavenCentral()
    }
}
dependencyResolutionManagement { DependencyResolutionManagement it ->
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories { RepositoryHandler it ->
        google()
        mavenCentral()
        jcenter()
        maven { url 'https://jitpack.io' }
    }
}
rootProject.name = "TrabajoFinalGrado"
include ':app'
include ':openCVLibrary310'
```

Parte de código 1. Incluir la carpeta de *OpenCV* en Android Studio

4. Acceder al menú de *Project Structure* y en el submenú de *Dependencies* importar la dependencia del módulo de *OpenCV*.

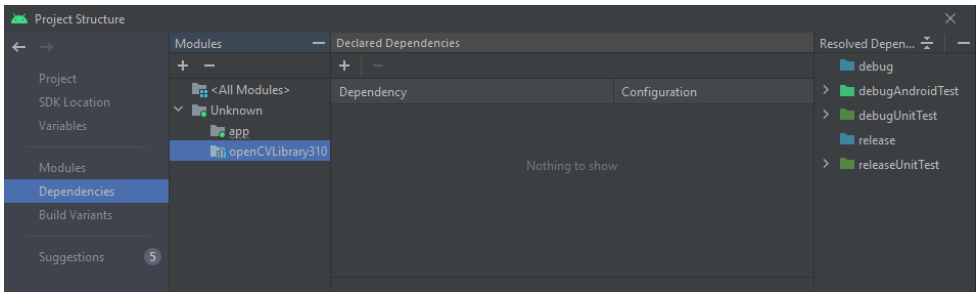


Ilustración 15. Implementación del módulo

5. Al confirmar que la dependencia del módulo de *OpenCV* está correctamente implementada, solo falta crear la carpeta *jniLibs* [6]en el paquete de código de la aplicación y copiar a mano los *libs* del *sdk* descargado de la página web.

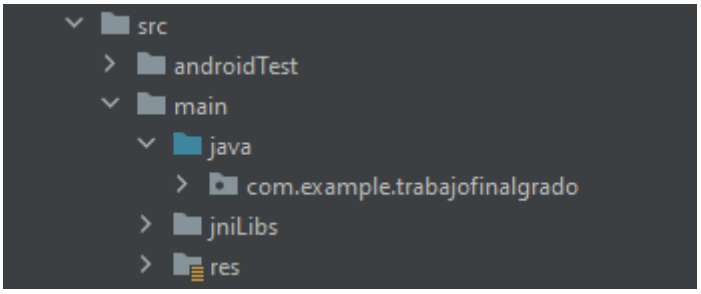


Ilustración 16. Creación de la carpeta *jniLibs*

Siguiendo todos los pasos [18] se logró su instalación y se creó una sencilla aplicación que abría una vista previa de la cámara en la que se procesaban los *frames*[7] en tiempo real. A raíz de esto, se fue construyendo poco a poco la aplicación.

[6] jniLibs: Biblioteca de desarrollo utilizado por Java Native Interface.
[7] Frames: Fotograma.

3.3.5 MPCharts

Una vez logrado el almacenamiento de las coordenadas del movimiento de la pesa en listas de valores hay que diseñar las gráficas que representan el movimiento *trackeado*.[8] Para ello se instaló la librería *MPCharts* en Android Studio. [19]

Su instalación resultó sencilla siguiendo los pasos del propio creador:

1. Configuración del *Gradle*.

```
34  dependencies {
35
36      def activity_version = "1.3.0"
37      def fragment_version = "1.3.6"
38      //noinspection GradleDependency
39      implementation "androidx.activity:activity-ktx:$activity_version"
40      //noinspection GradleDependency
41      implementation "androidx.fragment:fragment-ktx:$fragment_version"
42      //noinspection GradleDependency
43      implementation 'androidx.appcompat:appcompat:1.4.1'
44      //noinspection GradleDependency
45      implementation 'com.google.android.material:material:1.6.0'
46      implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
47      implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
48      implementation project(path: ':openCVLibrary310')
49      implementation 'com.android.support.constraint:constraint-layout:2.0.4'
```

Parte de código 2. Implementación de *MpCharts* en la línea 47

2. Configuración de las dependencias de compilación en *settings.gradle*.

```
settings.gradle (TrabajoFinalGrado)
1   pluginManagement { PluginManagementSpec it ->
2       repositories { RepositoryHandler it ->
3           gradlePluginPortal()
4           google()
5           mavenCentral()
6       }
7   }
8   dependencyResolutionManagement { DependencyResolutionManagement it ->
9       repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
10      repositories { RepositoryHandler it ->
11          google()
12          mavenCentral()
13          //noinspection JcenterRepositoryObsolete
14          jcenter()
15          maven { url 'https://jitpack.io' }
16      }
17  }
18  rootProject.name = "TrabajoFinalGrado"
19  include ':app'
20  include ':openCVLibrary310'
```

Parte de código 3. *Maven Setup*

[8] Tracking: Sistema de rastreo, en este caso, orientado a objetos.

Una vez instalada la librería se leyó la documentación oficial para entender su funcionamiento y poder representar gráficas con las coordenadas obtenidas.

Esta librería permite la personalización de las gráficas en función de cuales vayan a ser los valores representados. Estos son algunos de los ejemplos que ofrece de manera prediseñada:

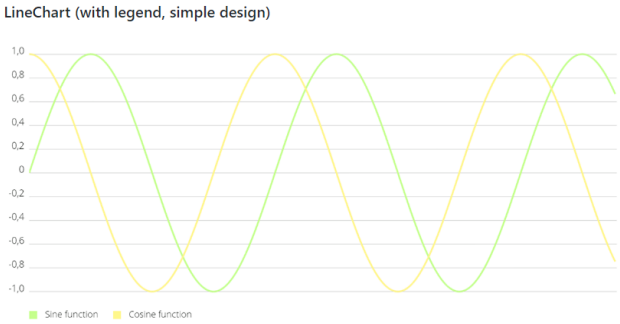


Ilustración 17. Ejemplo 1 de gráfica *MpCharts.*

Fuente: https://github.com/PhilJay/MPAndroidChart

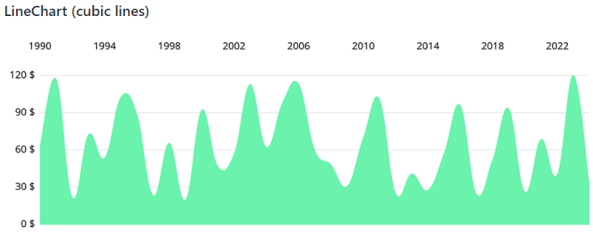


Ilustración 18. Ejemplo 2 de gráfica *MpCharts.*

Fuente: https://github.com/PhilJay/MPAndroidChart

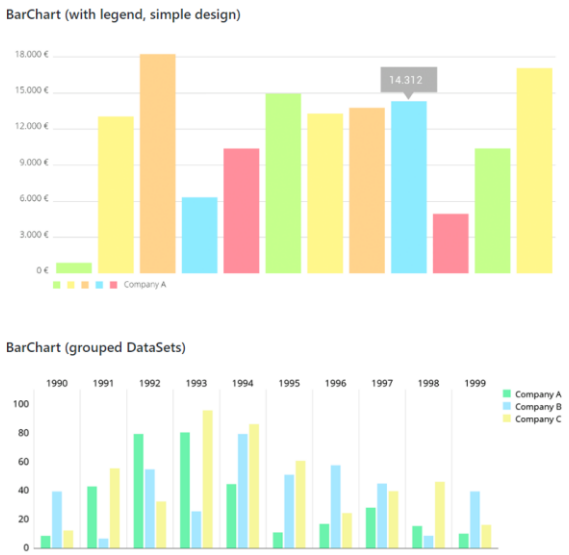


Ilustración 19. Ejemplo 3 de gráfica *MpCharts.*

Fuente: https://github.com/PhilJay/MPAndroidChart

Capítulo 4. Entorno virtual

En este apartado se va a describir el entorno en el que se ha trabajado, así como sus herramientas y el lenguaje escogido para el desarrollo de la aplicación.

4.1 Estructura de proyecto

La creación de una aplicación requiere crear una estructura con los pasos a seguir, mantener un proyecto organizado y declarar todos los permisos necesarios para asegurar un correcto funcionamiento.

Se van a describir todos los tipos de ficheros que existen dentro de Android Studio y para qué sirven cada uno de ellos. Aunque parezca tedioso tener que lidiar con tanto tipo de fichero distinto, a la larga esta clasificación de archivos permite tener un proyecto mejor ordenado y poder detectar errores de manera más efectiva.

4.1.1 Android

Android es un sistema operativo móvil basado en el núcleo Linux y programas de código abierto. Actualmente, según diversos estudios, se puede afirmar que Android es el sistema operativo más usado en el mundo, seguido de iOS [20].

Sistema Operativo	Porcentaje
Android	71,42 %
iOS	27,95 %
Samsung	0,2 %
KaiOS	0,14%
Desconocido	0,12 %
Windows	0,06 %

Ilustración 20. Sistemas operativos móviles más usados en 2020.

Fuente: https://itsoftware.com.co/content/sistemas-operativos-mas-usados/

Inicialmente fue desarrollado por Android Inc. hasta que fue adquirido por Google en 2005, dándole el reconocimiento que actualmente conocemos.

Este sistema operativo se expande a más dispositivos como smartphones, televisiones, tabletas, automóviles, etc. Esto permite la implementación de aplicaciones para todo tipo de usuario y para su uso en cualquier situación, haciendo que se puedan crear aplicaciones multiplataforma.

Ilustración 21. Icono de Android.

Fuente: https://es.wikipedia.org/wiki/Android

4.1.2 Activity

La clase *Activity* es un componente clave de una aplicación, y la forma en la que se inician y se crean las actividades es la marca principal de Android Studio. A diferencia de otros entornos, una vez se crea un nuevo proyecto, se inicia un método *main()* a partir del cual ya se pueden añadir las líneas de programación.

Al crear un proyecto se ofrece la posibilidad de poder arrancarlo con estructuras ya predefinidas o bien desde cero:

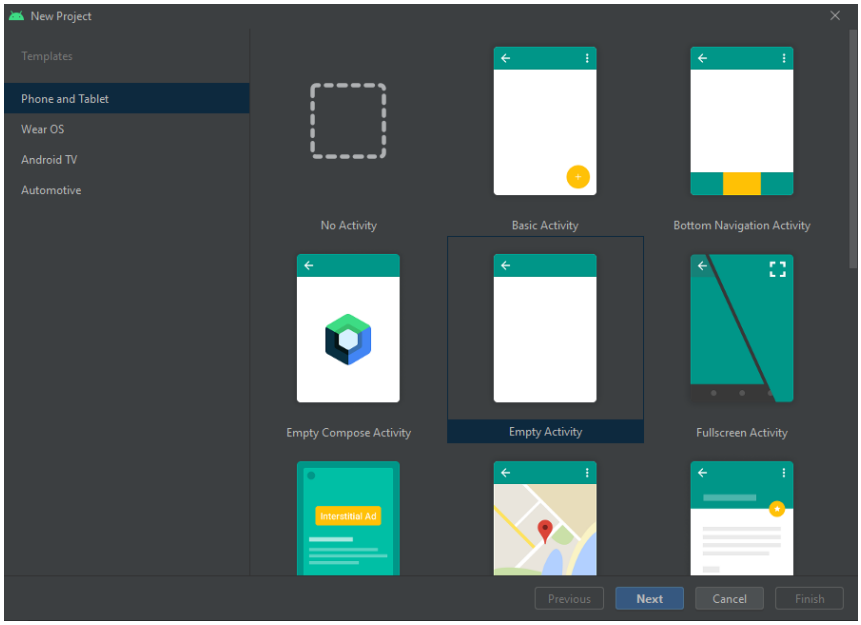


Ilustración 22. Pantalla de creación de proyecto

La clase *Activity* está diseñada para facilitar la experiencia del usuario. Cada *activity* corresponde a una pantalla de una aplicación de manera que todas las pantallas que la conforman han sido diseñadas de manera individual, gráfica y escrita.

Para la navegación entre estas pantallas se ha descrito en el contenido de cada *activity* cómo se ha de cambiar entre una actividad u otra según la interacción del usuario. Esto permite diseñar cada *activity* con un funcionamiento concreto y no mezclar códigos, diseñando un proyecto ordenado y con la cantidad de *activities* necesarias sin tener límitaciones.

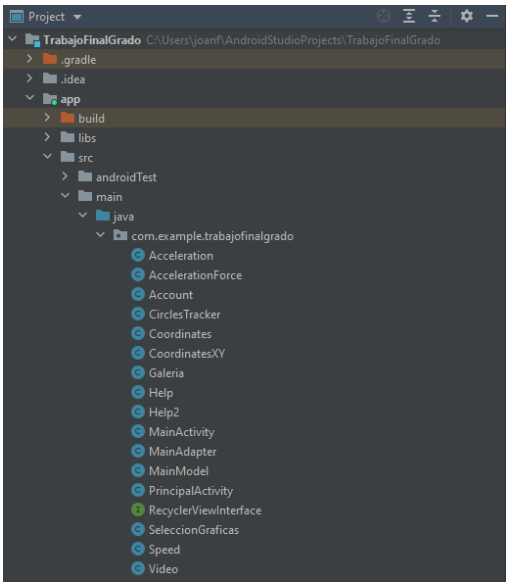


Ilustración 23. *Activities* creadas para la aplicación

El uso de métodos y clases conforman la parte lógica de las *activities*. En éstas debe quedar registrado todo el contenido necesario para que la aplicación funcione de manera correcta y no se cuelgue en ningún momento [21].

Además, en cada actividad se diseña una secuencia de métodos de devolución de la llamada que la da por iniciada, así como métodos de devolución de la llamada que la eliminan, creando su ciclo de vida:

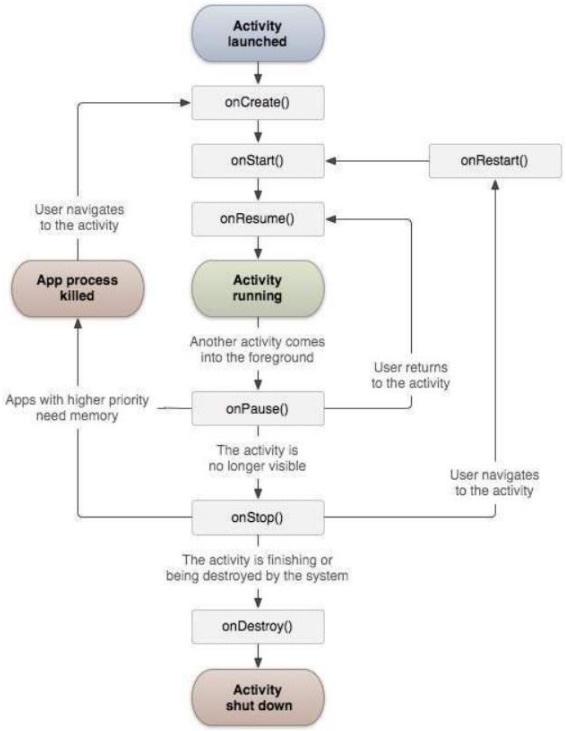


Ilustración 24. Ciclo de vida de una actividad.

Fuente: https://developer.android.com/reference/android/app/Activity

4.1.3 Layouts

Un *layout* (diseño) define la estructura de una interfaz de usuario en la aplicación, como pueden ser las *Activity*. Todos los elementos del *layout* se basan en una jerarquía de objetos *View* o *ViewGroup* con los que se diseña el aspecto gráfico e interactivo de cada pantalla.

Estos objetos se llaman widgets y poseen muchas subclases. El uso de estos objetos viene estructurado por los denominados *ViewGroup*, que pueden ser:

- *ConstraintLayout:* Permite la creación de diseños complejos con una jerarquía de vistas plana, sin *Views* anidados.

- *RelativeLayout:* Permite la muestra de *Views* secundarias en posiciones relativas. La posición de cada View se específica cómo relativa a los demás elementos del mismo nivel.

- *LinearLayout*: Permite una introducción de widgets más sencilla. Se pueden organizar de forma horizontal o vertical, pudiendo anidar los *Views* de forma ordenada y rápida.

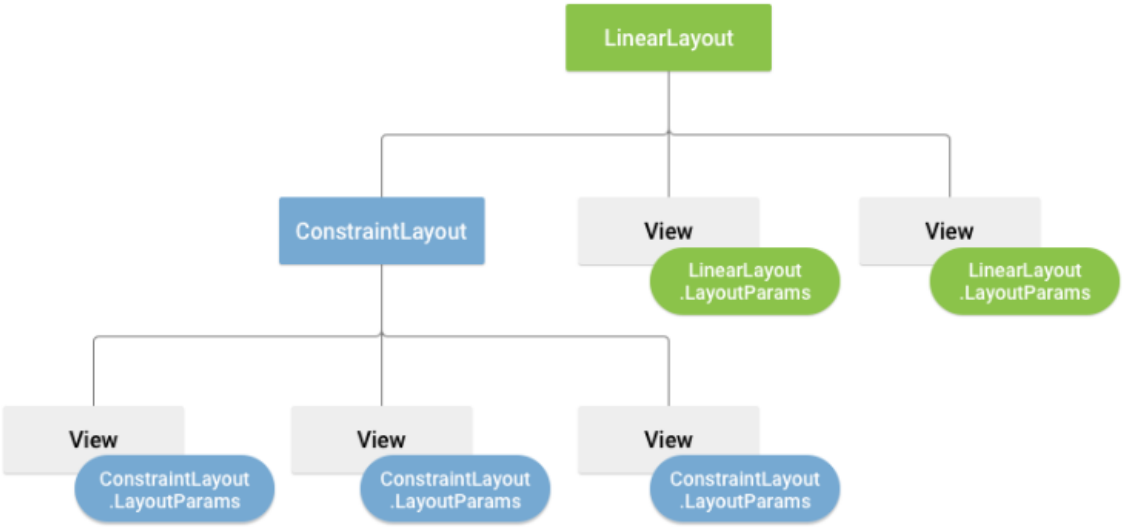


Ilustración 25. Ejemplo de jerarquía de diseño.

Fuente: https://developer.android.com/reference/android/app/Activity

Los *layouts* se almacenan en ficheros XML, este tipo de fichero permite la creación de diseños de IU y de los elementos que la *activity* asociada contiene.

Cuando la aplicación se compila, cada archivo XML se compila en un recurso *View*. Para cargar estos ficheros se deben implementar en el método *OnCreate()* de cada *activity*, de manera que al acceder a dicha pantalla se fusione la parte lógica con la gráfica mostrando el resultado al usuario.

Kotlin Java

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Parte de código 4. Declaración en método *OnCreate()*.

Fuente: https://developer.android.com/reference/android/app/Activity

Para asociar cada objeto *View* o *ViewGroup* a su parte lógica descrita en la *activity* se deben identificar de una forma única, de manera que, al arrancar la aplicación, se haga referencia a esta ID y se muestre correctamente [22].

4.1.4 Intent

Un *intent* es un objeto que se usa para solicitar una acción a otro componente de la aplicación [23]. Existen tres casos principales en los que se suele usar este tipo de objeto:

- Iniciar una actividad: cada *Activity* es una pantalla de la aplicación, y es en este caso cuando se usa un *intent* para navegar entre ellas mediante el uso de *startActivity()* o *startActivityForResult()*. Este es el *intent* más utilizado en la aplicación desarrollada.

- Iniciar un servicio: un *service* es un componente que actúa en segundo plano sin una interfaz diseñada por el usuario. Para iniciar un servicio se hace mediante el uso de *startService()*.

- Transmitir una emisión: una emisión es un aviso que puede recibir cualquier aplicación. Para iniciar una emisión se usa *sendBroadcast()* o *sendOrderedBroadcast()*.

Kotlin Java

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Parte de código 5. Uso de un *intent.*

Fuente: https://developer.android.com/guide/components/intents-filters?hl=es-419

Además, existen dos tipos de *intents*:

- *Intents* explícitas: especifican que aplicación los administrará, incluyendo el nombre del paquete de la aplicación o el nombre de la clase de la propia aplicación desde donde se ha llamado. Este es el tipo de *intent* que se usa para cambiar entre actividades a petición del usuario.

- *Intents* implícitas: no se especifica el componente de la aplicación, pero se declara una acción general a realizar por parte de un componente de otra aplicación. Este es el tipo de *intent* que se usa para solicitar que otra aplicación muestre una ubicación específica.

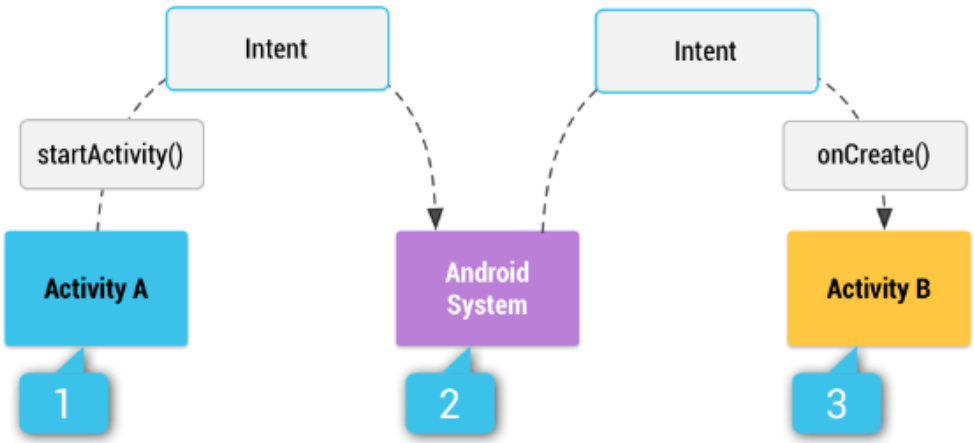


Ilustración 26. Entrega de una *intent* implícita.

Fuente: https://developer.android.com/guide/components/intents-filters?hl=es-419

4.1.5 Manifest

En todos los proyectos de aplicaciones debe existir un fichero AndroidManifest.xml en la raíz de la fuente del proyecto. Este archivo describe información esencial de la aplicación para las herramientas de creación del propio Android [24].

El archivo de manifiesto debe declarar lo siguiente:

- El nombre del paquete de la aplicación que se está diseñando. Las herramientas de compilación usan esta información para ubicar las entidades de código al compilar el proyecto.

- Los componentes de la aplicación, entre ellas todas las actividades, servicios, receptores de emisiones y proveedores de contenido. Todos los componentes deben definir propiedades básicas como su nombre.

- Los permisos necesarios para el correcto funcionamiento de la aplicación. Estos son necesarios para acceder a las partes protegidas del sistema u otras aplicaciones.

- Las funciones de hardware y software que requiere la aplicación.

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

    <!-- Beware that these values are overridden by the build.gradle file -->
    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="26" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <!-- This name is resolved to com.example.myapp.MainActivity
             based upon the package attribute -->
        <activity 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=".DisplayMessageActivity"
            android:parentActivityName=".MainActivity" />
    </application>
</manifest>
```

Parte de código 6. Ejemplo de AndroidManifest.xml.

Fuente: https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419

El fichero de manifiesto se generará de manera automática en la creación del proyecto.

Los elementos esenciales que se vayan creando se irán añadiendo automáticamente en dicho fichero a medida que se vaya compilando la aplicación.

En nuestro caso, aparte de los componentes básicos como serían las clases, se ha necesitado pedir los permisos para poder usar la cámara integrada del teléfono móvil, así como los permisos para poder acceder y escribir ficheros internos del dispositivo móvil.

4.1.6 Gradle

El sistema de compilación de Android compila los recursos y código fuente que empaqueta en APK[9] usando *Gradle* [25].

[9] APK: Android Application Package. Archivos que contienen los datos necesarios para instalar aplicaciones.

El *gradle* es un paquete de herramientas de compilación avanzadas para automatizar dicho proceso y definir sus configuraciones.

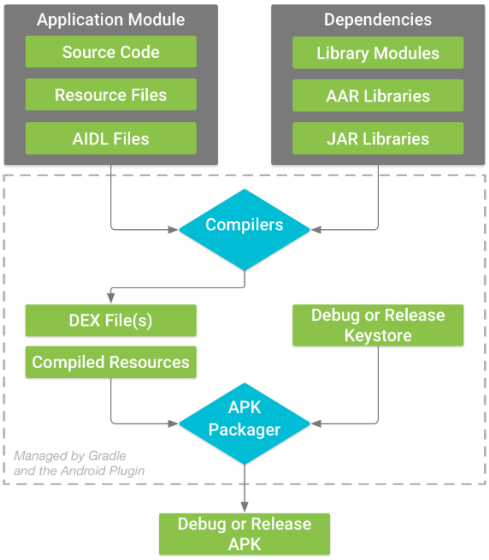


Ilustración 27. Proceso de compilación de un módulo de aplicación.

Fuente: https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419

El proceso de compilación sigue los siguientes pasos:

- El compilador convierte el código fuente en archivos DEX[10] que incluyen el código de bytes que se ejecuta en los dispositivos Android y los recursos compilados.
- El empaquetador combina los archivos DEX y los recursos compilados en un APK.
- El empaquetador firma la APK usando claves de depuración.
- Finalmente, el empaquetador usa la herramienta *zipalign*[11] para optimizar la aplicación usando la menor memoria posible.

Groovy Kotlin

```
plugins {

    /**
     * You should use `apply false` in the top-level build.gradle file
     * to add a Gradle plugin as a build dependency, but not apply it to the
     * current (root) project. You should not use `apply false` in sub-projects.
     * For more information, see
     * Applying external plugins with same version to subprojects.
     */

    id 'com.android.application' version '7.2' apply false
    id 'com.android.library' version '7.2' apply false
    id 'org.jetbrains.kotlin.android' version '1.6.10' apply false
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Parte de código 7. Ejemplo de archivo de *Gradle*.

Fuente: https://developer.android.com/studio/build?hl=es-419

[10] DEX: Dalvik Executable Format. Archivos que contienen datos para el arranque de la aplicación.
[11] Zipalign: herramienta para la alineación de archivos ZIP.

Capítulo 5. Aplicación Android: "God Analysis"

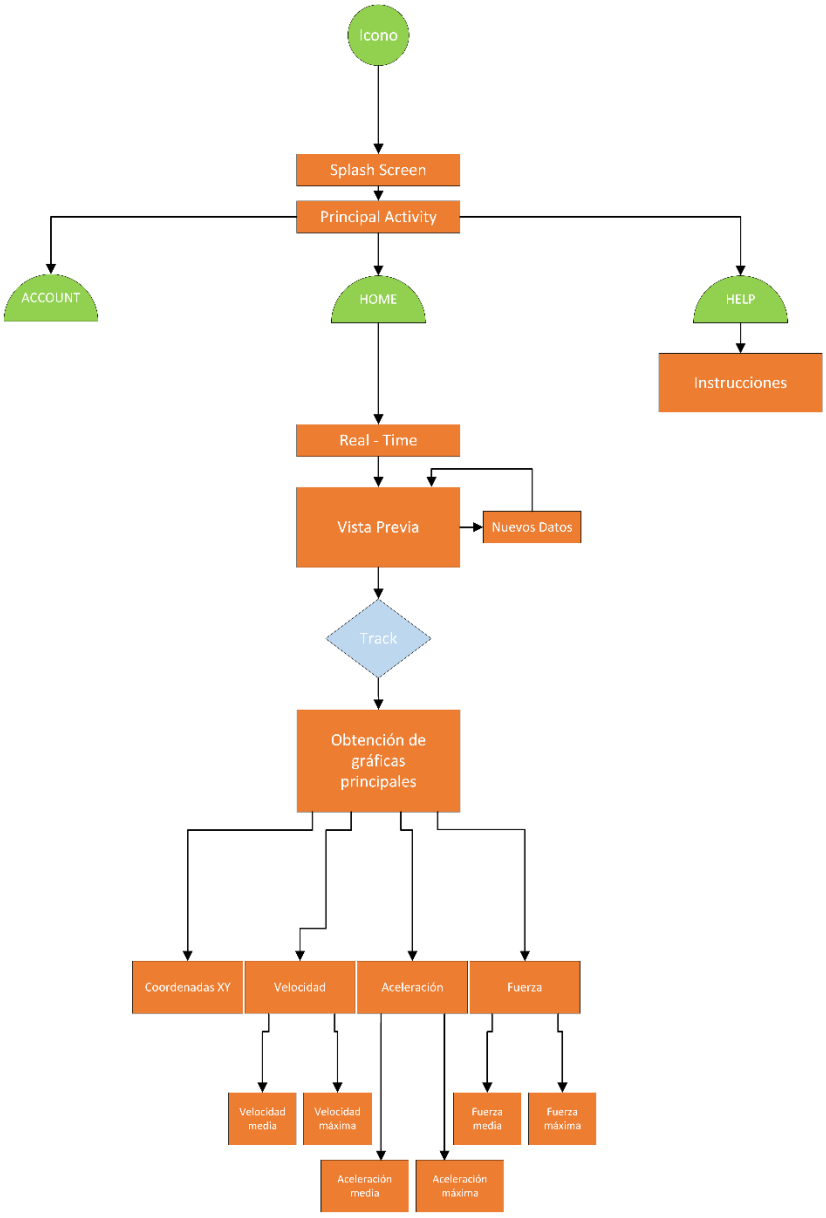
En este apartado se va a detallar paso a paso cómo se ha diseñado la parte lógica y gráfica de cada actividad.

5.1 Descripción

En este trabajo final de grado se ha desarrollado una aplicación llamada *God Analysis* orientada a personas que realizan ejercicios con pesas durante sus entrenamientos

Esta aplicación realiza un seguimiento de los discos durante el ejercicio aportando gráficas tras su finalización. Estas aportan al usuario material visual con el que se podrá analizar si el movimiento realizado en el ejercicio es el correcto, así como la velocidad, aceleración y fuerza con la que se ha realizado cada repetición.

5.2 Flujograma general de la aplicación



Flujograma 1. General

5.3 Inicio de la aplicación

5.3.1 Splash Screen

La primera pantalla con la que arranca la aplicación es la denominada *Splash Screen*.

Esta actividad se ha diseñado con la finalidad de proporcionar a la aplicación un toque más profesional ya que de esta forma no se inicia directamente en el menú principal, sino que se muestra una pantalla con el diseño del título.

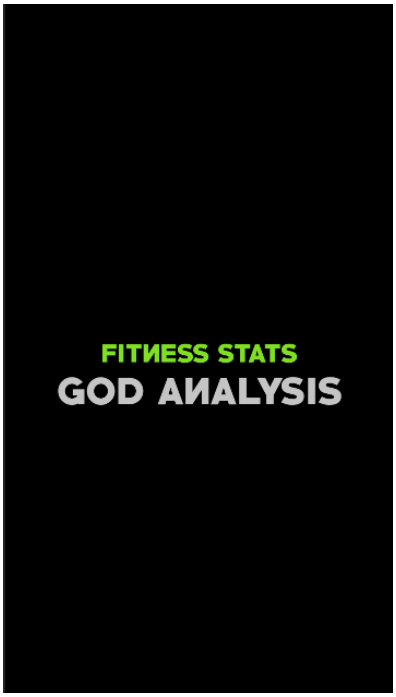


Ilustración 28. Diseño del *Splash Screen*

La parte lógica de esta actividad es muy simple. Solo existen dos métodos:

1. Método *OnCreate()*: se utiliza para mostrar el diseño descrito en el archivo *xml* y para crear un retardo de tres segundos, que será el tiempo que durará la muestra de esta pantalla, antes de que el *Intent* nos traslade al menú principal.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);


    TimerTask tarea = () -> {
            Intent intent = new Intent( packageContext: MainActivity.this, PrincipalActivity.class);
            startActivity(intent);
            finish();

    };

    Timer tiempo = new Timer();
    tiempo.schedule(tarea,  delay: 3000); //3 segundos

    }
```

Ilustración 29. Método *OnCreate()* de *Splash Screen*

2. Método para Pantalla Completa: para evitar que el menú de navegación aparezca en esta primera pantalla se ha descrito un conjunto de dos métodos que muestran la pantalla completa. Este código es de uso libre y se puede conseguir en la página web oficial para desarrolladores de Android.

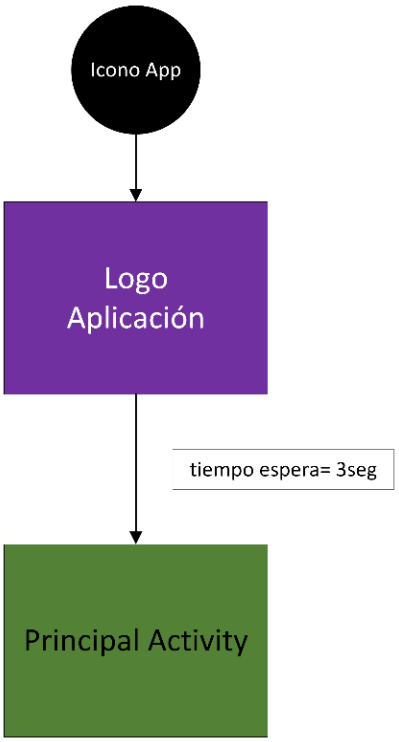
```
//PANTALLA COMPLETA
public void onWindowFocusChanged(boolean hasFocus) {
    super.onWindowFocusChanged(hasFocus);
    if (hasFocus) {
        hideSystemUI();
    }
}

private void hideSystemUI() {
    // Enables regular immersive mode.
    // For "lean back" mode, remove SYSTEM_UI_FLAG_IMMERSIVE.
    // Or for "sticky immersive," replace it with SYSTEM_UI_FLAG_IMMERSIVE_STICKY
    View decorView = getWindow().getDecorView();
    decorView.setSystemUiVisibility(
            View.SYSTEM_UI_FLAG_IMMERSIVE
                    // Set the content to appear under the system bars so that the
                    // content doesn't resize when the system bars hide and show.
                    | View.SYSTEM_UI_FLAG_LAYOUT_STABLE
                    | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
                    | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
                    // Hide the nav bar and status bar
                    | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION
                    | View.SYSTEM_UI_FLAG_FULLSCREEN);
}
```

Parte de código 8. Métodos para pantalla completa

Con la destrucción de esta actividad tras tres segundos se da paso al menú principal de la aplicación a través del *Intent* que se muestra en el método *OnCreate()*.

5.3.2 Flujograma Splash Screen



Flujograma 2. Splash Screen

5.4 Menú principal

5.4.1 Principal Activity

Esta actividad corresponde al menú principal de la aplicación desde donde se puede acceder al modo de *trackeo* y a las instrucciones.

Es en esta pantalla donde se han de introducir los parámetros necesarios para informar a la aplicación que peso estamos levantando y las características del disco usado para su correcto *trackeo*.

La interfaz gráfica diseñada para esta actividad es la siguiente:

Ilustración 30. Interfaz gráfica de *PrincipalActivity*

En la parte central de la pantalla se han diseñado dos campos obligatorios de introducción de datos, el peso y el diámetro del disco. Estos dos objetos *View* llamados *EditText* [26] se encargan de almacenar los valores introducidos que se irán enviando entre actividades para su uso.

Los *EditText* se han personalizado para que solo se puedan escribir números y no provocar errores.

Para comprobar que los valores se han introducido de manera correcta se han añadido dos botones para guardar los valores introducidos. En caso de que los datos no se hayan almacenado de forma correcta, la aplicación lanzará un mensaje de alerta informando al usuario de que el dato introducido no es correcto.

```
if(editTextValue.length()==0){
    AlertDialog.Builder builder = new AlertDialog.Builder( context: PrincipalActivity.this);
    builder.setMessage("¡Alerta! No ha introducido los valores necesarios correctamente.");
    builder.show();
    introducirPeso=false;

}
```

Parte de código 9. Código de la alerta introducción de datos

Una vez se han introducido los datos de manera correcta se puede pasar al *tracking*.

El botón que haya sido pulsado llamará a un *Intent* que enviara al usuario a la actividad del *tracking* en *Real-Time* desde donde se accederá a una nueva interfaz que nos indicará como debemos proseguir.

```
Button btn = (Button) findViewById(R.id.botoninicio);
btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {


        if(!introducirPeso || !introducirDiametro){
            AlertDialog.Builder builder = new AlertDialog.Builder( context: PrincipalActivity.this);
            builder.setMessage("¡Alerta! No ha introducido los valores necesarios correctamente.");
            builder.show();


        }else{
            Intent intent = new Intent (v.getContext(), CirclesTracker.class);
            intent.putExtra( name: "DiametroCm", editTextValue1);
            intent.putExtra( name: "PesoKg", editTextValue);

            startActivityForResult(intent,  requestCode: 0);

        }
    }
});
```

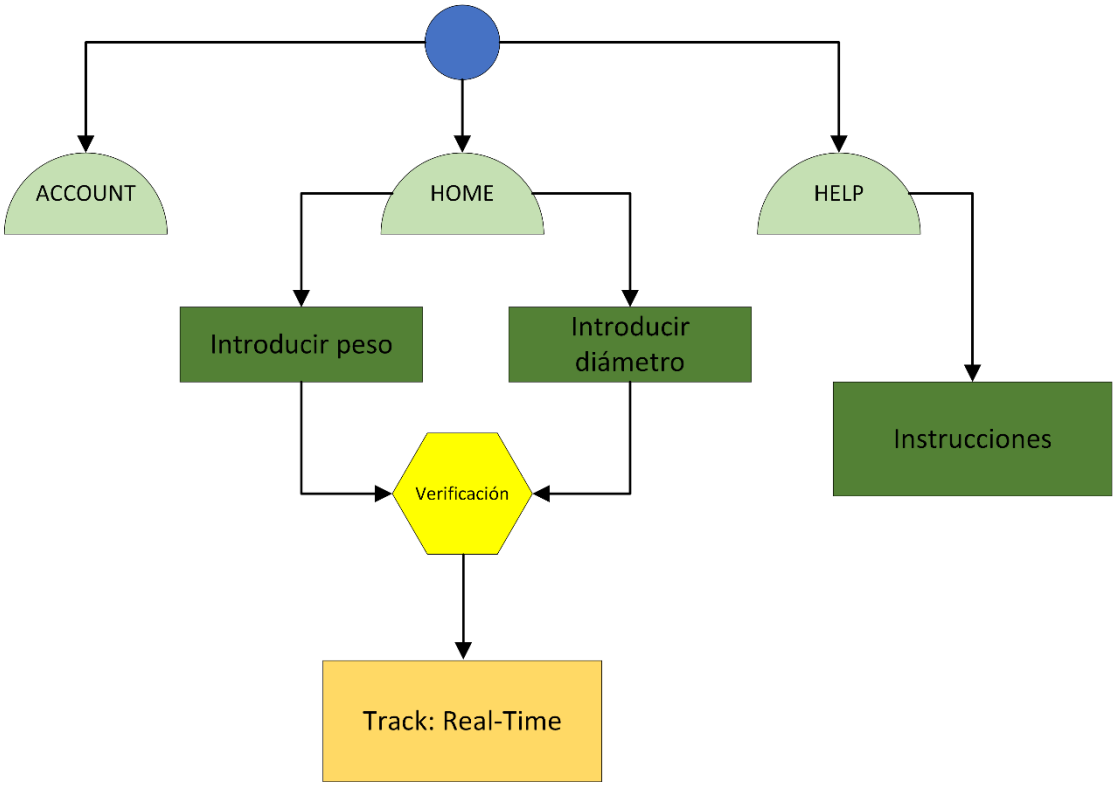
Parte de código 10. Acceso al modo Real-Time

Por último, se ha diseñado un menú de navegación extra que nos permite acceder a dos actividades más, el *Help* y el *Account*.

La actividad *Help* se ha diseñado para ayudar al usuario a conocer el funcionamiento de la aplicación en caso de no entenderlo en algún punto en concreto de la misma, mientras que la actividad Account ha sido diseñada con la intención de poder llevar un control de los ejercicios realizados durante la sesión de entrenamiento.

Ambas actividades se explican con detenimiento más adelante en la memoria.

5.4.2 Flujograma Principal Activity



Flujograma 3. Principal Activity

5.5 Track: Real Time

Este proceso es la piedra angular de la aplicación. La actividad en la que se ha escrito el código para el tracking en tiempo real del objeto se llama *CirclesTracker* y de ella va a salir toda la información necesaria para la representación de las gráficas.

En este primer bloque de la aplicación se detectan los objetos redondos y se *trackean*. Este procedimiento conlleva un procesado de *frames* en tiempo real mediante el uso de algoritmos de *OpenCV* para el uso de la cámara y el *tracking* del objeto.

5.5.1 CirclesTracker

El primer paso que se planteó fue obtener la vista previa de la cámara en la que se pudieran procesar los *frames*. Esto se consiguió mediante algoritmos de *OpenCV* llamados *CameraBridgeViewBase* y *JavaCameraView*.

El teléfono móvil requiere los permisos necesarios para poder acceder a la cámara. Estos han sido declarados en el fichero *Manifest* [27]:

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
<uses-feature android:name="android.hardware.camera.front" />
<uses-feature android:name="android.hardware.camera.front.autofocus" />
```

Parte de código 11. Permisos para el acceso a la cámara

Al arrancar la actividad el usuario se encuentra con un mensaje de información en el que se le da la bienvenida y se le dan unas pequeñas indicaciones de cómo realizar correctamente el

tracking de la pesa. Se pretende informar al usuario de cómo colocar el dispositivo móvil para mejorar la efectividad del seguimiento.

Estas indicaciones son simples y lógicas. Se le pide al usuario que coloque el móvil en un trípode o sitio sobre el que se mantenga quieto y recto. Además, se aconseja que se sitúe de manera que quede totalmente libre de objetos a su alrededor y a una distancia desde donde se detecte un círculo completo.

La interfaz gráfica se ha diseñado con una serie de botones que ofrecen funciones con las que el usuario puede mejorar su experiencia de uso.

Ilustración 31. Interfaz gráfica de *CirclesTracker*

Las funciones que realizan los botones son las siguientes:

1. Cambiar de cámara: de manera predeterminada la vista previa que se mostrará será la de la cámara trasera, pero se le da la oportunidad al usuario de cambiar a la cámara frontal en caso de que esa sea su decisión.

   Para ello se ha escrito el siguiente código:

```
flip_camera = findViewById(R.id.flip_camera);
flip_camera.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        swapCamera();
        cameraSwap = !cameraSwap;
    }
});
```

```
private void swapCamera() {
    mCameraId=mCameraId^1;
    cameraBridgeViewBase.disableView();
    cameraBridgeViewBase.setCameraIndex(mCameraId);
    cameraBridgeViewBase.enableView();
}
```

Parte de código 12. Cambiar de cámara

2. Introducir nuevos datos: en caso de que haya terminado el ejercicio y se quiera cambiar el disco utilizado, se ofrece la posibilidad de cambiar los valores de la pesa sin tener que retroceder al menú principal.

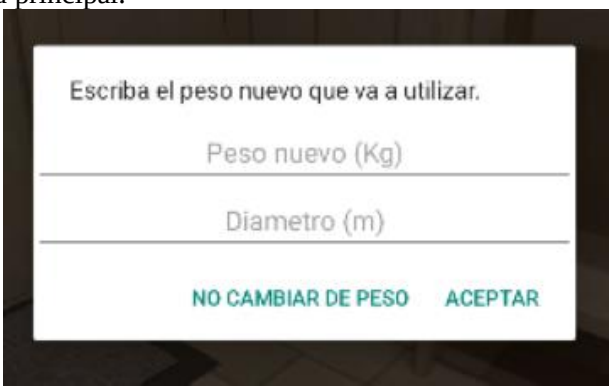


Ilustración 32. Ventana para introducir nuevos datos

Para ello se ha escrito el siguiente código:

```
pesoCamera = findViewById(R.id.peso);
pesoCamera.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        AlertDialog.Builder builder = new AlertDialog.Builder( context: CirclesTracker.this);
        builder.setCancelable(false);
        EditText editText = new EditText( context: CirclesTracker.this);
        builder.setMessage("Escriba el peso nuevo que va a utilizar.");
        editText.setHint("Peso nuevo (Kg)");
        editText.setInputType(InputType.TYPE_CLASS_NUMBER | InputType.TYPE_NUMBER_FLAG_DECIMAL);
        editText.setGravity(Gravity.CENTER);
        builder.setView(editText);

        EditText editText1 = new EditText( context: CirclesTracker.this);
        editText1.setHint("Diametro (m)");
        editText1.setInputType(InputType.TYPE_CLASS_NUMBER | InputType.TYPE_NUMBER_FLAG_DECIMAL);
        editText1.setGravity(Gravity.CENTER);
        builder.setView(editText1);

        LinearLayout linearLayout = new LinearLayout( context: CirclesTracker.this);
        linearLayout.setOrientation(LinearLayout.VERTICAL);
        linearLayout.addView(editText);
        linearLayout.addView(editText1);
        builder.setView(linearLayout);

        builder.setPositiveButton( text: "Aceptar", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialogo1, int id) {
                pesoKg = editText.getText().toString();
                diametroCm=editText1.getText().toString();
            }
        });
        builder.setNegativeButton( text: "No cambiar de peso", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialogo1, int id) {
            }
        });
        builder.show();

    }
});
```

Parte de código 13. Cambiar valores de la pesa

3. Empezar tracking: un círculo definido en la actividad *CirclesTracker* seguirá al objeto detectado al terminar la cuenta atrás. Se activará mediante el uso de un booleano definido en el método *OnCameraFrame()*.

   Para ello se ha descrito el siguiente código:

```
trackCamera = findViewById(R.id.trackCamera);
trackCamera.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        if (!startTrack) {
            new CountDownTimer( millisInFuture: 5000,  countDownInterval: 1000) {
                @Override
                public void onTick(long l) {
                    counter.setText(String.valueOf(contador));
                    contador++;
                }
                @Override
                public void onFinish() {
                    counter.setVisibility(View.GONE);
                    startTrack = true;
                    Toast.makeText(getApplicationContext(),  text: "cOUNTER END", Toast.LENGTH_LONG).show();
                }
            }.start();
        } else {
            startTrack = false;
        }

    }
});
```

Parte de código 14. Inicio de *tracking*

4. Obtener coordenadas: tras obtener una lista de coordenadas se permite al usuario acceder a la actividad *SelecciónGráficas* al pulsar el botón. En caso de no haber iniciado el proceso de tracking saltará un mensaje de alerta informando al usuario de que aún no se han obtenido las coordenadas.

   Para ello se ha descrito el siguiente código:

```
coordinatesCamera = findViewById(R.id.coordinatesButton);
coordinatesCamera.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

            if(coordenadax.isEmpty() && coordenaday.isEmpty()) {
                AlertDialog.Builder builder = new AlertDialog.Builder( context: CirclesTracker.this);
                builder.setMessage("¡Empiece a trackear! No ha iniciado el proceso correctamente.");
                builder.show();
            } else{
                Intent intent3 = new Intent(v.getContext(), SeleccionGraficas.class);
                //startActivityForResult(intent3, 0);
                intent3.putExtra( name: "Coordenadas Y", coordenaday.toString());
                intent3.putExtra( name: "Coordenadas X", coordenadax.toString());
                intent3.putExtra( name: "PesoKg", pesoKg);
                intent3.putExtra( name: "Diametro", diametroCm);
                startActivityForResult(intent3,  requestCode: 0);
            }
    }
});
```

Parte de código 15. Acceso a *SelecciónGráficas*

5. Temporizador: crear una cuenta atrás es indispensable para que al usuario le dé tiempo a pulsar el botón del track y llegar a levantar las pesas a tiempo para evitar cruzarse en el campo visual de la cámara. De esta manera se evita la detección de círculos falsos que estropeen las medidas recogidas.

Para ello se ha descrito el siguiente código:

```
counter = findViewById(R.id.counter);
trackCamera = findViewById(R.id.trackCamera);
contador=5;
contador1=5;
trackCamera.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        if (!startTrack) {
            new CountDownTimer( millisInFuture: contador*1000,  countDownInterval: 1000) {
                @Override
                public void onTick(long l) {
                    counter.setVisibility(View.VISIBLE);
                    counter.setText(String.valueOf(contador--));
                }
                @Override
                public void onFinish() {
                    counter.setVisibility(View.GONE);
                    startTrack = true;
                    coordenaday.clear();
                    coordenadax.clear();
                }
            }.start();
        } else {
            startTrack = false;
            if(contador==0){
                contador=contador1;
            }
        }
    }
});
```

Parte de código 16. Creación del temporizador

Procedentes de la actividad anterior se reciben los valores del peso y diámetro con los que se guiará al detector de objetos a estimar unas coordenadas reales en el movimiento.

Una vez el usuario escoge la cámara y ha introducido los valores del disco que esté utilizando, la aplicación está lista para dar inicio al proceso del *trackeo*.

Para arrancar la vista previa de la cámara se implementa la clase *CameraBridgeViewBase*.

CameraBridgeViewBase es una interfaz de clase que captura *frame* a *frame* la vista previa de la cámara y lo devuelve al método *OnCameraFrame()* para procesarlos mediante el uso de matrices.

El código se ha programado para que en el momento en que el usuario quiera *trackear* el objeto, pulse un botón y active el proceso de almacenar los *frames* en el método *OnCameraFrame()*, en donde se ha descrito el código para la detección de objetos.

El código empleado para implementar el *CameraBridgeViewBase* a la actividad es el siguiente:

```
cameraBridgeViewBase = (JavaCameraView) findViewById(R.id.cameraview); //Set up de camera bridge
cameraBridgeViewBase.setVisibility(SurfaceView.VISIBLE);
cameraBridgeViewBase.setCvCameraViewListener(this);
cameraBridgeViewBase.setMaxFrameSize( maxWidth: 720,  maxHeight: 480); //A menor resolucion, mejor trackeo: 480p 16:9
baseLoaderCallback = new BaseLoaderCallback( AppContext: this) {
    //En estas lineas se comprueba que todo esta funcionando perfectamente
    @Override
    public void onManagerConnected(int status) {
        super.onManagerConnected(status);

        switch (status) {

            case BaseLoaderCallback.SUCCESS:
                cameraBridgeViewBase.enableView();
                break;
            default:
                super.onManagerConnected(status);
                break;
        }
    }
};
```

Parte de código 17. Configuración *CameraBridgeViewBase*

Al implementar esta clase en la actividad se crean automáticamente nuevos métodos que controlan el funcionamiento y ciclo de vida de la vista previa de la cámara para evitar problemas en su uso. Esos métodos son los siguientes:

1. Método *onResume()*.
2. Método *onPause()*.
3. Método *onDestroy()*.
4. Método *onCameraViewStarted()*.
5. Método *onCameraViewStopped()*.

El método que lleva el peso de esta actividad es el *onCameraFrame()*, donde se recibirán los *frames* por parte del *CameraBridgeViewBase* y se procesarán.

Dentro de este método se han utilizado varios algoritmos de *OpenCV* para conseguir el *tracking*, los cuales se van a explicar por su orden de funcionamiento:

1. Botón *startTrack*: al pulsar este botón se inicia el proceso del tracking y se procesará cada *frame* obtenido.

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

    if(cameraSwap==true){
        if (startTrack == true) {
```

Parte de código 18. Inicio del método *onCameraFrame*

2. Creación de matrices: se crean dos matrices.
   La primera de ellas, llamada input, es la matriz que va a almacenar los *frames* recibidos del *CameraBridgeViewBase* y los convierte a escala de grises para procesarlos con mayor facilidad.
   La segunda matriz se llama *circles* y se encarga de recibir la información de las coordenadas del objeto y de dibujar sobre él un círculo.

```
Mat input = inputFrame.gray();
Mat circles = new Mat();
```

Parte de código 19. Creación de matrices

3. *Blur*: el primer efecto que se aplica sobre la matriz que contiene los *frames* es un *blur*. Este procesado de *frames* suavizará la imagen de manera que se facilita la detección de contornos para distinguir los objetos circulares [28].

```
Imgproc.blur(input, input, new Size( width: 7,  height: 7), new Point( x: 2,  y: 2));
```

Parte de código 20. Efecto *Blur* de *OpenCV*

4. Transformada de *Hough*: este efecto es utilizado para la detección de figuras en imágenes digitales, en nuestro caso, un círculo [29].
   Para definir la transformada de *Hough* de un círculo se debe expresar mediante la siguiente ecuación:

$$(x-a)^2+(y-b)^2=r^2$$

Fórmula 1. Transformada de *Hough* para círculos

   Se necesita un acumulador con tres dimensiones para (a,b) que son el centro y para (r) que es su radio.
   A partir de aquí, cada punto de la imagen vota por los círculos en los que podría estar. Una vez terminado este proceso con todos los puntos de la imagen se buscan los picos en el acumulador obteniendo el radio y el centro de la circunferencia.

   Estos valores los almacena el propio Android Studio facilitando su introducción en *arrays* o listas.

```
Imgproc.HoughCircles(input, circles, Imgproc.CV_HOUGH_GRADIENT,  dp: 1,  minDist: input.rows()/2,
        param1: 100, param2: 50, minRadius: 1, maxRadius: 0);
```

Parte de código 21. Transformada de *Hough* de *OpenCV*

5. Número de círculos: se ha creado un *if* con un *for* en su interior para establecer el número de círculos que se quiera detectar, en nuestro caso solo uno.

```
if (circles.cols() > 0) { //Si no detecta minimo uno no funciona

    validTrack = true;

    for (int x = 0; x < Math.min(circles.cols(), 1); x++) { //El 1 es la cantidad de circulos que detcta en pantalla

        double circleVec[] = circles.get(0, x);

        if (circleVec == null) {
            break;
        }
```

Parte de código 22. Establecer número de círculos detectados

6. Interfaz del *track*: para mostrar al usuario que el seguimiento se está realizando de manera correcta se dibujan dos círculos con la librería de *OpenCV* sobre el objeto, usando las coordenadas obtenidas de la transformada de *Hough*.

```
Imgproc.circle(input, center,  radius: 3, new Scalar(0, 100, 255),  thickness: 5); //punto del centro
Imgproc.circle(input, center, radius, new Scalar(0, 0, 255),  thickness: 2); //circulo exterior
```

Parte de código 23. Interfaz del *track*

7. Crear lista de coordenadas: mediante el uso de un *booleano* se permite al código ir añadiendo las coordenadas obtenidas en una lista hasta que se deje de *trackear*.
   Las coordenadas obtenidas se multiplican por menos uno debido a que se ha girado la vista previa desde el *CameraBridgeViewBase* para conseguir una vista previa correcta. Esto provoca que el móvil detecte el movimiento de manera invertida a la realidad.

```
Point center = new Point((int) circleVec[0], (int) circleVec[1]);
int radius = (int) circleVec[2];
double factor= (diametro_cm/2)/radius;

coordenaday.add(center.x*factor*(-1)); //milimetros
coordenadax.add(center.y*factor*(-1)); //milimetos
```

Parte de código 24. Guardar coordenadas en una lista

8. Mostrar resultado: en todo momento se muestran los valores de la matriz *input* en RGBA de manera que todo el proceso de tracking se realiza en un segundo plano mejorando la calidad visual de la aplicación.

```
circles.release();
input.release();
```

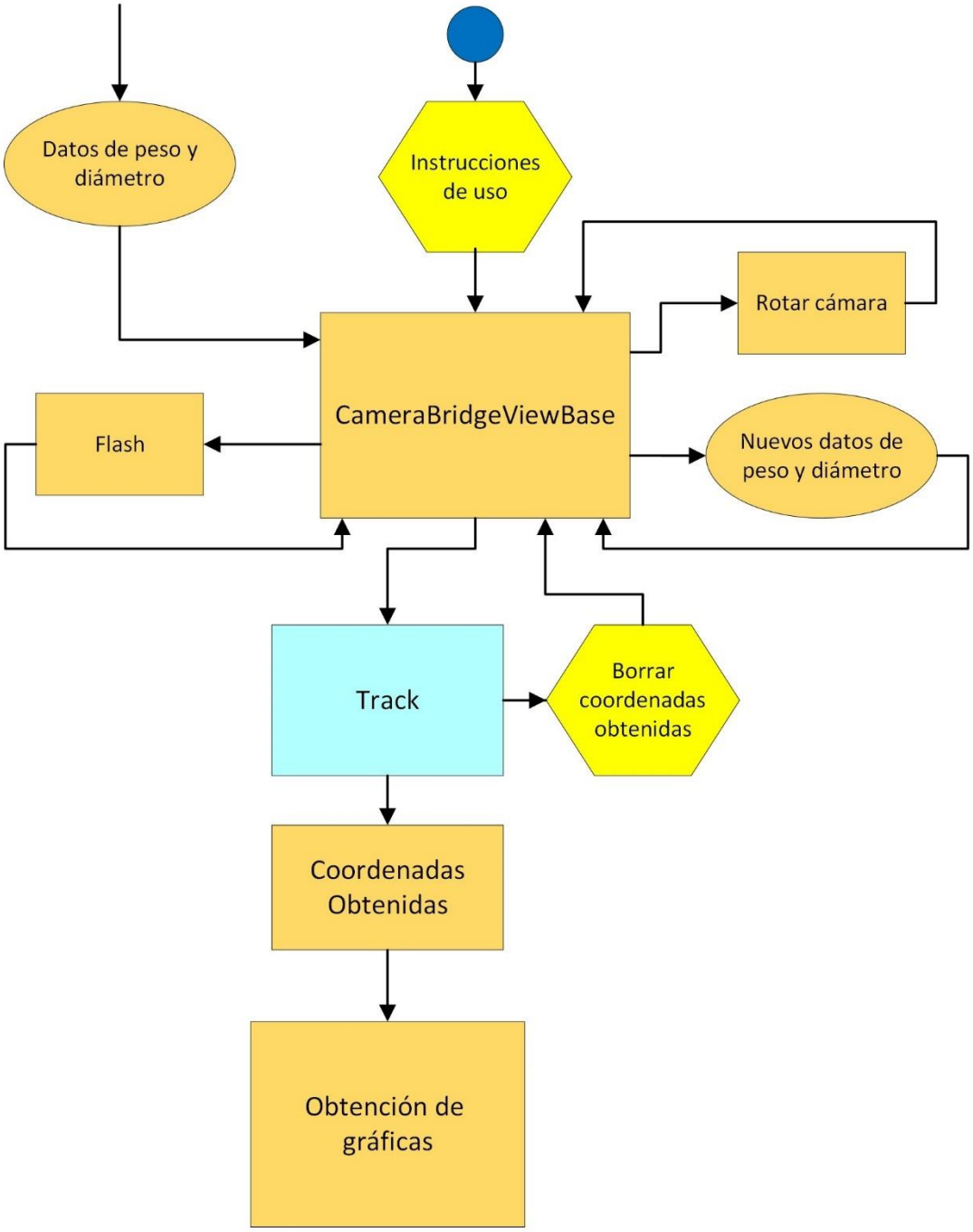
Parte de código 25. Mostrar resultados

Tras este proceso se obtiene una lista completa con coordenadas X e Y listas para enviarse a la siguiente actividad en el momento que el usuario lo decida. Para ello se dispone de un botón que permite el paso a la obtención de gráficas.

Aunque ya se haya obtenido la lista de coordenadas se seguirá mostrando la vista previa de la cámara esperando la respuesta del usuario para iniciar de nuevo el proceso de *tracking* u obtener las gráficas.

En caso de navegar entre pantallas y regresar a *CirclesTracker*, la lista de coordenadas se actualiza a cero al usar de nuevo el botón de *track*. Para ello se ha creado una alerta que avisa al usuario de que en caso de volver al menú principal perderá las coordenadas obtenidas, vaciando la lista de valores.

5.5.2 Flujograma CirclesTracker



Flujograma 4. CirclesTracker

5.6 Gráficas

Este es el segundo y último gran bloque de la aplicación donde se van a procesar todos los datos que se han obtenido en *CirclesTracker* para construir las gráficas que mostrarán el resultado del ejercicio.

Se han diseñado 5 actividades para mostrar las gráficas:

- Actividad *SelecciónGráficas*.
- Actividad *CoordenadasXY*.
- Actividad *Speed*.
- Actividad *Acceleration*.
- Actividad *AccelerationForce*.

5.6.1 SelecciónGráficas

Desde *CirclesTracker* el usuario pide a la aplicación que se muestren las gráficas. Al pulsar el botón se traslada la información a una nueva pantalla que funciona como menú desde donde se puede acceder a cualquier gráfica individual. En la parte central se muestra una vista previa de las tres gráficas más importantes, coordenadas X, coordenadas Y y la velocidad V.

La actividad se llama *SelecciónGráficas* [30].

La interfaz gráfica diseñada queda tal que así:

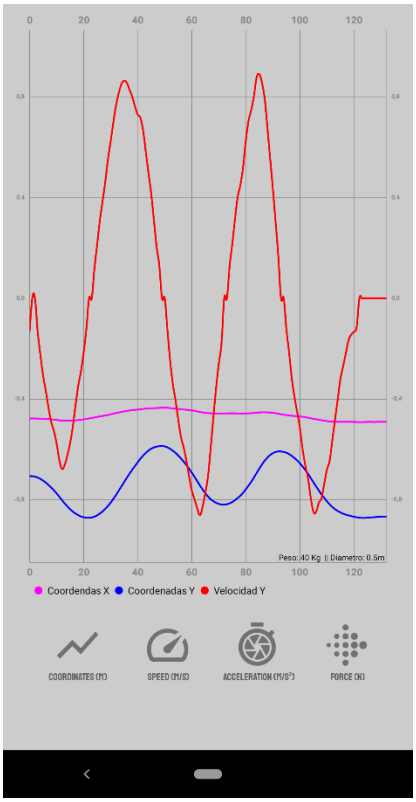


Ilustración 33. Interfaz gráfica de la actividad *SelecciónGráficas*

Esta pantalla ha sido creada para mostrar al usuario la relación que existe entre el movimiento de la pesa y la velocidad. Se puede observar claramente que los picos de velocidad se obtienen en los momentos de subida y bajada de las coordenadas Y (línea azul) y cómo en los momentos pico de esta la velocidad vuelve a cero, debido a que es el momento donde el usuario ha subido del todo la pesa y se dispone a bajarla o viceversa. En esta vista previa la coordenada X sirve simplemente como información complementaria del desplazamiento del usuario durante el ejercicio.

Para la creación de las gráficas se necesita obtener las coordenadas de la actividad anterior mediante el uso de un *Intent* y *bundle*.

Antes de introducir los valores obtenidos a la librería de *MpCharts*, se han seguido los siguientes pasos:

1. Creación de listas de valores *Floats*.
2. Convertir las listas a matrices.
3. Aplicar efecto *Blur* sobre las matrices.
4. Convertir las matrices suavizadas a lista de *Floats*.

Uno de los primeros resultados que se obtuvo fue el siguiente:

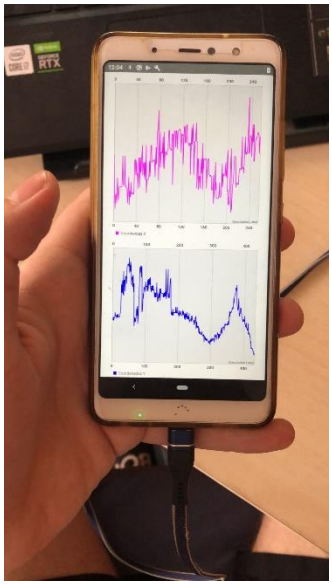


Ilustración 34. Imperfección en la toma de coordenadas

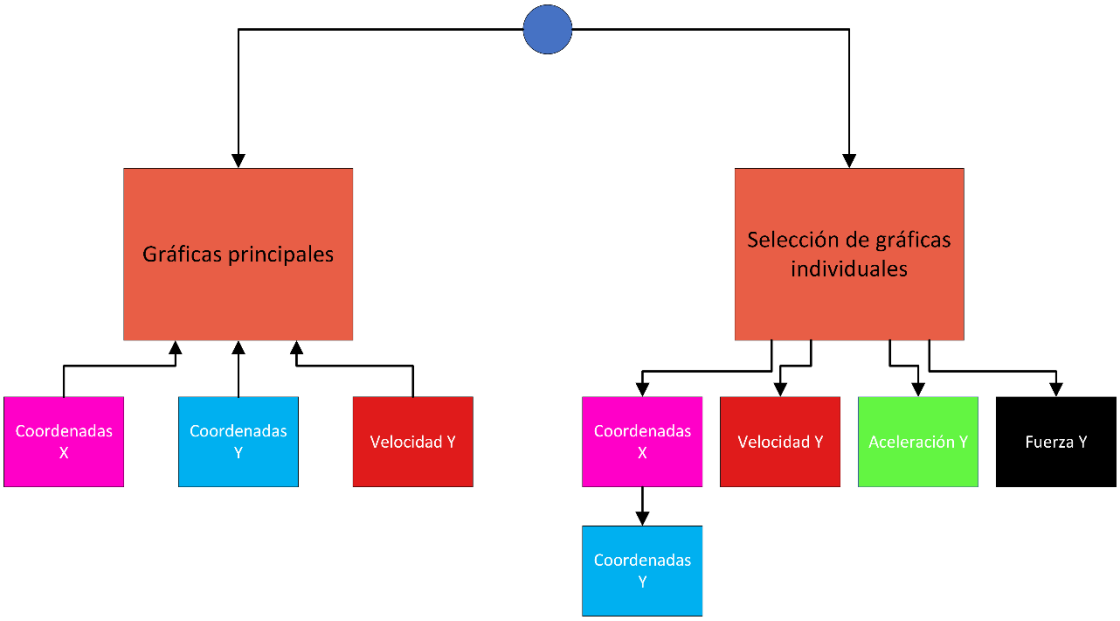
Analizando los resultados obtenidos se decidió aplicar un filtrado gaussiano a los valores obtenidos para lograr una línea limpia con la que sí que se pueda leer correctamente los resultados. Para ello se tuvo que transformar la lista de *floats* que almacenaba las coordenadas en una matriz para poderla suavizar. El código es el siguiente:

```
listaSinFiltrarY= Converters.vector_float_to_Mat(datay);
Imgproc.blur(listaSinFiltrarY,listaFiltradaY,new Size( width: 1,  height: 21));
Converters.Mat_to_vector_float(listaFiltradaY,tempY);
```

Parte de código 26. Proceso de filtrado de coordenadas

Una vez se obtiene una lista de coordenadas filtradas, se procesan y se opera con dichos valores para obtener la velocidad, la aceleración y la fuerza.

5.6.2 Flujograma SelecciónGráficas



Flujograma 5. SelecciónGráficas

5.6.3 CoordenadasXY

El usuario tiene la opción de poder acceder a un interfaz individual de cada tipo de gráfica. A la primera que se puede acceder se llama *Coordenadas(m)*. La actividad obtiene el nombre de *CoordenadasXY*.

Estas gráficas son las más importantes ya que una representación limpia de estas va a mostrar con exactitud el movimiento realizado en el ejercicio y permite obtener el resto de ellas de la forma más clara posible.

La interfaz gráfica diseñada queda tal que así:

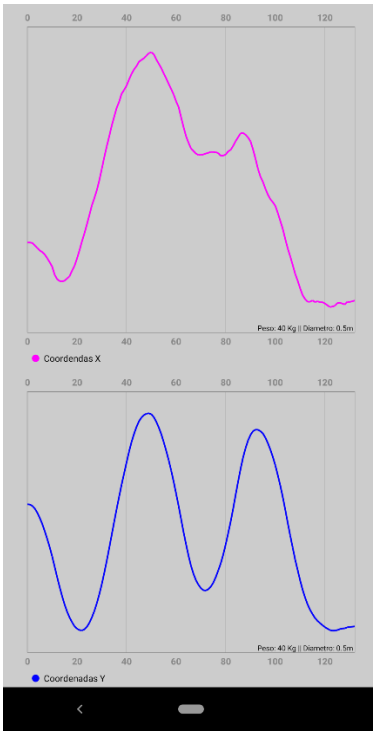


Ilustración 35. Interfaz gráfica de *CoordenadasXY*

Una de las ventajas que ofrece esta librería es poder ampliar a mano cada gráfica y poder ver el valor en el eje Y de cada coordenada.

Visualmente se puede observar que cada tipo de dato lleva consigo su propio color indicado en la leyenda, además de una información extra en la esquina derecha inferior de la gráfica que recuerda en que peso y diámetro del disco se ha realizado el ejercicio.

El código necesario para añadir las coordenadas a la gráfica es el siguiente:

```
//GRÁFICA DE COORDENADAS Y
ArrayList<Entry> lineEntriesY = new ArrayList<~>();
for (int i = 0; i< tempY.size(); i++) { //f.length

    float val;
    val=tempY.get(i);

    lineEntriesY.add(new Entry((float) i, (float) val));

}
```

Parte de código 27. Añadir coordenadas a *MPCharts*

El usuario tiene la posibilidad de ampliar y mover a su antojo la gráfica para ver el valor exacto en cada posición.

Para acceder al resto de gráficas individuales se tiene que volver a la actividad *SelecciónGráficas* mediante el uso del botón del menú de navegación.

5.6.4 Speed

El usuario tiene la opción de poder acceder a un interfaz individual de cada tipo de gráfica. A la segunda que se puede acceder se llama *Speed(m/s)*. La actividad obtiene el nombre de *Speed*.

En esta actividad y las siguientes, la representación de las gráficas sigue los mismos pasos, con el mismo diseño y cambiando solo el color de cada tipo de valor.

El único cambio apreciable en la parte lógica es el tratamiento de las coordenadas para obtener el tipo de valor que se quiere representar, en este caso, la velocidad.

La velocidad se obtiene restando la tercera coordenada menos la primera y dividiendo el resultado por el periodo, que se trata de la inversa de los *fps*. Se sigue la siguiente ecuación:

$$\frac{X[i+1]-X[i-1]}{1/fps}$$

Fórmula 2. Obtención de velocidad

Este proceso se realiza recorriendo el array que contiene las coordenadas Y, mediante el uso de un *for*.

El código es el siguiente:

```
for (int i =2, j=0; i<tempY.size();i++, j++) {


    float val;
    val = tempY.get(i);

    float val1;
    val1 = tempY.get(j);

    float vx1 = (val - val1) / (periodoVelocidad);
```

Parte de código 28. Obtención de valores de velocidad

Estos resultados se añaden a una nueva lista de *floats* en la que se ha rellenado la primera y última posición copiando los valores de las respectivas vecinas para mantener la longitud de la lista de coordenadas y mantener una lógica de tamaño. Para recorrer esa nueva lista de *floats* que contiene los valores de la velocidad, se sigue el mismo procedimiento que para la representación de coordenadas:

```
ArrayList<Entry> lineEntriesVelocityY = new ArrayList<~>();
for (int i = 0; i<vx.size(); i++) {

    lineEntriesVelocityY.add(new Entry((float) i,(float) vx.get(i)));

    lineDataSetVelocityY = new LineDataSet(lineEntriesVelocityY,  label: "Velocidad Y");
```

Parte de código 29. Añadir valores de velocidad a *MPCharts*

La interfaz gráfica diseñada queda tal que así:

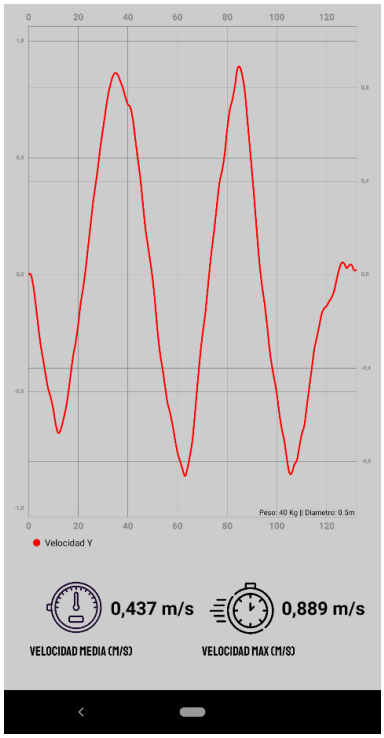


Ilustración 36. Interfaz gráfica de *Speed*

En las interfaces de gráficas individuales se ha añadido el valor medio y el valor máximo como información extra. En el caso de la ilustración anterior se muestra la velocidad media a la que se ha realizado el ejercicio y el valor de la velocidad máxima obtenida.

La velocidad media de la fase positiva del ejercicio se ha obtenido sumando todos los valores positivos de velocidad y dividiendo entre la cantidad de valores. También se muestra el valor de la velocidad máxima.

El código para obtener la velocidad máxima es el siguiente:

```
float valorAlto = 0;
for (int i = 0; i<vx.size(); i++) {
    if(valPositivos.get(i)>valorAlto){
        valorAlto = valPositivos.get(i);
    }
}
```

Parte de código 30. Obtención de velocidad máxima

El usuario tiene la posibilidad de ampliar y mover a su antojo la gráfica para ver el valor exacto en cada posición. Para acceder al resto de gráficas individuales se tiene que volver a la actividad *SelecciónGráficas* mediante el uso del botón del menú de navegación.

5.6.5 Acceleration

El usuario tiene la opción de poder acceder a un interfaz individual de cada tipo de gráfica. A la tercera que se puede acceder se llama *Acceleration(m/s2).* La actividad obtiene el nombre de *Acceleration*.

En esta actividad se va a realizar una operación matemática con los valores de velocidad obtenidos en la anterior actividad.

Anteriormente para obtener la velocidad se debía restar la tercera coordenada menos la primera y posteriormente dividir el resultado por el periodo de la velocidad, pero ahora, para obtener la aceleración, se repite el proceso cambiando las coordenadas por las velocidades obtenidas en la operación de la actividad anterior.

La operación matemática quedaría así:

$$\frac{Vx[i+1]-Vx[i-1]}{1/fps}$$

Fórmula 3. Obtención de aceleración

Este proceso se realiza recorriendo el array que contiene los valores de velocidad, mediante el uso de un *for*.

El código es el siguiente:

```
for (int i =2, j=0; i<vx.size();i++, j++) {

    float ax1 = (vx.get(i) - vx.get(j)) / (periodoVelocidad);
    ax.add(ax1);
}
```

Parte de código 31. Obtención de valores de aceleración

Estos resultados se añaden a una nueva lista de *floats* en la que se ha rellenado la primera y última posición copiando los valores de las respectivas vecinas para mantener la longitud de la lista de coordenadas y mantener una lógica de tamaño al igual que se ha hecho en el apartado de la velocidad.

Para recorrer esa nueva lista de *floats* que contiene los valores de la aceleración se sigue el mismo procedimiento que para la representación de coordenadas:

```
ArrayList<Entry> lineEntriesAccelerateY = new ArrayList<~>();
for (int i =0; i<ax.size();i++) {

    lineEntriesAccelerateY.add(new Entry((float) i, (float) ax.get(i)));

    lineDataSetAccelerateY = new LineDataSet(lineEntriesAccelerateY,  label: "Accelerate Y");
```

Parte de código 32. Añadir valores de aceleración a *MPCharts*

La interfaz gráfica diseñada queda tal que así:

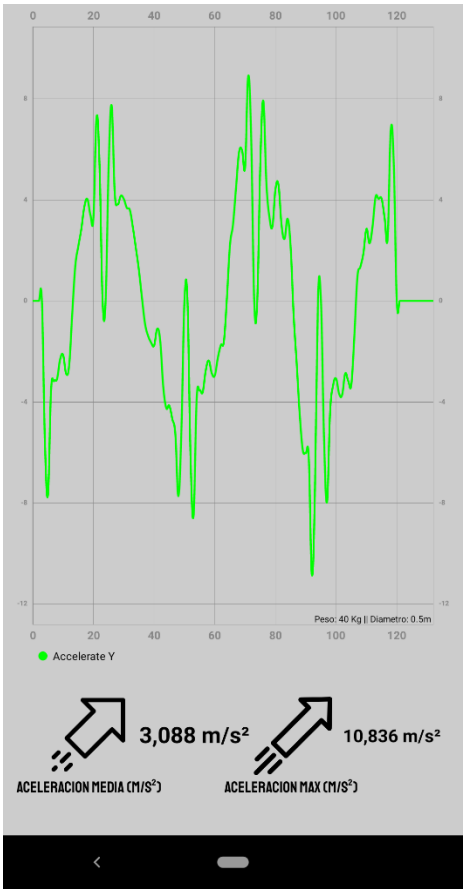


Ilustración 37. Interfaz gráfica de *Acceleration*

En las interfaces de gráficas individuales se ha añadido el valor medio y el valor máximo como información extra. En el caso de la ilustración anterior se muestra la aceleración media a la que se ha realizado el ejercicio y el valor de la aceleración máxima obtenida.

Utilizando el mismo método para obtener la velocidad media y máxima se consigue la aceleración media y máxima.

El código para obtener la aceleración máxima es el siguiente:

```
float valorAlto = 0;
for (int i = 0; i<ax.size(); i++) {
    if(valPositivos.get(i)>valorAlto){
        valorAlto = valPositivos.get(i);
    }
}
```

Parte de código 33. Obtención de aceleración máxima

El usuario tiene la posibilidad de ampliar y mover a su antojo la gráfica para ver el valor exacto en cada posición. Para acceder al resto de gráficas individuales se tiene que volver a la actividad *SelecciónGráficas* mediante el uso del botón del menú de navegación.

5.6.6 AccelerationForce

El usuario tiene la opción de poder acceder a un interfaz individual de cada tipo de gráfica. A la tercera que se puede acceder se llama *Force(N).* La actividad obtiene el nombre de *AccelerationForce*.

En esta última actividad se va a realizar una operación matemática con los valores de aceleración obtenidos en la anterior actividad.

A diferencia de las anteriores actividades solo se va a sumar a las medidas de la aceleración el valor de la gravedad para posteriormente multiplicar el resultado por el peso del disco que se ha introducido en la *Principal Activity*.

La operación matemática quedaría así:

$$(Ax + 9.8) * PesoKg$$

Fórmula 4. Obtención de aceleración

Este proceso se realiza recorriendo el array que contiene los valores de velocidad, mediante el uso de un *for*.

El código es el siguiente:

```
for (int i =0; i<ax.size();i++) {

    float f = (ax.get(i)+9.8f)*(peso_kg); //los kilos que sean
    fx.add(f);
```

Parte de código 34. Obtención de valores de fuerza

Para añadir los valores de la fuerza a la gráfica se usa el mismo código de la actividad *CoordenadasXY*.

La interfaz gráfica queda tal que así:

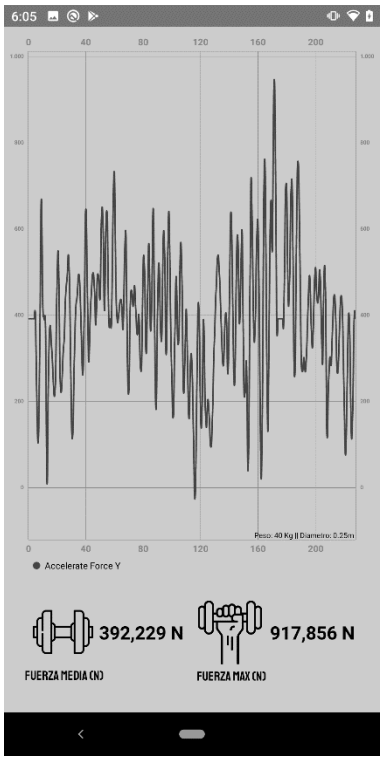


Ilustración 38. Interfaz gráfica de *AccelerationForce*

En las interfaces de gráficas individuales se ha añadido el valor medio y el valor máximo como información adicional. En el caso de la ilustración anterior se muestra la fuerza media a la que se ha realizado el ejercicio y el valor de la fuerza máxima realizada.

La fuerza media se ha obtenido sumando todos los valores y dividiendo entre la cantidad de valores. También se ha obtenido el valor de la fuerza máxima.

El código para obtener la fuerza máxima es el siguiente:

```
float valorAlto = 0;
for (int i = 0; i<fx.size(); i++) {
    if(valReales.get(i)>valorAlto){
        valorAlto = valReales.get(i);
    }
}
```

Parte de código 35. Obtención de fuerza máxima

El usuario tiene la posibilidad de ampliar y mover a su antojo la gráfica para ver el valor exacto en cada posición. Para acceder al resto de gráficas individuales se tiene que volver a la actividad *SelecciónGráficas* mediante el uso del botón del menú de navegación.

5.7 Control de entrenamiento

Durante la sesión de entrenamiento el usuario puede realizar varios ejercicios en los que quiera *trackear* su movimiento. Estos ejercicios pueden realizarse con el mismo o distinto peso y tal vez al terminar cada uno de ellos quiera apuntarse cómo le ha ido, por lo que se ha diseñado una nueva actividad llamada *Account* en la que puede dejar escrito un seguimiento de los ejercicios realizados.

5.7.1 Account

La actividad *Account* se basa en un *TableLayout* [31] que contiene la información básica para poder mantener un control de los ejercicios realizados. Estos datos son:

1. Nombre del usuario.
2. Ejercicio realizado.
3. Peso utilizado.
4. Hora.
5. Dia, mes y año.

En el caso de los dos primeros datos se introducen mediante el uso de dos *EditText* diseñados en la misma pantalla ya que es la única información que aún no ha sido declarada en la aplicación hasta el momento.

Los siguientes tres datos ya forman parte de la aplicación por lo que simplemente se deben obtener en la actividad y mostrarlos por pantalla. En el caso del peso, este ha sido declarado anteriormente en la *Principal Activity* con lo que solamente es necesario un *bundle* para obtener la variable de datos, mientras que los datos de tiempo, como lo son la hora y el día, simplemente se obtienen gracias al uso de una clase integrada en *Android Studio* llamada *Time*.

```
Time today = new Time(Time.getCurrentTimezone());
today.setToNow();
int dia = today.monthDay;
int mes = today.month;
int ano = today.year;
int hora = today.hour;
int minuto = today.minute;
```

Parte de código 36. Clase *Time* de *Android Studio*

Una vez obtenidos por separados todos los datos de tiempo, peso e información del ejercicio se diseña un *TableLayout* que añadirá una fila más con toda la información cada vez que el usuario haya realizado un ejercicio nuevo.

El usuario debe escribir en los dos *EditText* su nombre y el nombre del último ejercicio realizado y finalmente pulsar el botón de Guardar para que se muestre toda la información junta.

La interfaz gráfica tiene el siguiente aspecto:

Ilustración 39. Interfaz gráfica de *Account*

5.8 Instrucciones de uso

En la redacción de esta memoria se ha ido explicando actividad a actividad su funcionamiento. Además, se han diseñado alertas en la propia aplicación que acompañan e informan al usuario, aunque esto no es suficiente.

Para conseguir la mejor experiencia de uso se han diseñado unas nuevas actividades que aportan al usuario más información detallada del funcionamiento de cada pantalla de la aplicación.

En la *Principal Activity* se ha diseñado un menú de navegación que contiene la actividad *Help*, en la que se ha descrito toda la parte lógica y gráfica de las instrucciones.

5.8.1 Help

Al analizar otras aplicaciones similares a la que se ha creado en este proyecto, se ha comprobado que en todas existía un submenú de instrucciones que explican al usuario cómo utilizar la aplicación.

Este aspecto era muy interesante por lo que se ha decidido trabajar en un bloque de actividades que aportará al usuario una información más clara y eficaz.

La actividad *Help* es la interfaz principal del listado de instrucciones en la que se ha diseñado un *RecyclerView* con distintas imágenes desde las que se accede a la pantalla de información.

La interfaz gráfica tiene el siguiente aspecto:

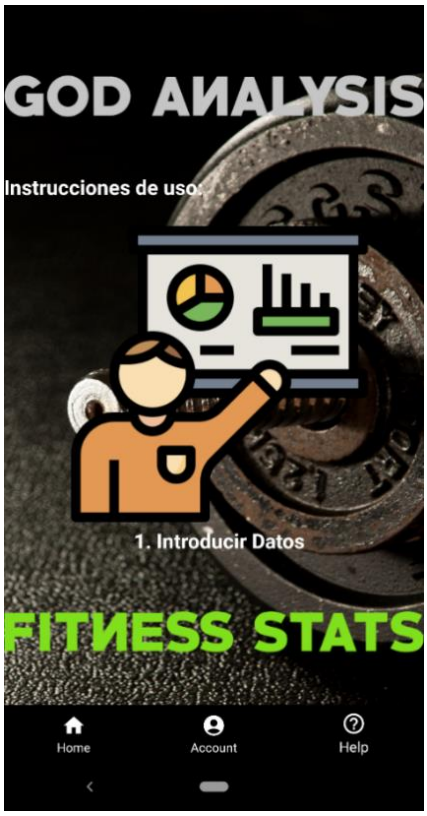


Ilustración 40. Interfaz gráfica de *Help*

Como se observa en la imagen anterior el primer paso que requiere la aplicación es la introducción de datos. En caso de no entender cómo se deben introducir estos datos, el usuario puede acceder a una nueva pantalla al pulsar la imagen donde se han escrito las instrucciones de uso.

En esta actividad se han declarado todos los nombres e información necesaria para el *RecyclerView* [32].

Un *RecyclerView* facilita que se muestren grandes conjuntos de datos de manera eficiente. Para ello se deben facilitar los datos que se quiera que contenga y definir el aspecto que se quiera para cada elemento.

Para crear una interfaz gráfica un poco distinta a lo que se puede encontrar en cualquier otra aplicación se ha decidido crear un *RecyclerView*, que usara un *scroll* de manera horizontal mostrando los iconos y su nombre en grande, ayudando al usuario a ver todas las instrucciones a las que se puede acceder.

El listado de instrucciones disponible es:

1) Introducir datos.
2) Real-Time.
3) Obtención de gráficas.

Primero se han definido todos los elementos que se han incluido en la lista dinámica. Como se observa en la imagen, cada elemento posee un icono y un título numerado que indica los pasos a seguir.

Para ello se ha declarado toda la información necesaria en la actividad:

```
recyclerView = findViewById(R.id.recycler_view);

Integer[] pasosImg = {R.drawable.introducir_datos, R.drawable.escoger_camara, R.drawable.realtime, R.drawable.graficas};

String[] pasosName= {"1. Introducir Datos", "2. Real Time", "3. Obtención de Gráficas"};
```

Parte de código 37. Introducción de imágenes y títulos

De esta manera el *RecyclerView* añade la información a los elementos de manera sucesiva evitando tener que crear distintos botones, que luego habría que programarlos individualmente. Esto cumple con la definición de una lista dinámica.

5.8.2 Diseño de Help2

Esta actividad se corresponde con la pantalla a la que el usuario accede al pulsar uno de los iconos del anterior *RecyclerView*, diseñado en la actividad *Help*.

La interfaz gráfica de esta actividad tiene el siguiente aspecto:

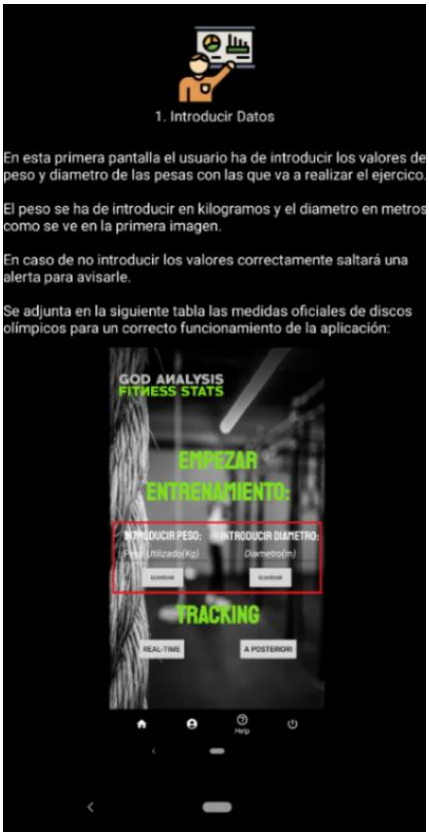


Ilustración 41. Interfaz gráfica de Help2

La pantalla que se muestra en la imagen anterior corresponde con las instrucciones del primer paso, la introducción de datos.

Todas las pantallas comparten el mismo aspecto. Todas contienen el icono principal con el título que informa al usuario en que paso está, una descripción escrita de los pasos a seguir en la parte de la aplicación en la que se encuentre, una captura de pantalla de la aplicación para aportar información gráfica y una imagen extra.

En ciertos pasos se ha dado el caso de que los elementos no se podían mostrar enteros en una pantalla de móvil por lo que se ha diseñado un *scroll* [33] vertical, creando un diseño más actualizado y la posibilidad de incluir una descripción más exhaustiva.

Se ha diseñado el siguiente esquema para entender la estructura de esta actividad:

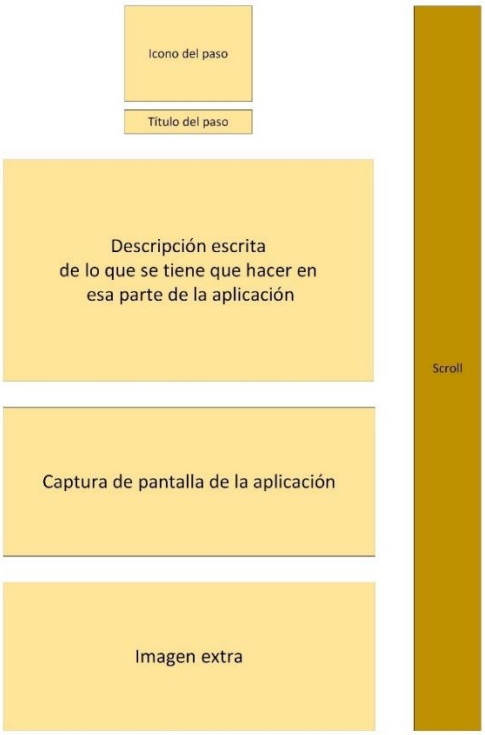


Ilustración 42. Esquema de Help2

La parte lógica de esta actividad que contiene los datos de cada elemento de la imagen anterior es la siguiente:

```
Bundle extras = getIntent().getExtras();
String name= extras.getString( key: "Nombre");
int img= extras.getInt( key: "Img");
String descrip = extras.getString( key: "Descrip");
int imgDescrip= extras.getInt( key: "ImgDescrip");
int imgExtra= extras.getInt( key: "ImgExtra");

TextView namePaso = findViewById(R.id.pasoName);
ImageView imgPaso = findViewById(R.id.pasoimg);
TextView descripPaso = findViewById(R.id.pasoDescrip);
ImageView imgPasoDescrip = findViewById(R.id.pasoimgdescrip);
ImageView imgExtra1 = findViewById(R.id.imgextra);

namePaso.setText(name);
imgPaso.setImageResource(img);
descripPaso.setText(descrip);
imgPasoDescrip.setImageResource(imgDescrip);
imgExtra1.setImageResource(imgExtra);
```

Parte de código 38. Parte lógica de Help2

Todos los elementos que forman esta actividad han sido declarados en el código de la anterior actividad *Help*. Para ello se ha descrito en un array de *strings* en el que se han escrito todas las descripciones correspondientes a cada paso, así como también se han descrito dos *arrays* de Integer para introducir las capturas de pantalla e imágenes extras.

```
Integer[] pasosImg = {R.drawable.introducir_datos, R.drawable.escoger_camara, R.drawable.realtime, R.drawable.graficas};

String[] pasosName= {"1. Introducir Datos", "2. Escoger modo", "3. Real Time", "3.1. Obtención de Gráficas"};

String[] pasosDescrip= {"En esta primera pantalla el usuario ha de introducir los valores del peso y diametro de las pesas con la
        "el ejercico.\n\nEl peso se ha de introducir en kilogramos y el diametro en metros como se ve en la primera imagen.\n\nEn
        "los valores correctamente saltará una alerta para avisarle.\n\nSe adjunta en la siguiente tabla las medidas oficiales d
        "la aplicación:",

        "Una vez introducidos los valores de manera correcta se procede al 'tracking' de los discos para obtener el analisis del
                "\n\nComo se ve en la imagen superior, existen dos modos de trackeo que el usuario puede elegir.\n\n1) Real-Time
                "tiempo real.\n\n2) A Posteriori: se realiza el track a partir de un video existente en la galeria o incluso grab
                ,
        "Esta será la interfaz gráfica con la que se encontrará al seleccionar Real-Time. Se ha diseñado de manera que sea intuit
                "Existen cinco botones con sus correspondientes funciones, se explicarán a continuación de izquierda a derecha y
                "1) Cambio de cámara: Intercambio de vista entre la camara frontal y posterior del dispositivo móvil.\n\n" +
                "2) Flash: Activar/Desactivar flash.\n\n" +
                "3) Cambio de peso: Se abrirá un cuadro de diálogo donde le pedirá introducir de nuevo el peso y el diametro del
                "4) Track: Botón que inicia el proceso de trackeo.\n\n" +
                "5) Obtención de gráficas: Se obtienen las gráficas de analisis producidas por el tracking.",
        "Tras realizar el proceso de trackeo accedemos a esta pantalla donde en la interfaz principal podemos ver representadas l
                "coordenadas verticales y horizontales, y la gráfica de la velocidad en el eje vertical.\n\n" +
                "La leyenda nos proporciona la información de los colores de cada representación para una fácil detección.\n\n" +
                "Además, tenemos una breve descripción en la esquina inferior derecha que nos informa del peso y diametro para co
                "Por último, esta aplicación trae como novedad el hecho de poder  obtner las gráficas de aceleración y fuerza. Pa
                "poder acceder a las gráficas individualmente y obtener una información más detallada."};

Integer[] pasoImgDescrip = {R.drawable.introducirdatos ,R.drawable.escogermodo
        ,R.drawable.circlestracker, R.drawable.selecciongraficas};

Integer[] imgExtras = {R.drawable.discosolimpicos ,R.drawable.titulo1app
        ,R.drawable.titulo2app, R.drawable.titulo1app};
```

Parte de código 39. Introducción de datos necesarios para *Help2*

Para unir los datos correspondientes a cada elemento del *RecyclerView* se ha creado una clase adaptador desde cero.

Un adaptador es un objeto de una clase que implementa la interfaz *Adapter*, actuando como enlace entre datos y un adaptador View. El adaptador es el responsable de recuperar los datos desde un conjunto de ellos y generar objetos de tipo View, tantos como elementos se hayan creado en el *RecyclerView* de nuestra aplicación.

Una vez se obtienen los datos, el adaptador permite minimizar las operaciones de formatos grandes y reciclar los objetos View que quedan fuera de la pantalla, manteniendo el bajo consumo de CPU del móvil.

```
@Override
public void onBindViewHolder(@NonNull ViewHolder holder, @SuppressLint("RecyclerView") int position) {
    holder.imageView.setImageResource(mainModels.get(position).getPasoImg());
    holder.textView.setText(mainModels.get(position).getPasoName());

    holder.itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(holder.itemView.getContext(), Help2.class);
            intent.putExtra( name: "Img", mainModels.get(position).getPasoImg());
            intent.putExtra( name: "Nombre", mainModels.get(position).getPasoName());
            intent.putExtra( name: "Descrip", mainModels.get(position).getPasoDescrip());
            intent.putExtra( name: "ImgDescrip", mainModels.get(position).getPasoImgDescrip());
            intent.putExtra( name: "ImgExtra", mainModels.get(position).getImgExtra());
            holder.itemView.getContext().startActivity(intent);

        }
    });
}
```

Parte de código 40. Método *onBindViewHolder()* de la clase adaptador

Finalmente se envían todos los datos a la actividad *Help2* que se encarga de mostrarlos gráficamente en cuanto reciba las órdenes del usuario.

Capítulo 6. Resultados

Una vez se ha explicado con detenimiento cada apartado de la aplicación se van a mostrar una serie de capturas de pantalla de la aplicación en funcionamiento. Para comprobar que la detección de pesas está funcionando de manera correcta se han realizado distintos ejercicios con discos (las pesas) y personas diferentes.

6.1 Curl Bíceps

Para hacer el ejercicio [34] se colocan los discos en los extremos de una barra (plana, en W o E-Z) y se pegan los codos a los costados del cuerpo (sin moverlos) y se debe subir y bajar lentamente el peso estirando el brazo completamente.

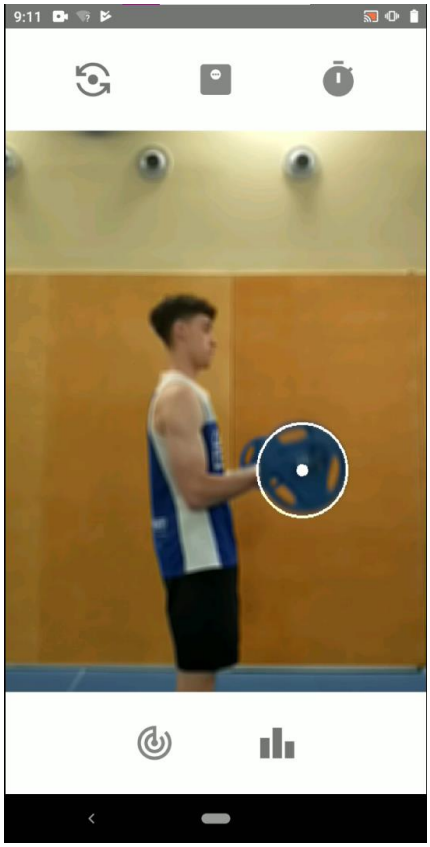


Ilustración 43. Ejercicio *Curl* Bíceps

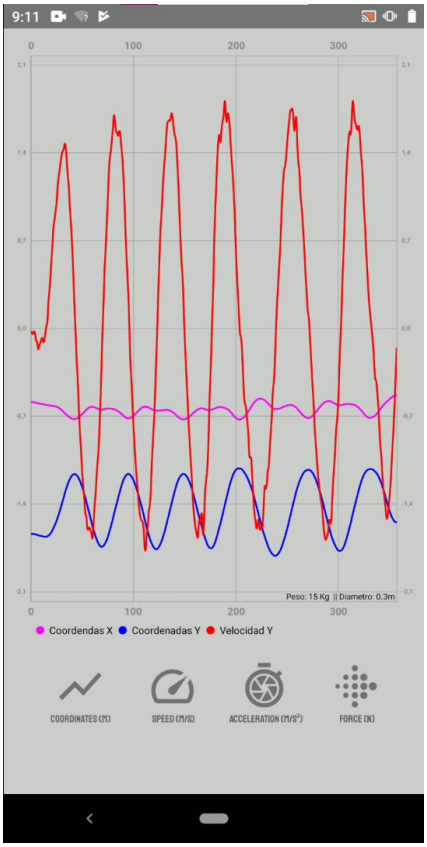


Ilustración 44. Resultados *Curl* Bíceps

Las medidas se han tomado en una sala con un fondo liso y usando unas pesas de color azul para comprobar que el algoritmo no da problemas al acercarse a una camiseta de un color similar.

El seguimiento de la pesa es correcto y se obtienen unos resultados buenos. Se ha conseguido evitar la detección de círculos falsos estableciendo un radio máximo para el círculo *trackeado*.

Gracias al uso del temporizador configurable, el usuario puede configurar el tiempo que necesita para darle al *play* del tracking y levantar la pesa para empezar el movimiento.

6.2 Press Banca

El press banca es un ejercicio [35] en el que se usa peso libre para trabajar el tronco superior del cuerpo. Este es un ejercicio que se utiliza para trabajar el pecho, tríceps y deltoides anterior.

La realización de este ejercicio consiste en levantar y bajar una barra de un peso considerable con los discos a los extremos por encima de la altura del pecho. Se realiza acostándose sobre un banco horizontal o ligeramente inclinado dependiendo del grupo muscular que se quiera trabajar.

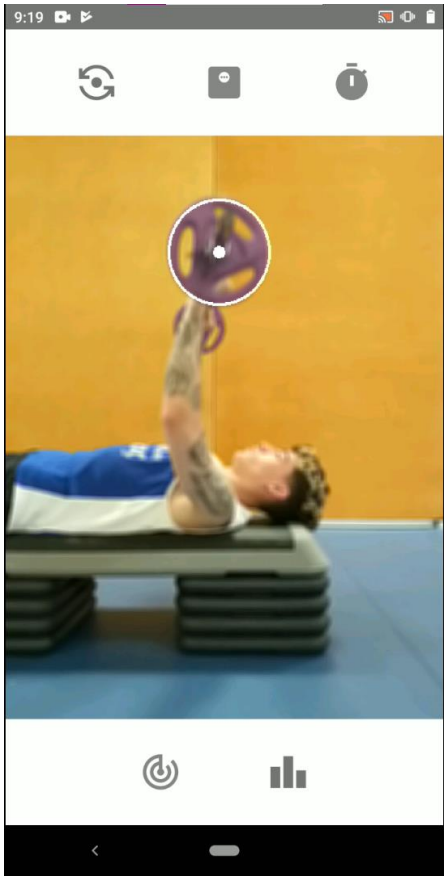


Ilustración 45. Ejercicio *Press* Banca

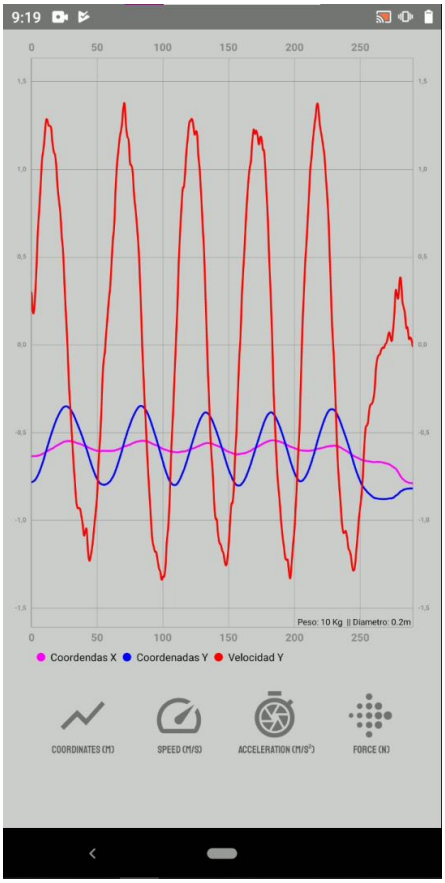


Ilustración 46. Resultados *Press* Banca

El seguimiento de la pesa es correcto y se obtienen unos buenos resultados. Se ha conseguido evitar la detección de círculos falsos estableciendo un radio máximo para el círculo *trackeado*.

En este caso el movimiento en la coordenada X es más lineal cuando el ejercicio está bien realizado. Los picos mínimos de las coordenadas Y son siempre los mismos ya que el punto más bajo se produce al tocar la barra con el pecho, al contrario que en *Curl Bíceps* que dependiendo de la repetición y el cansancio del usuario baja más en unas que otras.

Gracias al uso del temporizador configurable el usuario puede configurar el tiempo que necesita para darle al *play* del tracking y volver al banco donde está sentado durante la realización del ejercicio.

6.3 Peso Muerto

El peso muerto es un ejercicio [36] en el que se usa peso libre para trabajar la espalda y las piernas. Este es uno de los tres ejercicios que forman parte del powerlifting. Además, influye en el fortalecimiento del sistema nervioso central.

La realización de este ejercicio consiste en levantar la barra que está apoyada en el suelo con peso a los extremos. La barra debe llegar a la altura de la cintura manteniendo la espalda recta en todo el movimiento. Se empieza con el tronco inclinado y las rodillas flexionadas por encima de la barra, se debe levantar sin tirones y mantener un movimiento constante.

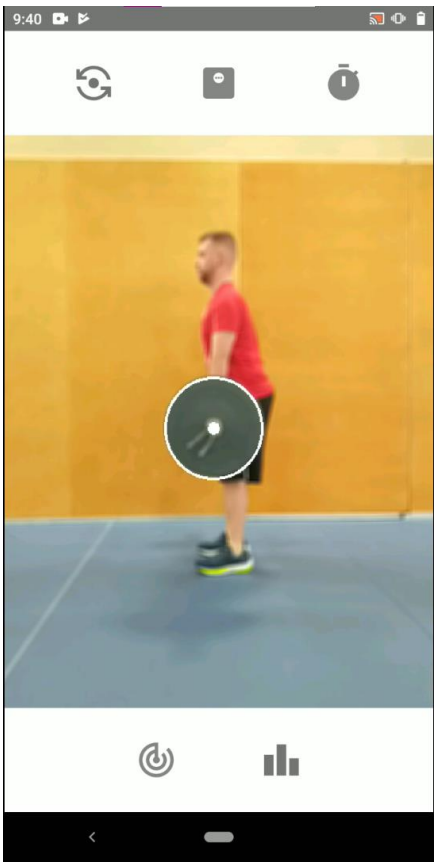


Ilustración 48. Ejercicio Peso Muerto

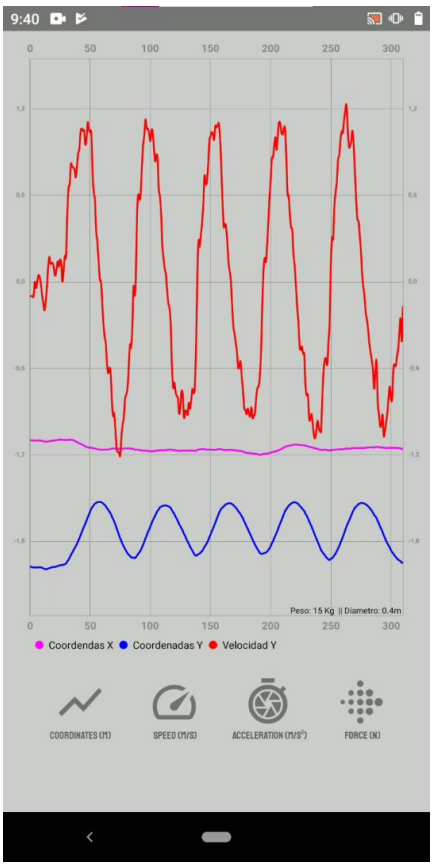


Ilustración 47. Resultados Peso Muerto

El seguimiento de la pesa es correcto y se obtienen unos resultados buenos. Se ha conseguido evitar la detección de círculos falsos estableciendo un radio máximo para el círculo *trackeado*.

Durante el movimiento realizado en esta prueba se obtienen resultados parecidos a los del ejercicio del *Press Banca*, con coordenadas X bastante lineales y las coordenadas Y desde el suelo hasta la cintura. En este caso las medidas de velocidad son un poco más irregulares debido a que el usuario está más alejado de la cámara y la pesa no estaba bien fijada en el momento de la grabación para ver cómo reaccionaba la aplicación.

Las medidas se han tomado en una sala con un fondo liso y usando unas pesas de color gris que no da problemas al acercarse a la camiseta del usuario.

Gracias al uso del temporizador configurable el usuario puede configurar el tiempo que necesita para darle al *play* del tracking y levantar la pesa para empezar el movimiento

6.4 Elevación frontal

Las elevaciones frontales es un ejercicio [37] realizado con barra o mancuernas que trabaja los músculos del hombro, centrándose en el deltoides anterior y deltoides medio.

La realización de este ejercicio consiste en levantar la barra hasta la altura de los hombros manteniendo los brazos lo más estirados posibles. Se debe formar un ángulo de 90º entre brazos y torso. Se puede realizar sentado en un banco o de pie.

Ilustración 49. Ejercicio Elevación Frontal

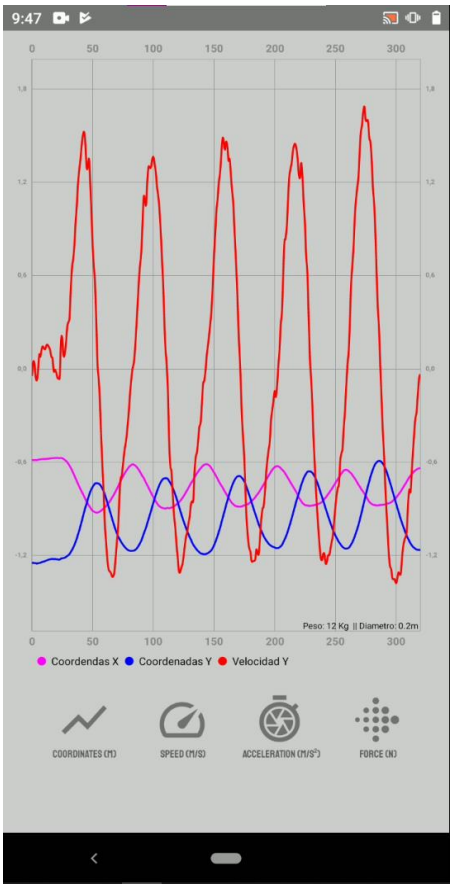


Ilustración 50. Resultados Elevación Frontal

El seguimiento de la pesa es correcto y se obtienen unos resultados buenos. Se ha conseguido evitar la detección de círculos falsos estableciendo un radio máximo para el círculo *trackeado*.

Durante el movimiento realizado en esta prueba se obtienen resultados parecidos a los del ejercicio del *Curl Bíceps*, con unas coordenadas X que varían debido a que, en este ejercicio, no solo se mueve en el eje vertical.

Gracias al uso del temporizador configurable el usuario puede configurar el tiempo que necesita para darle al *play* del tracking y levantar la pesa para empezar el movimiento.

6.5 Especificaciones técnicas de la aplicación

La aplicación funciona adecuadamente y deben tenerse en cuenta las siguientes especificaciones:

- El fondo sobre el que se debe realizar el ejercicio debe ser relativamente claro y liso para intentar evitar que la transformada de *Hough* genere posibles círculos erróneos.
- Mejorar la precisión de la detección que ofrece la transformada de *Hough* investigando otros algoritmos.

- La detección de las pesas está configurada a un cierto tamaño de círculo para mejorar la precisión de la detección siendo necesario que el usuario se coloque a una cierta distancia del móvil si las pesas fueran extremadamente pequeñas o grandes.
- Debido a la alta demanda de CPU provocada en el móvil utilizado se ha reducido la resolución de la imagen de 1080 a 720 para poder así trabajar a 25 *fps*.
- El tracking de objetos ha sido diseñado para realizarse en tiempo real pudiéndose ampliar a un análisis posterior a través de videos ya grabados.

6.6 Resultados en gráficas específicas

A continuación, se agregan capturas de las gráficas de cada conjunto de valores que se han obtenido en la medida de *Curl Bíceps*:

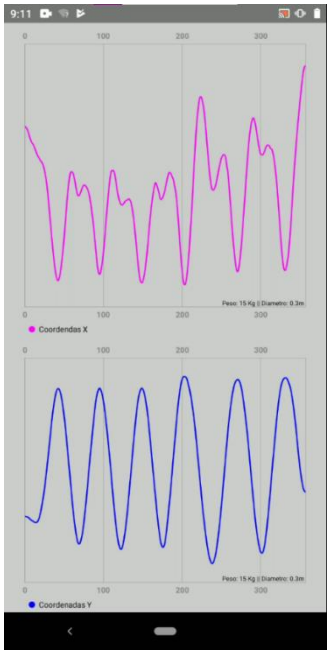


Ilustración 51. Gráficas de las Coordenadas

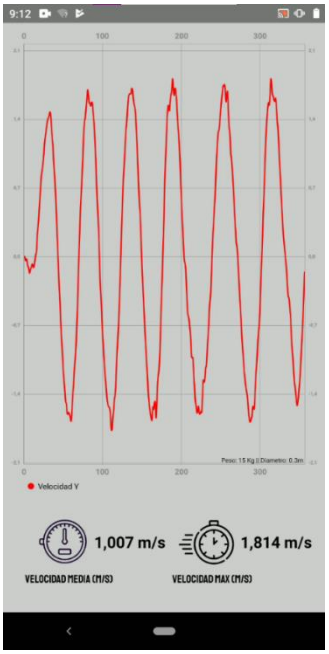


Ilustración 52. Gráfica de la Velocidad

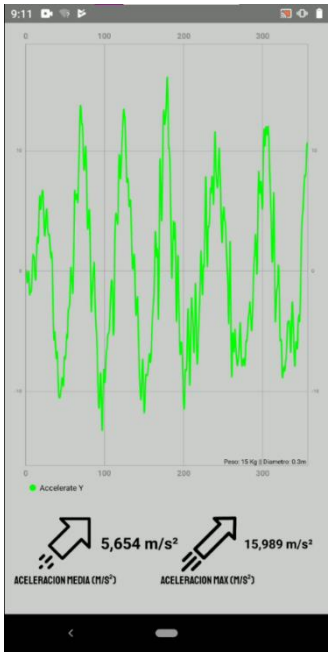


Ilustración 53. Gráfica de la Aceleración

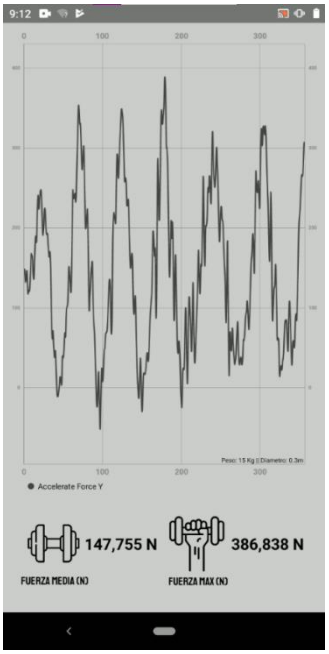


Ilustración 54. Gráfica de la Fuerza

Capítulo 7. Conclusiones

Las conclusiones obtenidas tras el desarrollo de la aplicación son las siguientes:

1. La realización de esta parte del proyecto global ha permitido explorar en profundidad la funcionalidad y aplicación práctica del *Machine Learning* y la Visión Artificial gracias al uso de *OpenCV*.

2. La aplicación ha conseguido realizar la detección, seguimiento y análisis del movimiento con la cámara de un móvil y la librería de *OpenCV*.

3. La aplicación se encuentra en un estado de completa funcionalidad, permitiendo una navegación adecuada entre pantallas. La creación de alertas e instrucciones ofrecen al usuario un uso correcto.

4. La detección de las pesas funciona en distintas situaciones y con persona distintas. La detección de círculos falsos se ha minimizado gracias a la configuración correcta de la trasformada de *Hough*.

5. Aunque la calidad de los resultados obtenidos es mejorable, el balance general es positivo ya que se han cumplido tanto el objetivo principal como los secundarios.

Capítulo 8. Bibliografía

[1] «Beneficios del entrenamiento de fuerza,» 2022. [En línea]. Available: https://www.medical-exercise.com/beneficios-del-entrenamiento-de-fuerza/. [Último acceso: Marzo 2022].

[2] J. H. G. Peña, «Iniciación al entrenamiento de fuerza en edades tempranas,» Enero 2015. [En línea]. Available: https://www.elsevier.es/es-revista-revista-andaluza-medicina-del-deporte-284-articulo-iniciacion-al-entrenamiento-fuerza-edades-S1888754615000830. [Último acceso: Marzo 2022].

[3] J. A. Ruiz, «Diferencias entre entrenar con pesas y máquinas,» 13 Agosto 2019. [En línea]. Available: https://ossfitness.com/diferencias-entrenar-con-pesas-y-maquinas/. [Último acceso: Marzo 2022].

[4] M. P. C. Inmaculada Aznar, «Impacto de las apps móviles en la actividad física: un meta-análisis,» 1 Julio 2019. [En línea]. Available: https://recyt.fecyt.es/index.php/retos/article/view/66628. [Último acceso: Marzo 2022].

[5] «WL Analysis,» [En línea]. Available: https://wlanalysis.com/. [Último acceso: Marzo 2022].

[6] Iberdrola, «Descubre los principales beneficios del 'Machine Learning',» 2022. [En línea]. Available: https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico#:~:text=El%20Machine%20Learning%20es%20una,elaborar%20predicciones%20(an%C3%A1lisis%20predictivo).. [Último acceso: Junio 2022].

[7] Sas, «Aprendizaje automático,» 2022. [En línea]. Available: https://www.sas.com/es_es/insights/analytics/machine-learning.html. [Último acceso: Junio 2022].

[8] Iberdrola, «¿Qué es la visión artificial y cuáles son sus aplicaciones?,» 2022. [En línea]. Available: https://www.iberdrola.com/innovacion/vision-artificial. [Último acceso: Junio 2022].

[9] TensorFlow, «Deploy machine learning models on mobile and edge devices,» [En línea]. [Último acceso: Marzo 2022].

[10] J. Buhigas, «Todo lo que necesitas saber sobre TensorFlow, la plataforma para Inteligencia Artificial de Google,» 14 Febrero 2018. [En línea]. Available: https://puentesdigitales.com/2018/02/14/todo-lo-que-necesitas-saber-sobre-tensorflow-la-plataforma-para-inteligencia-artificial-de-google/#:~:text=TensorFlow%20es%20una%20gran%20plataforma,razonamiento%20usados%20por%20los%20humanos.. [Último acceso: Junio 2022].

[11] Jcosmoscc, «El aprendizaje automático de TensorFlow Lite se integra en Android para mejorar el tamaño y rendimiento de las aplicaciones,» 9 Julio 2021. [En línea]. Available: https://www.xatakandroid.com/sistema-operativo/aprendizaje-automatico-tensorflow-lite-se-integra-android-para-mejorar-tamano-rendimiento-aplicaciones. [Último acceso: Junio 2022].

[12] AWS, «Amazon Rekognition,» 2022. [En línea]. Available: https://aws.amazon.com/es/rekognition/. [Último acceso: Julio 2022].

[13] R. d. l. Vega, «Introducción al procesamiento de imágenes en Python con OpenCV,» 2022. [En línea]. Available: https://pharos.sh/introduccion-al-procesamiento-de-imagenes-en-python-con-opencv/. [Último acceso: Junio 2022].

[14] Android Studio, «Descargar Android Studio,» 1 Febrero 2021. [En línea]. Available: https://developer.android.com/studio. [Último acceso: Marzo 2022].

[15] Java, «Get Java for desktop applications,» 2022. [En línea]. Available: https://www.java.com/es/. [Último acceso: Marzo 2022].

[16] Wikipedia, «OpenCV,» 23 Abril 2022. [En línea]. Available: https://es.wikipedia.org/wiki/OpenCV. [Último acceso: Marzo 2022].

[17] OpenCV, «Releases,» 30 Diciembre 2021. [En línea]. Available: https://opencv.org/releases/. [Último acceso: Marzo 2022].

[18] J. Vicente, «Como configurar OpenCV en Android Studio para Windows de 64 bits,» 25 Noviembre 2018. [En línea]. Available: https://github.com/yucaret/OpenCV-AndroidStudio. [Último acceso: Marzo 2022].

[19] P. Jay, «MPAndroidChart,» 21 Junio 2021. [En línea]. Available: https://github.com/PhilJay/MPAndroidChart. [Último acceso: Abril 2022].

[20] P. Ramirez, «¿Cuales son los sistemas operativos más usados en 2012?,» 4 Enero 2022. [En línea]. Available: https://itsoftware.com.co/content/sistemas-operativos-mas-usados/. [Último acceso: Marzo 2022].

[21] Developers Android, «Documentation Activity,» 8 Julio 2022. [En línea]. Available: https://developer.android.com/reference/android/app/Activity. [Último acceso: Marzo 2022].

[22] Developers Android, «Documentation Layouts,» 27 Octubre 2021. [En línea]. Available: https://developer.android.com/guide/topics/ui/declaring-layout. [Último acceso: Marzo 2022].

[23] Developers Android, «Documentation Intents,» 8 Julio 2022. [En línea]. Available: https://developer.android.com/reference/android/content/Intent. [Último acceso: Marzo 2022].

[24] Developers Android, «App Manifest Overview,» 8 Julio 2022. [En línea]. [Último acceso: Marzo 2022].

[25] Developers Android, «Configure your build,» 13 Julio 2022. [En línea]. Available: https://developer.android.com/studio/build. [Último acceso: Marzo 2022].

[26] Developers Android, «EditText,» 8 Julio 2022. [En línea]. Available: https://developer.android.com/reference/android/widget/EditText. [Último acceso: Abril 2022].

[27] O. Garcia, «Abrir cámara con aplicación de android,» 18 Agosto 2017. [En línea]. Available: https://es.stackoverflow.com/questions/95800/abrir-c%C3%A1mara-con-aplicaci%C3%B3n-de-android. [Último acceso: Mayo 2022].

[28] Juhirajput004, «Java Program to Blur Images using OpenCV,» 17 Mayo 2021. [En línea]. Available: https://www.geeksforgeeks.org/java-program-to-blur-images-using-opencv/. [Último acceso: Mayo 2022].

[29] T. Martinez, «La Transformada de Hough. Detección de Líneas y Círculos,» [En línea]. Available: https://porprofesionalmic.files.wordpress.com/2015/09/investigacion-documental-transformada-hough.pdf. [Último acceso: Mayo 2022].

[30] G. Muniz, 7 Agosto 2015. [En línea]. Available: https://stackoverflow.com/questions/26873445/plot-multiple-charts-in-one-in-mpandroidchart. [Último acceso: Mayo 2022].

[31] Android Studio, «Tabla,» 22 Enero 2021. [En línea]. Available: https://developer.android.com/guide/topics/ui/layout/grid?hl=es-419. [Último acceso: Junio 2022].

[32] Developers Android, «Create dynamic lists with RecyclerView,» 8 Julio 2022. [En línea]. Available: https://developer.android.com/guide/topics/ui/layout/recyclerview. [Último acceso: Mayo 2022].

[33] Develou, «Desplazar Contenido Con El ScrollView En Android,» 2019. [En línea]. Available: https://www.develou.com/scrollview-en-android/. [Último acceso: Junio 2022].

[34] Wikipedia, «Curl de Bíceps,» 1 Junio 2022. [En línea]. Available: https://en.wikipedia.org/wiki/Biceps_curl#:~:text=A%20biceps%20curl%20usually%20starts,weight%20to%20the%20starting%20position.. [Último acceso: Junio 2022].

[35] Wikipedia, «Press de banca,» 17 Abril 2021. [En línea]. Available: https://es.wikipedia.org/wiki/Press_de_banca#:~:text=El%20press%20de%20banca%2C%20press,la%20zona%20superior%20del%20cuerpo.. [Último acceso: Junio 2022].

[36] Wikipedia, «Peso muerto,» 2 Mayo 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Peso_muerto#Beneficios_del_peso_muerto. [Último acceso: Junio 2022].

[37] G. Gottau, «Guía para principiantes (XV): Elevaciones frontales alternas con mancuernas,» 25 Mayo 2011. [En línea]. Available: https://www.vitonica.com/musculacion/guia-para-principiantes-xv-elevaciones-frontales-alternas-con-mancuernas. [Último acceso: Junio 2022].