UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Desarrollo de una aplicación de Realidad Aumentada con OpenCV

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Pablo López Hernández

Tutor: Manuel Agustí Melchor

Curso 2017-2018

Resumen

La realidad aumentada es un término que caracteriza a algunas de las aplicaciones actuales y cuyo uso podría mejorar y facilitar la vida de las personas en un futuro próximo. Las perspectivas de mercado pronostican que este tipo de aplicaciones nos acompañarán en nuestro día a día. Dado este incremento de presencia y su aplicabilidad en diversos ámbitos, se ha decidido realizar una aplicación que haga uso de ella, permitiéndonos experimentar cómo es el desarrollo de aplicaciones de esta naturaleza.

Este trabajo aborda la realización de este tipo de aplicaciones utilizando la librería de funciones OpenCV, que implementa técnicas de visión por computador. Con este objetivo, se decidió realizar una aplicación para dispositivos móviles Android, que permitiera al usuario realizar mediciones en tiempo real. Para ello, se emplea la librería de realidad aumentada ArUco, que está integrada como módulo extra en OpenCV y permite la estimación de posición de marcas blancas y negras.

Palabras clave: Realidad aumentada, OpenCV, ArUco, Android, medición.

Abstract

Augmented reality is a term that characterizes current applications and whose use could improve and facilitate the lives of people in the near future. Market prospects predict that this type of applications will be used in our daily activities. For this reason and for its applicability in various fields, we have been decided to make an application that uses this technology, allowing us to experience how is the development of applications of this nature.

The purpose of this project has been implementing this type of applications using OpenCV, which is the function library, which implements computer vision techniques. For this purpose, it was decided to do make an application for Android mobile devices, which would allow the user to make measurements in real time. For this, use ArUco library, which is augmented reality library that is integrated as an extra module in OpenCV, allowing us to estimate the position of black and white marks.

Keywords: Augmented reality, OpenCV, ArUco, Android, measure.

Resum

La realitat augmentada és un terme que caracteritza a algunes de les aplicacions actuals i l'ús del qual podria millorar i facilitar la vida de les persones en un futur pròxim. Les perspectives de mercat pronostiquen que este tipus d'aplicacions ens acompanyaran en el nostre dia a dia. A causa d'aquest increment de presència i la seua aplicabilitat en diverses àmbits, s'ha decidit realitzar una aplicació que empra esta tecnologia, permetent-nos experimentar com és el desenrotllament d'aplicacions d'aquesta naturalesa.

Este treball aborda la realització d'una aplicació de realitat augmentada utilitzant la llibreria de funcions OpenCV, que implementa tècniques de visió per computador. Amb este objectiu, es va decidir realitzar una aplicació per a dispositius mòbils Android, que permetera a l'usuari realitzar mesuraments en temps real. Per a això, s'empra la llibreria de realitat augmentada ArUco, que està integrada com a mòdul extra en OpenCV i permet l'estimació de posició de marques blanques i negres.

Paraules clau: Realitat augmentada, OpenCV, ArUco, Android, mesurament

Tabla de contenidos

Índice de figuras

Índice de tablas y cuadros

1. Introducción

El presente trabajo permite adquirir una mejor comprensión sobre la realidad aumentada, además de mostrar qué herramientas y procesos se necesitan para desarrollar una aplicación que haga uso de este concepto, mediante la librería de visión por computador OpenCV. Para ello, se expondrán y valorarán las opciones para implementar una aplicación de realidad aumentada, justificando la elección de ArUco como facilitador de la interacción entre usuario y aplicación. Así como también el enfoque para aplicaciones móviles, por su versatilidad para el usuario.

1.1 Motivación

La realidad aumentada es un término que caracteriza a algunas de las aplicaciones actuales, y cuyo uso podría mejorar y facilitar la vida de las personas en un futuro próximo. El uso de este enfoque ha ido incrementándose cada vez más y tan solo se ha visto una parte de su potencial. A pesar de esto, ya se ha observado su utilidad en diversos ámbitos profesionales, educativos y de ocio, permitiéndonos apreciar un futuro en el cual este tipo de aplicaciones nos acompañe en nuestro día a día.

Sin embargo, su potencial se extiende mucho más allá del espacio del consumidor. Las empresas y el sector público aportan una gran inversión en ella. En [14] se nos indica que en una encuesta reciente de los EE. UU, a ejecutivos de negocios y Tecnologías de la Información, la friolera de 77% de los encuestados dijeron que su compañía ya estaba haciendo uso de ella. De este porcentaje, alrededor del 36% dijo que estaban en las primeras etapas de pruebas, el 15% se encontraba en la etapa piloto, el 17% en las primeras implementaciones, y aproximadamente 9% se encontraban en despliegues de la última etapa. En otras palabras, si una empresa opera dentro de una industria que tiene indicadores competitivos clave que se basan en servicios, transferencia de conocimiento y capacitación, ventas y *marketing* o fabricación, y no está mirando de cerca la realidad aumentada, se está quedando atrás. En la Figura 1 podemos apreciar el gasto por parte de las empresas y una estimación de su crecimiento, que se espera en los próximos años.

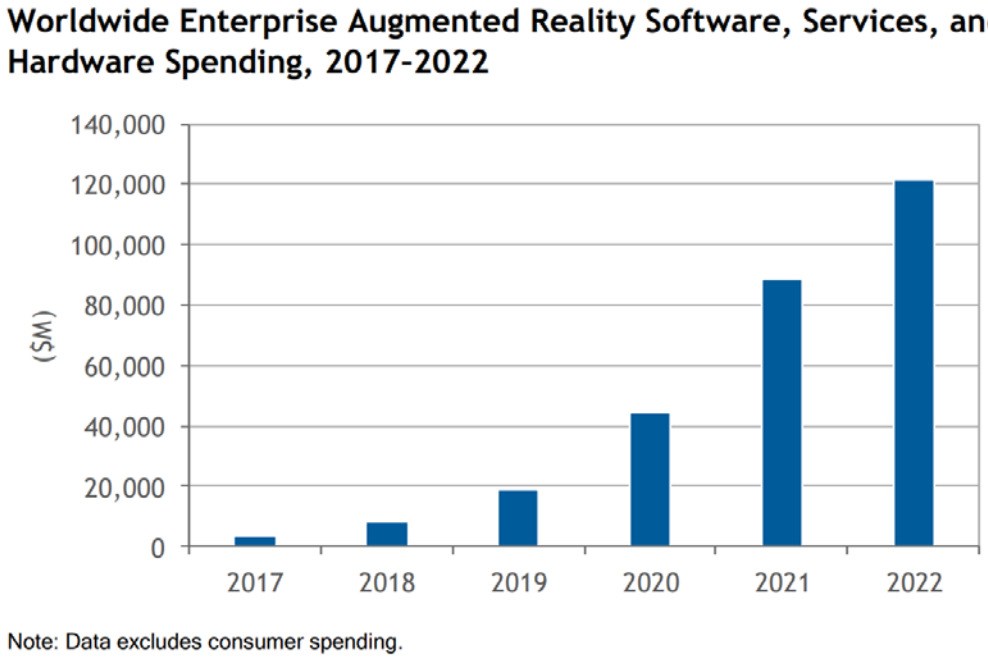

Figura 1: Gasto en *software*, servicios y *hardware* para realidad aumentada, realizada por empresas a nivel mundial. Extraída de [14].

Debido a este incremento de presencia y su aplicabilidad en diversos ámbitos, se decidió realizar una aplicación que empleara esta tecnología, permitiéndonos experimentar cómo es el desarrollo de aplicaciones de esta índole.

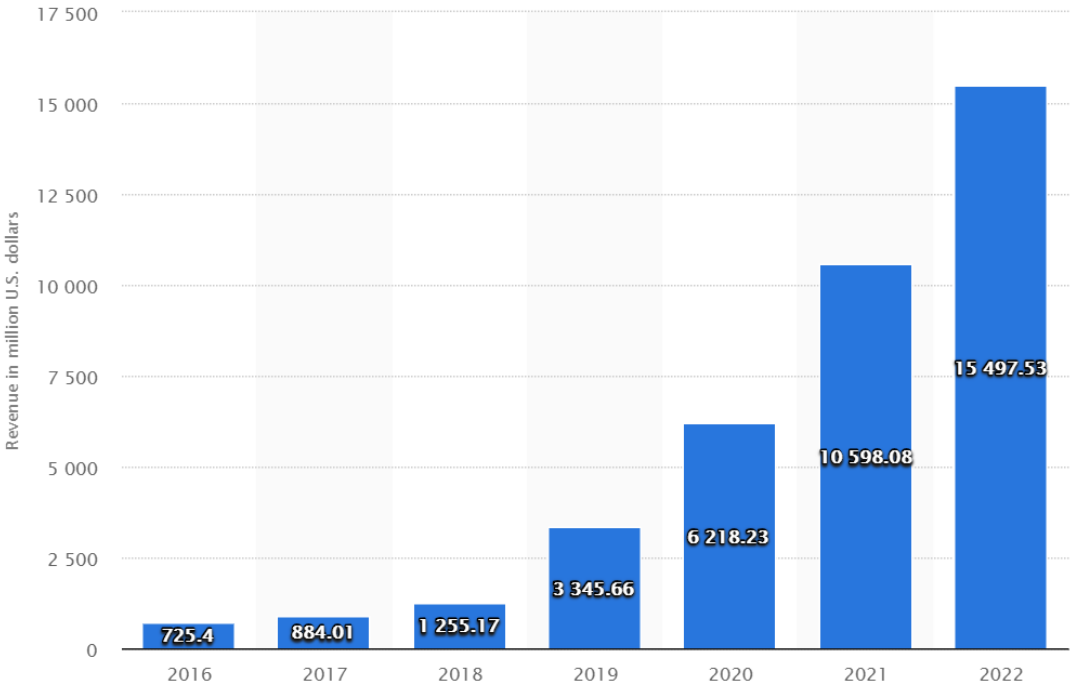

Figura 2: Ingresos globales del mercado de aplicaciones móviles de realidad aumentada para el consumidor. Extraída de [9].

Por otro lado, se optó por Android como plataforma. La causa principal es la enorme presencia de estos dispositivos en la sociedad y su aportación de utilidad a la aplicación, ya que nos permite hacer uso de ella en cualquier momento y de una manera más cómoda que un portátil. Con la Figura 2 podemos apreciar los ingresos globales del mercado de aplicaciones de realidad aumentada para estos dispositivos, está expresada en millones de dólares y permite vislumbrar un atisbo de la aportación económica que pueden llegar a ofrecer esta tecnología en mundo de los dispositivos portátiles.

Por último, la decisión de la temática de mediciones fue condicionada por el deseo de realizar una aplicación que tuviera una utilidad práctica para el usuario. Además, de permitir comparar con otras aplicaciones que no utilicen el enfoque de realidad aumentada para realizar el mismo objetivo.

1.2 Objetivos

El objetivo general del proyecto es el desarrollo de una aplicación de realidad aumentada para dispositivos móviles, empleando OpenCV. En particular, establecimos una serie de objetivos más específicos:

- El proyecto debe describir claramente que es una aplicación de realidad aumentada.

- Se caracterizará diversas herramientas que se pueden emplear en el desarrollo de la aplicación, de la cual se empleará la que ofrezca un mejor compromiso.

- La aplicación deberá funcionar en un gran número de dispositivos, concretamente en dispositivos Android, que disponga de una versión igual o superior a la 5.0.

- Debe permitir realizar diferentes tipos de medición, como superficie, distancia y perímetro. Además de permitir realizar comprobaciones de alineamiento o paralelismo en tiempo real.

- Debe posibilitar la interacción con el usuario, minimizando los requerimientos hardware necesarios para ello. De manera que pueda ser llevado al mayor número posible de dispositivos y que no precise el uso de otros.

1.3 Estructura

En el capítulo Estado del arte, trataremos el concepto de realidad aumentada, diferenciándolo de otros tipos de realidades con las cuales se suele confundir, expondremos los diferentes tipos en los que se puede clasificar, así como las diferentes herramientas que se pueden emplear para su desarrollo. Además de comentar aplicaciones actuales, con la misma temática que la de nuestro proyecto.

Posteriormente, en Análisis del problema, comentaremos las causas de las decisiones tomadas para el desarrollo del proyecto y los requisitos que establecimos.

En diseño de la solución se tratará temas relacionados con las actividades realizadas con anterioridad a la fase de programación del proyecto. Comentando las tecnologías que se emplearán en el proyecto y el diseño que se pensará para la aplicación. Además de mostrar una fase previa, donde se interactuó por primera vez con la librería OpenCV y ArUco, para comprender mejor los límites que tendría la aplicación y cómo afrontar el desarrollo de esta.

Tras ello, en el capítulo de desarrollo de la solución, se comentará los problemas con los cuales nos encontramos durante el desarrollo y las decisiones que se tomaron para afrontar estos problemas. También se tratará algunos aspectos técnicos de la librería, así como fragmentos de código de algunas de las funcionalidades desarrolladas.

Para finalizar, en el capítulo de Conclusiones, analizaremos si se han cumplido los objetivos propuestos en los capítulos anteriores, destacaremos los conocimientos adquiridos durante el desarrollo del proyecto, propondremos mejoras que se podrían llevar a cabo y realizaremos una valoración personal del trabajo realizado.

Además, se realizará un anexo que contendrá una guía de instalación de las herramientas que se necesitaron en el desarrollo del proyecto, que será de utilidad en caso de que alguien desee utilizar estas herramientas en el desarrollo de aplicaciones Android. En este anexo también se añadirá una guía de usuario, para que se comprenda mejor como emplear la aplicación.

Por último, se añadirá un glosario con la terminología y abreviaturas que se hayan empleado en la redacción de este documento.

2. Estado del arte

Para tener una visión más clara sobre qué se considera realidad aumentada (RA) hay que diferenciarla de la realidad mixta (RM) y realidad virtual (RV).

Como se muestra en [6] el continuo de la virtualidad es un concepto surgido en 1994, por Paul Milgram y Fumio Kishino, que sirve para describir que existe una escala continua que oscila entre lo que se puede definir como completamente virtual y lo que es completamente real. En la Figura 3 se puede observar un esquema de este concepto, de izquierda a derecha va aumentando el grado de estímulos generados por ordenadores.

Figura 3: Esquema del continuo de la virtualidad. Fragmento de imagen extraído de [10].

Tal y como se indica en la introducción de [10], la RA se encarga de estudiar las técnicas que permiten integrar en tiempo real contenido digital con el mundo real, teniendo la posibilidad de interactuar con los objetos físicos. El principal problema con el que deben tratar los sistemas de RA es el denominado registro, que consiste en calcular la posición relativa de la cámara real respecto de la escena para poder generar imágenes virtuales correctamente alineadas con esa imagen real. En la actualidad, los sistemas de RA suelen emplear combinaciones de diversas tecnologías: cámaras digitales, acelerómetros, sensores ópticos, giroscopios, GPS, hardware de procesamiento de sonido, etc. Se requiere de una unidad CPU potente y gran cantidad de memoria RAM para procesar las imágenes. La combinación de todos estos elementos se encuentra en los *smartphones* modernos, la cual al ser una plataforma de fácil transporte, se convierten en uno de los destinatarios más adecuados para el empleo de esta tecnología.

Un ejemplo realizado con esta tecnología sería la aplicación Inkhunter[1], una aplicación gratuita de dispositivos móviles que nos permite comprobar cómo nos quedaría un tatuaje, proyectando el diseño del tatuaje sobre una marca que realizamos en nuestra piel. En la Figura 4 podemos apreciar el uso de esta aplicación, en la parte izquierda la visión real y en la derecha la proyección que realiza la aplicación sobre la marca.

[1] http://inkhunter.tattoo/

Figura 4: Ejemplo de realidad aumentada, Inkhunter.

Por otro lado, La RV consiste en un entorno virtual generado por ordenador, que ocasiona en el usuario la sensación de estar inmerso en él, interactuando con los elementos virtuales que lo forman. Esta tecnología nos posibilita desarrollar aplicaciones o juegos similares a Knockout League[2], el cual te permite ponerte en la piel de un boxeador novato que se enfrenta a una serie de pintorescos contrincantes, para su uso se necesita poseer una de las diferentes gafas de RV del mercado, como PlayStation VR (también conocidas como PSVR). En la Figura 5 se puede observar el casco PSVR, en la parte izquierda, y lo que está visualizando el usuario, en la parte de la derecha.

Figura 5: Ejemplo de realidad virtual, Knockout League. Imagen compuesta por imágenes de la tienda de PlayStation.

[2] http://www.playstation.com/en-us/games/knockout-league-ps4/

Por último, la RM o realidad híbrida, es aquella que combina los dos tipos de realidades anteriores. Esta combinación permite al usuario interactuar tanto con objetos reales como virtuales, teniendo la posibilidad de emplear su cuerpo y los controladores remotos para interactuar con el entorno.

Un ejemplo de aplicación realizado con esta tecnología sería Microsoft Layout[3], que permite la creación de modelos tridimensionales en tamaño real con los que se puede interactuar, para su empleo es necesario el visor de RM de Microsft, conocido como HoloLens. Este tipo de gafas permiten experimentar tanto la experiencia de RM, como RA y RV. En la Figura 6 se puede observar el visor, a la izquierda, y un ejemplo de una persona usando Microsoft Layout, obviamente no podríamos visualizar los elementos virtuales sin algún dispositivo con dicho fin.

Figura 6: Ejemplo de realidad mixta, Microsoft Layout. Imagen compuesta por imágenes de la tienda de Microsoft.

Cabe mencionar que la RA es la más económica de ellas, siendo la RM más novedosa y con mayor coste. En la Figura 7 se puede observar una simplificación de lo mencionado en los anteriores párrafos, indicando que la RV es una completa inmersión en un entorno completamente virtual, la RA añade información virtual al mundo real y la RM combina ambas realidades, permitiendo interactuar tanto con el mundo real como con el entorno virtual. Además, permite visualizar una representación de cada una de ellas mediante dibujos, con el objetivo de facilitar su compresión.

[3] http://www.microsoft.com/es-es/p/microsoft-layout-preview/9nsjn53k3gfj

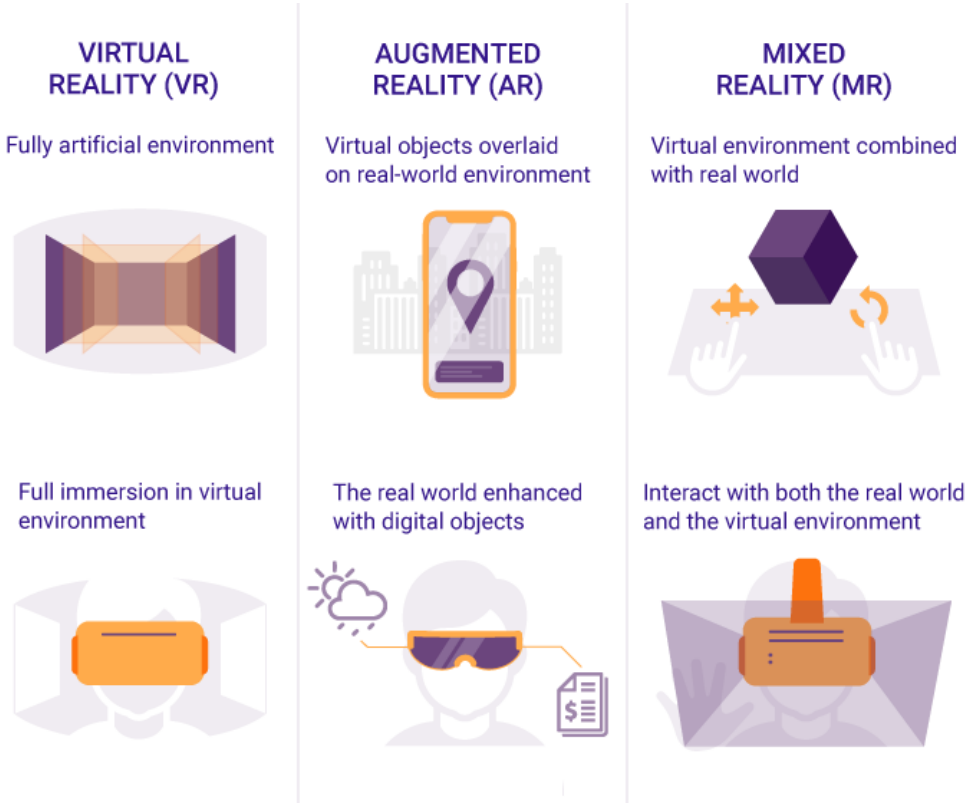

Figura 7: Diferencia entre tipos de realidades. Extraída de [8].

2.1 Tipos de realidad aumentada

La RA se puede clasificar en diferentes tipos, según [12] se puede realizar una división dependiendo de la manera en la que el contenido es integrado en la experiencia, distinguiendo cuatro tipos: con marcadores, a través de objetos tangibles, por geolocalización y centrada en el entorno.

La RA basada en marcadores consiste en símbolos impresos en papel o imágenes, sobre las cuáles se proyectan los elementos virtuales. La aplicación es capaz de identificar estos símbolos y determinar qué operaciones ejecutar, véase la Figura 8a para observar un ejemplo de ello. El contenido virtual puede desaparecer al dejar de apuntar al marcador, para su correcto funcionamiento es necesario que el marcador se encuentre en una superficie plana y que el dispositivo mantenga una distancia adecuada. En algunas aplicaciones se emplea los marcadores para activar la experiencia y el elemento virtual se mantiene en pantalla, aunque el dispositivo cambie de posición. Los códigos QR también entran en este grupo, aunque en su caso la operación a realizar puede estar codificada en el propio símbolo, permitiendo que cualquier lector QR sea capaz de llevar a cabo la operación definida en el código.

En cambio, la RA a través de objetos tangibles utiliza formas físicas para activar y mostrar la información. Necesita mayor potencia de cálculo para procesar los

elementos de la imagen y determinar si es el objeto con el cual tiene que interactuar. Es la más atípica de los cuatro tipos. Véase la Figura 8b para observar un ejemplo de este tipo.

Por otro lado, la RA por geolocalización emplea la información ofrecida por el GPS y otras herramientas, como el giroscopio y el acelerómetro, para localizar y superponer información sobre puntos de interés, en la Figura 8c se puede visualizar como sería el empleo de este tipo.

Por último, la RA centrada en el entorno, es el tipo más novedoso y consiste en calcular el lugar exacto en el que se encuentra el dispositivo respecto al conjunto de elementos que componen el espacio, permitiendo realizar funciones que antes eran inaccesibles, ya que permite desplazarse mientras el elemento virtual permanece en la posición, dando la posibilidad de aproximarte, distanciarte y rotar alrededor de él. En la Figura 8d se puede observar un ejemplo de cómo el objeto se proyecta sobre la superficie.

Figura 8: Ejemplos diferentes tipos de RA. Compuesta con imágenes de [20] [12] [24] [4].

2.2 Herramientas de desarrollo

Existe diversas herramientas de desarrollo para aplicaciones de RA, por ello vemos necesario exponer algunas de las diferentes opciones, para ser conscientes de las diversas alternativas en el desarrollo de aplicaciones de esta índole. Las principales herramientas son: Vuforia, Artoolkit, ArUco, ARCore y ARKit.

Vuforia es un SDK que permite construir aplicaciones de RA, que fue desarrollado por Qualcomm y posteriormente adquirido por PTC Inc. Es compatible con iOS, Android, Unity y UWP. Principalmente está orientada al desarrollo de aplicaciones para dispositivos móviles, pudiendo emplear Java o C++, y se suele recomendar emplearlo junto con Unity, debido a que simplifica en gran medida el desarrollo. Como se indica en [15] Vuforia utiliza la pantalla de la cámara combinada con datos del acelerómetro y del giroscopio para examinar el mundo. Usa la visión por computador para procesar los datos captados por la cámara, permitiendo al sistema ubicarse en la escena, obteniendo sus coordenadas y creando un modelo del entorno.

Además, permite múltiples opciones de desarrollo: reconocimiento de objetos, reconocimiento de objetos simples y complejos, reconocimiento de textos, reconocimiento del terreno (*Smart Terrain*), empleo de coordenada, etc. Existen diversas licencias[4], cada una con sus propias limitaciones y precios. También existe una licencia de desarrollo gratuita. Cabe mencionar que, en estos últimos meses, se ha desarrollado una nueva versión de Vuforia, concretamente la 7.2, que permite trabajar con ArCore. Esto se puede corroborar desde los propios foros de desarrollo de Vuforia[5]. En la Figura 9 podemos observar Arloon anatomy, aplicación con e fines educativos, sobre todo en materias de ciencias, que fue desarrollada con esta herramienta.

Figura 9: Ejemplo de aplicación realizada con Vuforia. Extraída de [17].

[4] https://developer.vuforia.com/vui/pricing
[5] https://developer.vuforia.com/forum/news-and-announcements/vuforia-72-available

En cambio, como se indica en el capítulo 3 de [10], Artoolkit es una librería de funciones para el desarrollo rápido de aplicaciones de RA. Fue escrita originalmente en C por H. Kato, y mantenida por el HIT Lab de la Universidad de Washington, y el HIT Lab NZ de la Universidad de Canterbury. Artoolkit facilita el problema del registro de la cámara empleando los métodos de visión por computador, de forma que obtiene el posicionamiento relativo de seis grados de libertad haciendo el seguimiento de marcadores cuadrados en tiempo real, incluso en dispositivos de baja capacidad de cómputo. Sus principales características son: permite *tracking* de cámara, empleo de marcas negras cuadradas, Código abierto, es rápido y multiplataforma. Un ejemplo desarrollado con Artoolkit lo podemos observar en la Figura 10.

Figura 10: Ejemplo realizado con Artoolkit. Extraída de [13]

Por otro lado, ArUco, del cual podemos encontrar una gran cantidad de información en el documento [19]. En él se indica que ArUco es una librería de código abierto, escrita en C++ y que requiere de OpenCV, que se emplea en la detección de marcadores *fiduciales* cuadrados en imágenes. Además de aportar la posibilidad de estimar la posición de la cámara respecto al marcador, si se realiza el calibrado de la cámara. Aunque no aporta la posibilidad de proyectar modelados en tres dimensiones, siendo necesario para ello usar conjuntamente librerías como OpenGL o OGRE. Su funcionamiento se basa principalmente en distinguir el contorno de los objetos del fotograma, elegir una serie de candidatos a ser un marcador, los cuales se irán desechando dependiendo de sus características, e identificar sus patrones binarios, que tiene la funcionalidad de identificador, posibilitado realizar la acción deseada en caso de que cierto marcador esté presente en el fotograma o emplear las coordenadas de los marcadores identificados. También, permite emplear mapas de marcadores, que son superficies compuestos por diversos marcadores, reduciendo la posibilidad de que la cámara falle en su detección, debido a que la probabilidad de dejar de detectar todos simultáneamente es reducida. Sus principales características, tal y como se indica en [18], son: detección de marcas con una sencilla línea de código, detección de diferentes marcas (ArUco, Artoolkit, AprilTag, ARTag, Chilitags), mayor velocidad

que otras librerías de detección de marcadores, pocas dependencias, detección de mapas de marcadores, integración trivial con OpenGL y OGRE, multiplataforma y licencia BSD. En la Figura 11 podemos observar un ejemplo desarrollado con ArUco y OGRE.

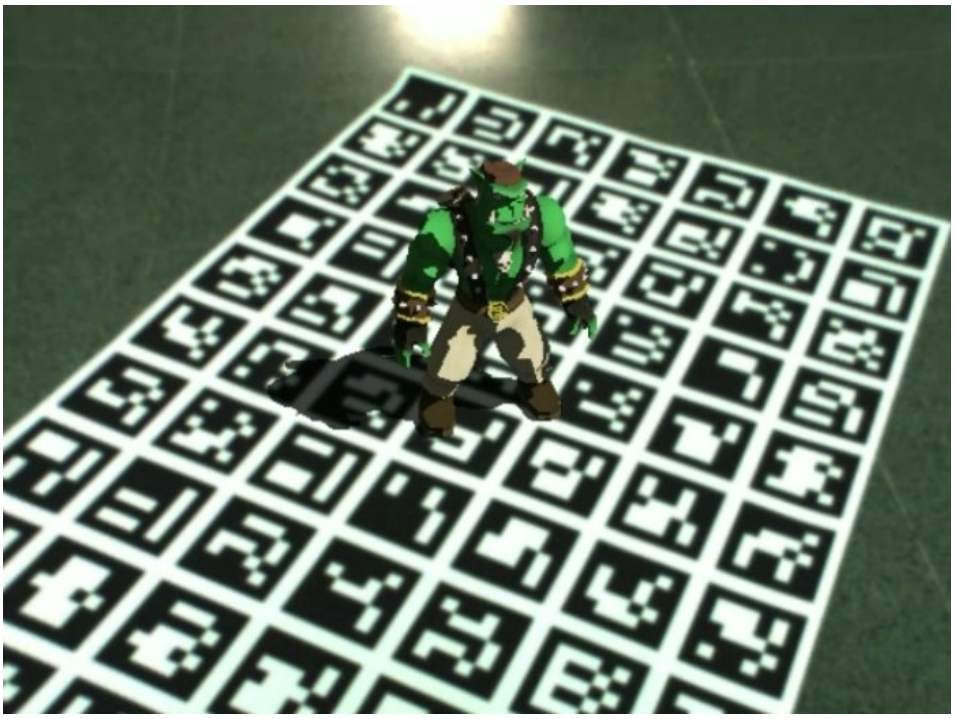

Figura 11: Ejemplo realizado con ArUco + OGRE. Extraída de [3].

También está disponible, ArKit, la opción desarrollada por Apple y destinada a los dispositivos iOS, fue lanzada en junio de 2017. Emplea una técnica llamada *Visual inertial odometry* (VIO), combina la información de la cámara con la de los sensores de movimientos, haciendo *tracking* del movimiento de nuestro dispositivo, procesando la información del acelerómetro y analizando los objetos que capta nuestra cámara. Dependiendo de si la tecnología a usar es de dos o tres dimensiones, usaremos SprintKit o SceneKit, respectivamente. Cuando empleemos tres dimensiones, tendremos la posibilidad de rotar el objeto virtual, pero con dos dimensiones los objetos siempre estarán mirando hacia nosotros. En la Figura 12 podemos observar la aplicación del cuento infantil “The Very Hungry Caterpillar”, donde los niños alimentarán a una oruga virtual hasta que se convierta en mariposa.

Por último, ArCore, la opción desarrollada por Google y destinada a los dispositivos Android, con el objetivo de competir con la RA de Apple, fue lanzada en mayo de 2018. Su funcionamiento se basa en Project Tango, una tecnología de RA, de la misma compañía, la cual necesitaba *hardware* específico y cuyo objetivo, como se indica en [21], era crear una herramienta portátil de visión por computador, que permitiera mapear espacios de tres dimensiones con un dispositivo móvil. Como se menciona en [23] ArCore permite crear experiencias de RA sin tener que programar desde cero el software de reconocimiento. A través de la cámara, sensores y algoritmos, el teléfono es capaz de localizar superficies,

detectar tu movimiento respecto a estas e incluso cuando una superficie está en diferentes condiciones de iluminación. No se limita solo a espacio de tres dimensiones, también puede trabajar con dos dimensiones. La ventaja que presenta frente a ArKit, es que la herramienta de Google es compatible tanto con Android como con IOS. En la Figura 13 podemos observar la aplicación Just a line, realizada con ARCore, que permite dibujar lo que deseemos en el aire.

Figura 12: Ejemplo aplicación realizada con ARKit. Extraída de [16].

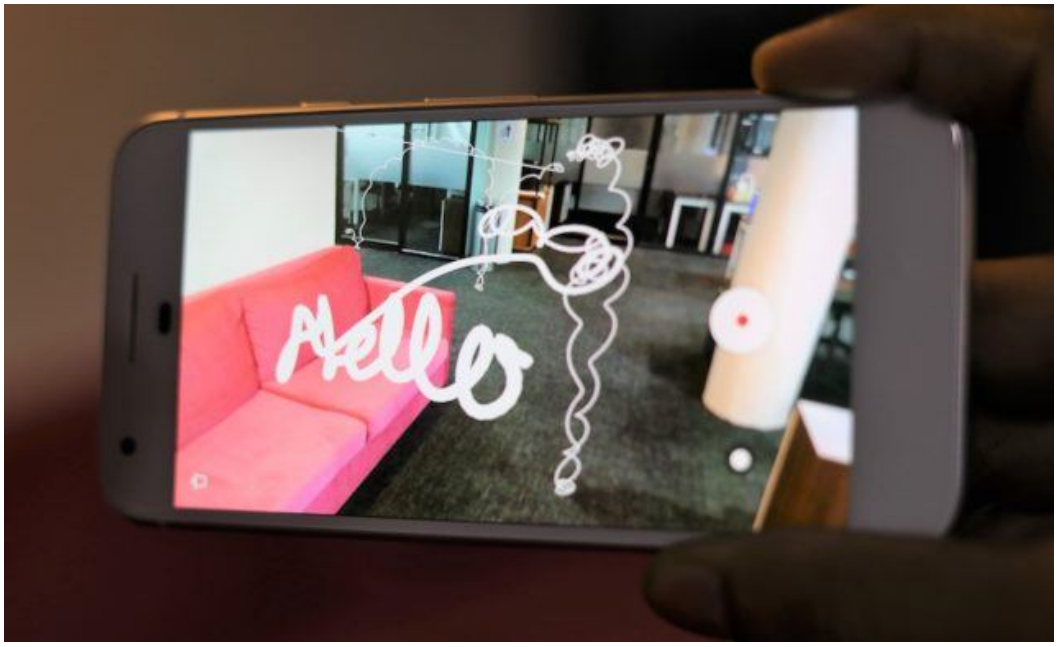

Figura 13: Ejemplo aplicación realizada con ARCore. Extraída de [22].

ArKit y ArCore, son las plataformas de desarrollo de aplicaciones de RA, para dispositivos móviles, más actuales y que aportan grandes novedades, en comparación al resto. Aunque cada una de ellas funciona de manera distinta, el resultado de ambas tecnologías es similar, se podría decir que son dos maneras diferentes de afrontar un mismo problema. Al ser los más recientes, no son compatibles con la mayoría de los dispositivos, hoy en día siendo empleados en los dispositivos de alta gama, esto se indica en diversos artículos y *blogs* como [2], donde también se mencionan diferentes dispositivos compatibles. En ambos casos el resultado es un modelo de alta precisión de la posición del dispositivo y su movimiento, permitiendo que los objetos virtuales se adapten al entorno.

En el Cuadro 1 podemos observar un resumen de las diferentes herramientas que se han mencionado para el desarrollo de aplicaciones RA.

Herramienta RA	Compañía	Licencia	Plataformas compatibles
Vuforia	PTC Inc.	Libre y Comercial	Android iOS Unity
Artoolkit	DAQRI	GNU GPL	Android iOS Windows Linux Mac OS X
ArUco	Aplicaciones de la visión artificial (AVA) - Universidad de Córdoba	BSD	Android iOS Windows Linux Mac OS X
ARKit	Apple	MIT 2.0	iOS 11 Unity Unreal
ARCore	Google	Apache 2.0	Android 7.0 iOS 11 Unity Unreal

Cuadro 1: Características de las herramientas de desarrollo de RA

Para finalizar este apartado hay que destacar que recientemente, este mes de junio, se ha anunciado ARKit 2, el cual aporta novedades interesantes respecto

ARKit, como la posibilidad de experiencias compartidas, es decir, realidad aumentada persistentes asociadas a una ubicación concreta, posibilitando que varios usuarios visualicen los mismos elementos virtuales, pero desde sus respectivos puntos de vista y puedan colaborar. Será compatible con dispositivos iOS 12 o superiores. Desde [1] podemos leer más sobre ello y visualizar algún video de ejemplo.

2.3 Aplicaciones similares

Si revisamos la *Store* de Android, comprobamos que hay algunas aplicaciones con la temática de medición como: RulAr[6], Measure[7], Prime Ruler[8], etc. De todas ellas podemos observar que, normalmente, las aplicaciones suelen afrontar la problemática de la toma de medidas de tres maneras:

- La primera opción consiste en convertir tu teléfono en una regla para realizar la medición con la pantalla de tu móvil, simulando la funcionalidad de una regla con el dispositivo.

- La segunda opción consiste en realizar la medición en imágenes, suelen pedir la medida real de algún objeto para poder deducir la medida del resto de objetos de la imagen.

- Por último, la tercera opción es calcular las distancias con RA, concretamente empleando tecnologías nativas de fabricantes como ARKit o ARCore. El problema de esta opción se debe a que, empleando estas recientes tecnologías, las aplicaciones no son compatibles con un gran número de dispositivos.

Actualmente las aplicaciones se suele emplear la tercera opción, este ese el caso de RulAr y Measure. También hay aplicaciones que tratan de combinar las tres opciones, como es el caso de Prime Ruler. Se puede observar un ejemplo de cada una de las opciones mencionadas en la Figura 14.

Figura 14: Opciones frecuentes en el mercado de aplicaciones de medición. Compuesta por imágenes de Google Play.

[6] https://play.google.com/store/apps/details?id=com.augmentedstartups.RulAR
[7] https://play.google.com/store/apps/details?id=com.google.tango.measure&hl=es
[8] https://play.google.com/store/apps/details?id=com.grymala.photoruler

La aplicación que se desarrollará realizará las mediciones mediante la RA, empleando ArUco, consiguiendo funcionar en un mayor número de dispositivos que la alternativa desarrollada en ARCore o ARKit. Además, nuestra aplicación aporta funcionalidad que no disponen las otras opciones, como determinar si dos objetos están en línea o son paralelos y cálculo de superficie.

3. Análisis del problema

El problema que este proyecto debe afrontar consiste en el desarrollo de una aplicación de RA empleado OpenCV. Con este propósito, nos plantemos una serie de decisiones que debíamos tomar, antes de comenzar el desarrollo, las cuales definen algunas características de la aplicación.

3.1 Decisiones iniciales

Para comenzar, nos planteamos sobre qué plataforma debíamos orientar la aplicación, llegando a la conclusión de que los dispositivos móviles eran la opción más adecuada, debido a que con la tecnología actual son capaces de soportar aplicaciones de RA y permiten más comodidad que la aportada por las computadoras, permitiendo que el usuario haga uso de la aplicación en cualquier situación, en vista de que posee mayor facilidad de transporte que cualquier portátil u ordenador de sobremesa. Entre los sistemas operativos decidimos céntranos en Android, debido a que disponíamos de mayor acceso a dispositivos con este sistema, facilitando la realización de las pruebas.

Posteriormente, investigamos sobres las diferentes herramientas que podían facilitarnos el desarrollo de este tipo de aplicaciones, las cuales comentamos en el capítulo anterior, y si era posible emplearlas juntamente con OpenCV. Tras deliberar las opciones, decidimos emplear la librería ArUco, ya que era una de las alternativas que ofrecía la posibilidad de desarrollar la aplicación para el mayor número de usuarios posibles y permitía la opción de desarrollar aplicaciones para Android. Además de proporcionar una instalación y programación más sencillas que el resto de las herramientas, debido a que tiene mayor compatibilidad con OpenCV, puesto que ha sido integrada como un módulo extra de esta.

En cuanto al lenguaje de programación, se recomendaba el uso de C o C++, puesto que la librería OpenCV y ArUco habían sido desarrolladas en estos lenguajes. Para emplear estos lenguajes en Android Studio, se debe realizar un desarrollo nativo, el cual mejora el rendimiento de la aplicación, pero no es recomendable si estás iniciándote en el desarrollo de aplicaciones Android. También existe la posibilidad de emplear Java, aunque para ello hay que realizar una instalación especial, una de las posibilidades es emplear las herramientas MinGW [9] y CMake [10], generando Java *wrappers*, que permitirían llamar a funciones de estas librerías mediante código Java, esta instalación fue la que lleve a cabo y aparece guiada en el Anexo. Tras deliberar las opciones, decidimos realizar el desarrollo en Java, debido a tener una mayor familiaridad con él y posibilitando un desarrollo más cómodo en Android Studio.

Tras decidir estos aspectos, realizamos un *brainstrom*, para plantear posibles ideas que sirvieran como objetivo de la aplicación. Una vez obtenida una lista de

[9] http://mingw.org/
[10] https://cmake.org/

suficiente extensión, debatimos cuál de ellas era la más realista y con mayor utilidad real. Finalmente, decidimos emplear la temática de mediciones, la idea base era una aplicación que fuera capaz de calcular la distancia entre varias marcas, posteriormente fuimos añadiendo más funcionalidad a esta base.

3.2 Requisitos

Una vez decididas las herramientas a utilizar y la temática de la aplicación, establecimos una serie de requisitos funcionales (RF) y no funcionales (RNF). Estos requisitos son plasmados en el Cuadro 2.

Tal y como se vio en la asignatura Análisis y especificación de requisitos, concretamente en el tema 2, Proceso de Ingeniería de Requisitos, los requisitos funcionales describen la funcionalidad o los servicios que se espera que el sistema proveerá. En cambio, los requisitos no funcionales se refieren a las propiedades emergentes (atributos) del sistema como la fiabilidad, el tiempo de respuesta, la capacidad de almacenamiento, la capacidad de los dispositivos de entrada/salida, ajuste a estándares, etc.

RF	RNF
La aplicación debe realizar las mediciones (distancia, perímetro y superficie) en tiempo real.	La aplicación debe funcionar en dispositivos Android, con versión de sistema operativo igual o superior a la 5.0.
Debe posibilitar la descargar de los marcadores que se usarán en la aplicación.	Se aportará un correo de contacto, para dudas o incidencias.
Será capaz de funcionar en español, inglés y francés.	No ocupará más de 170 MB
La aplicación deberá permitir realizar recalibrados, una vez hecho el calibrado inicial.	Debe de ser desarrollada usando OpenCv + ArUco
Deberá permitir seleccionar diversas resoluciones.	El usuario solo tendrá acceso a la aplicación si se concede los permisos indicados.
Dispondrá de diversos ajustes, con los cuales interactuará el usuario.	Debe proporcionar información, que facilite al usuario el uso de los marcadores ArUco.
La aplicación debe ser capaz de realizar las comprobaciones de alineamiento y paralelismo en tiempo real.	El tiempo de aprendizaje para dominar el uso de la aplicación, por un usuario, debe de ser menor a una hora.
El usuario tendrá la opción de ajustar el margen de decimales.	La fiabilidad de la aplicación no se asegura que sean del 100%, debido a pequeños errores en la detección de los marcadores.

Cuadro 2: Requisitos funcionales y no funcionales

4. Diseño de la solución

En este apartado trataremos las actividades que realizamos previamente al desarrollo de la aplicación, las cuales nos permitieron tener una visión más clara del proyecto y comprender algunos de los problemas que deberíamos solucionar durante su desarrollo.

4.1 Tecnología utilizada

Para el desarrollo del proyecto se han empleado diversas tecnologías y herramientas que facilitaron su realización. A continuación, procedemos a comentar cada una de ellas.

Se empleo Android Studio, que es el entorno de desarrollo integrado oficial para la plataforma Android, por lo cual proporciona todo lo necesario para el desarrollo de aplicaciones móviles de este sistema operativo, facilitando el desarrollo del proyecto.

Como indica el título del proyecto, empleamos OpenCV, que como se menciona en [11] es una librería de visión por computador y *machine learning*, de código abierto, desarrollada originalmente por Intel. La cual posee una gran cantidad de algoritmos, que permiten identificar objetos, caras, hacer *tracking* de movimiento de objetos, encontrar imágenes similares, ...

Además, empleamos ArUco, que es uno de los módulos extra con los cuales cuenta OpenCV. Como se ha mencionado ya a lo largo del proyecto, es una librería de código abierto, escrita en C++, para la estimación de posición de la cámara empleando marcadores cuadrados.

Para compilar la librería OpenCV con los módulos extra para Android, permitiendo el uso de Java, tuvimos que emplear las herramientas gratuitas CMake y MinGW. Tal y como se describe en Wikipedia, la primera de ellas es una herramienta de generación y automatización de código, la cual fue diseñada para construir, probar y empaquetar *software*. Se utiliza para controlar el proceso de compilación del *software* usando ficheros de configuración sencillos e independientes de la plataforma. Por otro lado, MinGW es un compilador que incluye un conjunto de la API de Win32, permitiendo un desarrollo de aplicaciones nativas para esa plataforma, pudiendo generar ejecutables y bibliotecas usando la API de Windows. Como en el anterior capitulo, la instalación con estas herramientas aparece guiada en el Anexo.

Para realizar copias de seguridad del proyecto empleamos Gitlab, que es un servicio de control de versiones basado en Git. Además, permite creación de wikis y posee un sistema de seguimiento de errores. A diferencia de Github este permite crear repositorios privados sin disponer de una cuenta premium.

También se hizo uso de Photoshop, el editor de gráficos rasterizados desarrollado por Adobe Systems Incorporated, para elaborar las diferentes imágenes e iconos de la aplicación. Además, para la elaboración de *Mockups* de la aplicación se hizo uso del programa Balsamiq Mockups 3.

4.2 Fase previa

Antes de comenzar el desarrollo de la aplicación llevamos a cabo una serie de ejercicios para familiarizarnos con las librerías de OpenCV y ArUco, comprobando su correcto funcionamiento tras la instalación y permitiéndonos hacernos una idea de cómo desarrollar la aplicación de mediciones. El resultado de estos ejercicios se puede observar en la Figura 15.

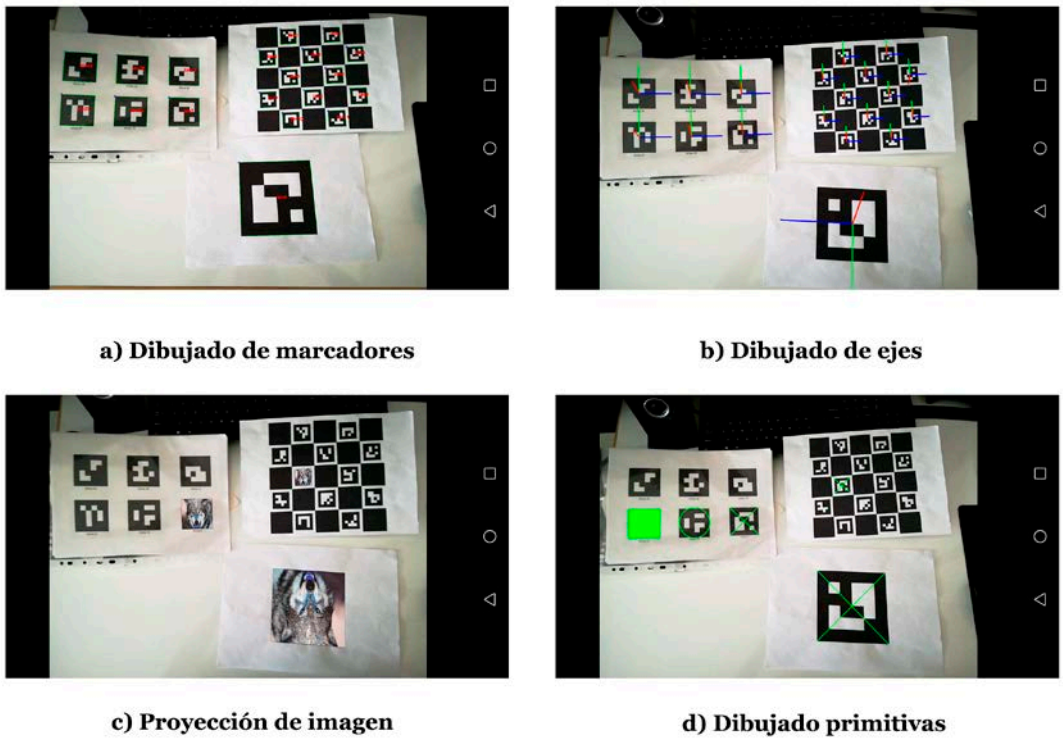

Figura 15: Ejercicios previos.

De estos ejercicios bastaría con realizar el primero, el correspondiente con el dibujado de marcadores, para comprobar que la librería OpenCV se ha instalado correctamente con ArUco. Para realizarlo, lo primero que deberíamos llevar a cabo es la inicialización de OpenCV. Hay diversas formas de realizarla, algún ejemplo puede ser visualizado desde la propia página de la librería. Concretamente nosotros realizamos una inicialización asíncrona, ejecutada en el método onResume[11] de la Activity y empleando BaseLoaderCallback de OpenCV, para recibir la respuesta de la inicialización:

[11] https://developer.android.com/training/basics/activity-lifecycle/pausing?hl=es-419

```
private BaseLoaderCallback mLoaderCallBack = new BaseLoaderCallback( AppContext: this) {
    //Método de callBack que se llamará después de la inicialización de openCV.
    @Override
    public void onManagerConnected(int status) {
        switch (status){
            case BaseLoaderCallback.SUCCESS:{
                break;
            }
            default:{
                super.onManagerConnected(status);
                break;
            }
        }
    }
};
@Override
protected void onResume(){
    super.onResume();
    if(!OpenCVLoader.initDebug()){
        Toast.makeText(getApplicationContext(), "Fallo en la inicialización de OpenCV. Puede que su disp...",Toast.LENGTH_SHORT).show();
        OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_4_0, AppContext: this,mLoaderCallBack);
    }
    else{
        mLoaderCallBack.onManagerConnected(LoaderCallbackInterface.SUCCESS);
    }
}
```

El Toast está únicamente para mostrar un mensaje, extraído del fichero string.xml, en el dispositivo que ejecuta la aplicación. Es innecesario para la inicialización, puede ser sustituirlo por una salida en consola.

Una vez inicializado, debemos proceder a detectar y dibujar los marcadores de ArUco, demostrando que la instalación ha sido un éxito y la aplicación es capaz de emplear ambas librerías conjuntamente. Para ello deberemos llamar al método de detectMarkers y drawDetectMarkers en cada fotograma, detectando y dibujando los marcadores captados. En nuestro caso, al emplear la cámara javaCameraView proporcionada por la versión 3.4.0 de OpenCV, bastaba con añadir la llamada a esas funciones desde el método onCameraFrame. Antes de realizar todo esto, inicializamos los componentes necesarios en el método onCreate de la Activity.

```
ids = new Mat();
corners = new ArrayList<>();

dictionary = Aruco.getPredefinedDictionary(Aruco.DICT_4X4_50);
javaCameraView = (JavaCameraView) findViewById(R.id.java_camera_view);
javaCameraView.setVisibility(SurfaceView.VISIBLE);
javaCameraView.setCvCameraViewListener(this);
```

```
@Override
public void onCameraViewStarted(int width, int height) {
    frame = new Mat(height,width, CvType.CV_8UC3);
}

@Override
public void onCameraViewStopped() {
    frame.release();
}

@Override
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
    //Convertimos, el frame que esta capturando la camara, de rgba(4 capas ) a rgb(3 capas).
    Imgproc.cvtColor(inputFrame.rgba(),frame,Imgproc.COLOR_RGBA2RGB);

    //Solo funcionan con frames de 1 o 3 capas.
    Aruco.detectMarkers(frame, dictionary, corners, ids);

    if(!ids.empty()) {
        Aruco.drawDetectedMarkers(frame, corners, ids, COLOR);
    }
    return frame;
}
```

Con dicho fragmento de código se conseguiría visualizar la misma escena que se observa en la Figura 15a. El código no es explicado, debido a que hicimos uso de estos ejercicios previos para el desarrollo de la aplicación final, por lo cual se comentarán en el capítulo de desarrollo de la solución.

Tras realizar los ejercicios, realizamos pruebas, para comprender mejor el funcionamiento de las librerías. Para su elaboración realizamos veinte intentos, en los cuales obteníamos el tiempo que tardaba en detectar un marcador, para ello calculábamos el tiempo transcurrido entre justo antes de realizar la llamada a detectMarkers y el momento en que la matriz de ids pasaba de estar vacía a tener un elemento. Debido a la ausencia de un trípode empleé la superficie más estable que encontré, colocando el dispositivo sobre él e intentando ubicar los marcadores en la misma ubicación. Para las pruebas utilice diferentes resoluciones (960 x 544 y 1920 x 1080) y diferentes tamaños de marcadores (3x3, 5x5 y 12x12). Los resultados obtenidos han sido plasmados en la Tabla 1, los resultados se muestran en ms.

	960 x 544			1929 x 1080		
	12x12	5x5	3x3	12x12	5x5	3x3
	21	17	15	118	121	133
	15	20	15	91	95	93
	15	19	15	84	83	85
	15	16	15	93	110	84
	15	16	16	99	113	94
	16	16	15	113	132	85
	17	17	15	133	130	82
	16	16	15	128	87	82
	17	16	16	89	91	87
	20	17	16	92	84	99
	32	16	16	111	84	80
	15	15	16	99	81	81
	14	15	17	95	92	81
	14	16	14	103	80	82
	15	15	14	110	81	83
	15	15	15	111	84	77
	15	15	15	100	81	81
	15	17	15	103	83	85
	15	15	16	96	91	85
	17	15	18	107	89	86
Media	16,7	16,2	15,45	103,75	94,6	87,35

Tabla 1: Resultado de las pruebas previas: tiempos promedio de respuesta.

De los resultados obtenidos dedujimos que analiza el fotograma píxel a píxel, por ello detecta antes marcadores pequeños que grandes, siempre y cuando sean suficientemente grandes como para que la cámara pueda visualizarlo correctamente, debido a que siempre obteníamos tiempos menores en los marcadores de 3x3 que en el resto. Además, a mayor resolución se obtienen más pixeles en el fotograma por lo que es capaz de detectar marcadores más pequeños y muestra una imagen más nítida, pero obtiene peores resultados en cuanto a velocidad, esto es debido a que posee una mayor cantidad de píxeles a analizar. Por esto último se nos ocurrió otorgar la posibilidad al usuario de seleccionar la resolución deseada, permitiéndole decidir qué característica prefiere, mayor calidad de imagen o rendimiento.

4.3 Diseño de la aplicación

En las primeras fases del desarrollo del proyecto elaboramos un mapa de interfaz de usuario, para visualizar mejor la navegabilidad entre las diversas ventanas que podrían formar la aplicación, permitiéndonos visualizar cómo se conectaría cada una de las funcionalidades que habíamos pensado. Esto se puede visualizar en la Figura 16.

Lo primero que se visualizaría al iniciar la aplicación sería una pantalla de bienvenida, desde la cual se comprobaría si se realizó el calibrado o no. Si no se detectan estos datos, se nos dirigiría a la ventana de calibrado. En caso de disponer ya de ellos o acabar de realizar el primer calibrado, se nos llevaría a la ventana principal, desde la cual se podría acceder al resto de ventanas de la aplicación, siendo esta el punto central.

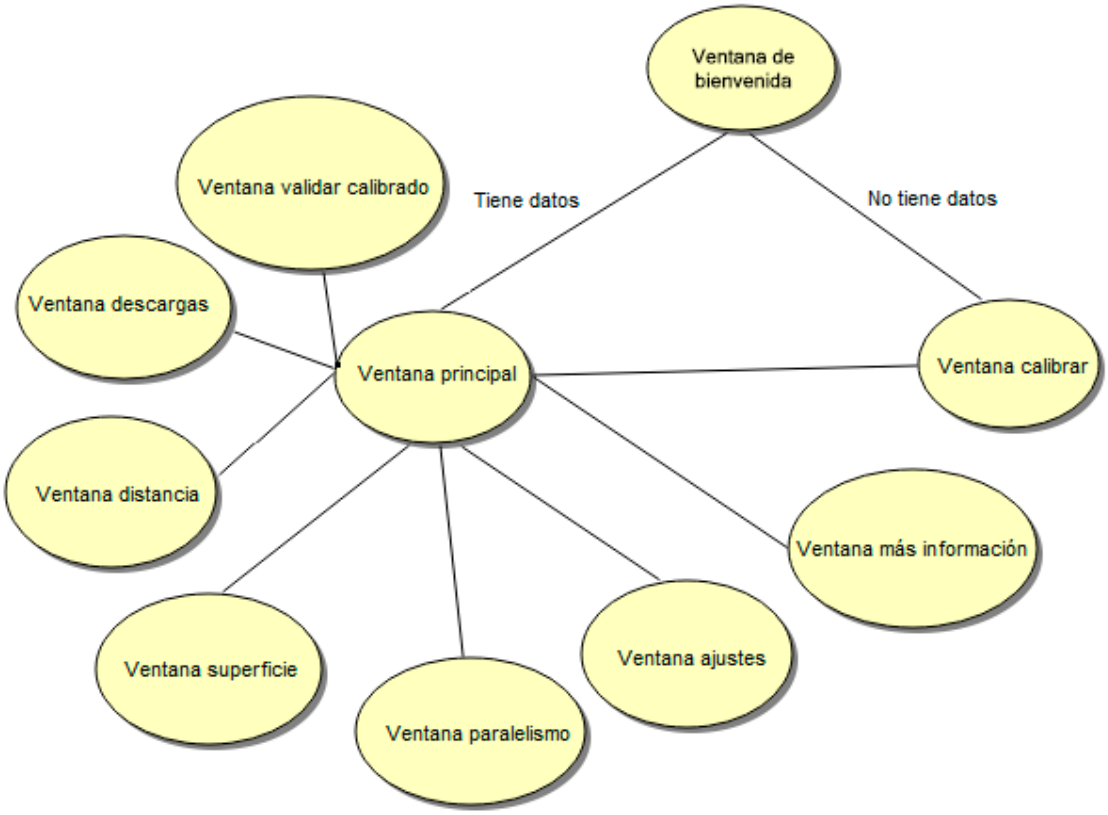

Figura 16: Mapa de interfaz gráfica de usuario.

Tras tener una idea de la posible navegación entre ventanas que habría en la aplicación, desarrollamos una serie de bocetos de cada una de ellas. Excepto el icono de la aplicación, el cual fue desarrollado en papel, todo el resto de los bocetos fueron desarrollados con la herramienta de creación de *mockups*, Balsamiq Mockups 3[12], la cual ya utilicé durante la carrera. En la Figura 17 podemos observar las alternativas que elaboramos para ser el icono de la aplicación, el cual también aparecería en la ventana de bienvenida o *Splash Screens*.

[12] https://balsamiq.com/download/

Figura 17: Bocetos icono aplicación.

A continuación, se muestran la serie de bocetos correspondientes a las ventanas de la aplicación. La primera, que se puede observar en la Figura 18 corresponde a la ventana principal de la aplicación. Esta ventana estará formada por dos páginas, las cuales contendrán las diferentes funcionalidades que permite la aplicación, posibilitando leer una pequeña descripción de cada una de ellas. Además, esta será el punto central de acceso a cualquiera del resto de ventanas.

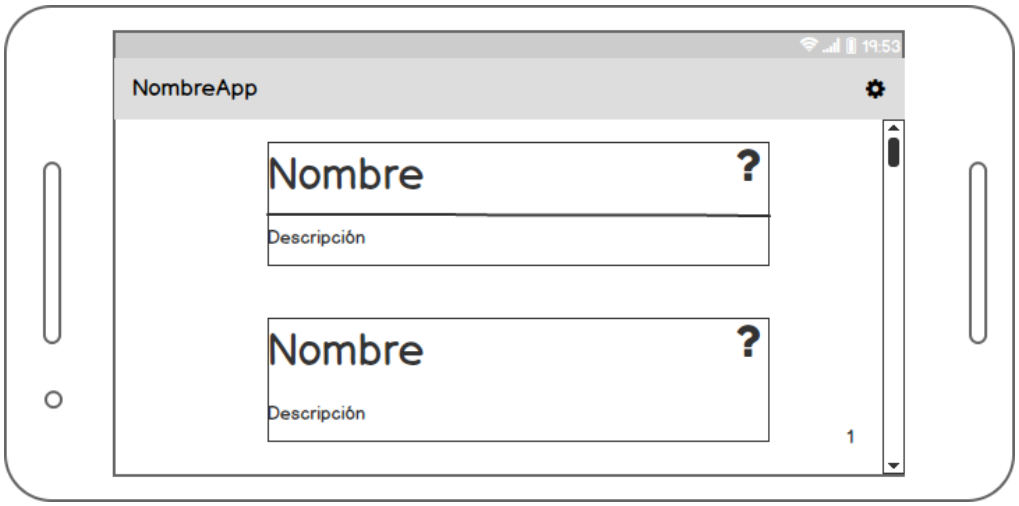

Figura 18: Boceto ventana principal.

Si pulsáramos sobre el botón de la interrogación, se montaría una ventana modal donde se visualizarían los marcadores que se necesitarían para desempeñar la función seleccionada. El boceto de esta ventana se puede apreciar en la Figura 19.

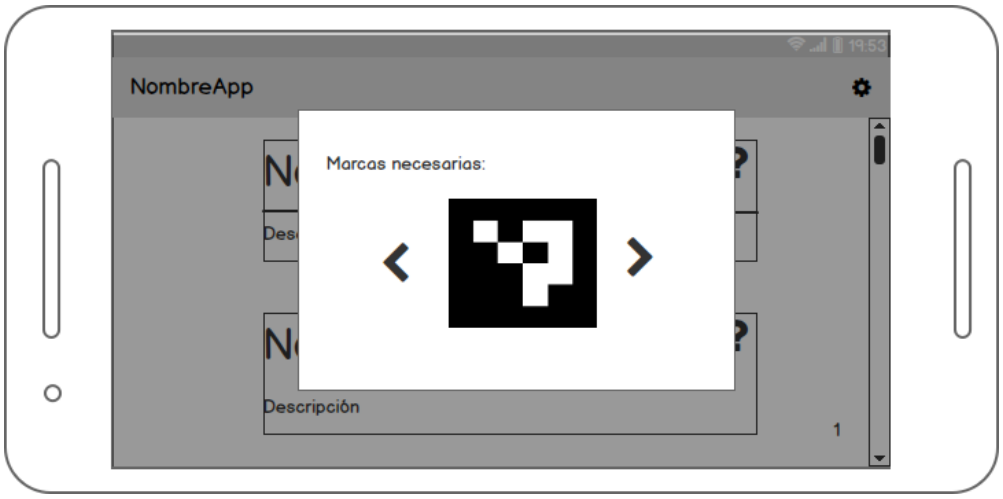

Figura 19: Boceto ventana más información.

En cambio, si pulsáramos sobre el engranaje superior, accederías a la ventana de ajustes. En esta ventana se configurarían ciertos parámetros de la aplicación, como la resolución o el idioma deseados por el usuario. En la Figura 20 podemos observar los bocetos de esta ventana.

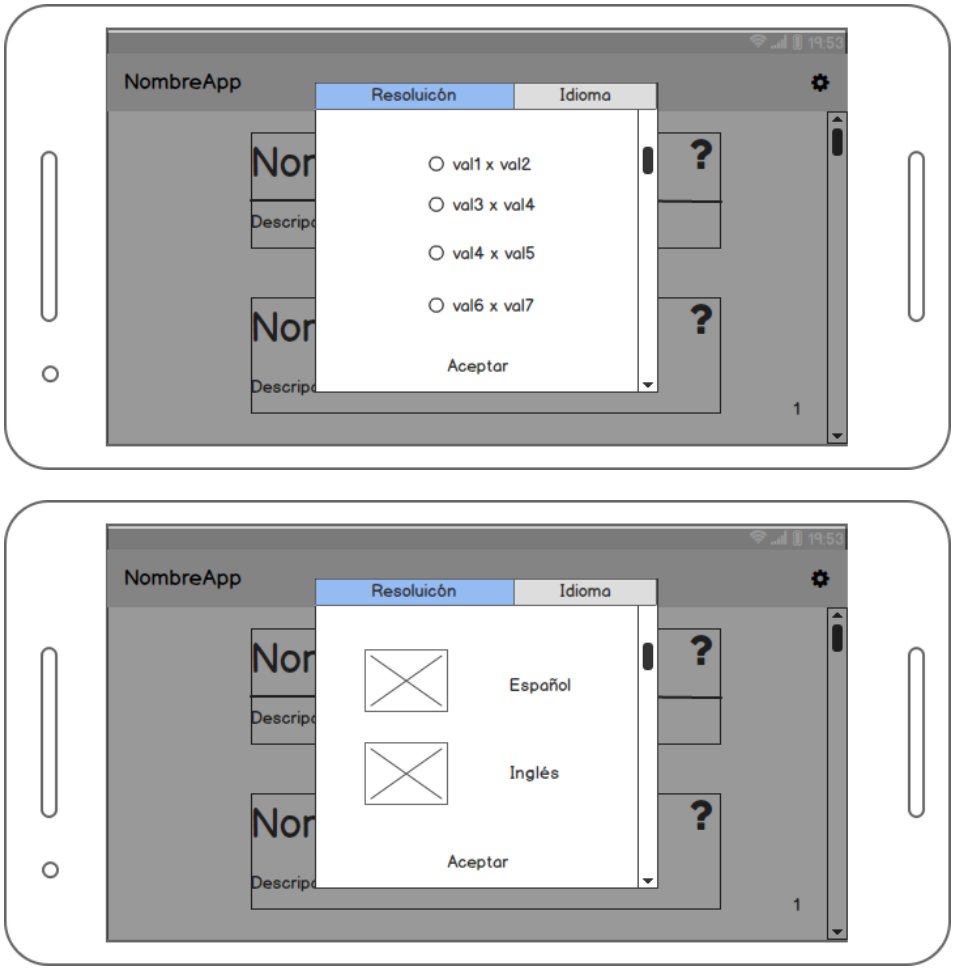

Figura 20: Boceto ventana ajustes.

Si pulsamos sobre la opción de descargas, se abrirá dicha ventana de forma modal, desde la cual el usuario podrá descargar todos los marcadores necesarios para el uso de las diversas funciones que proporciona la aplicación. Además de disponer de la opción de descargar un marcador en concreto, para ello se deberá indicar su identificador. En la Figura 21 podemos observar el boceto de esta ventana.

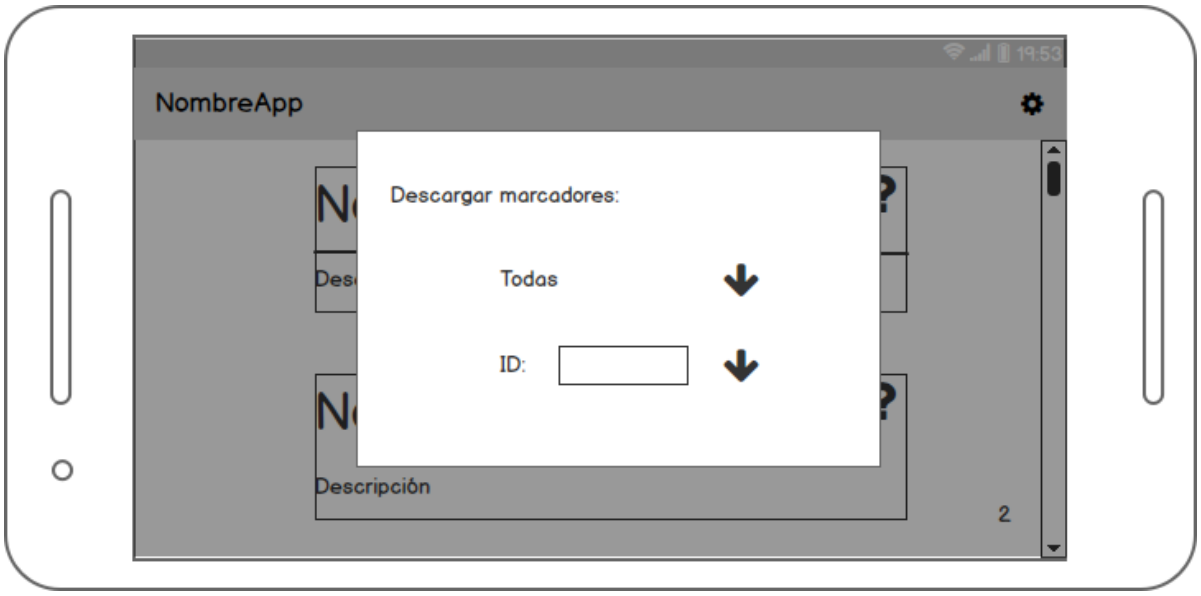

Figura 21: Boceto ventana descargas.

A la ventana de calibrado se accedería desde la ventana de bienvenida, siempre que se detecte que no hay datos, y desde la opción de recalibrado. En esta ventana realizaríamos capturas sobre un tablero de marcadores, hasta llegar al número estipulado de capturas, para averiguar los parámetros de la cámara. Tras realizar el calibrado, se mostraría el error obtenido de este, tanto mediante un texto como un código típico de colores (rojo, amarillo y verde). En la Figura 22 podemos observar el boceto de esta ventana, en ambas fases.

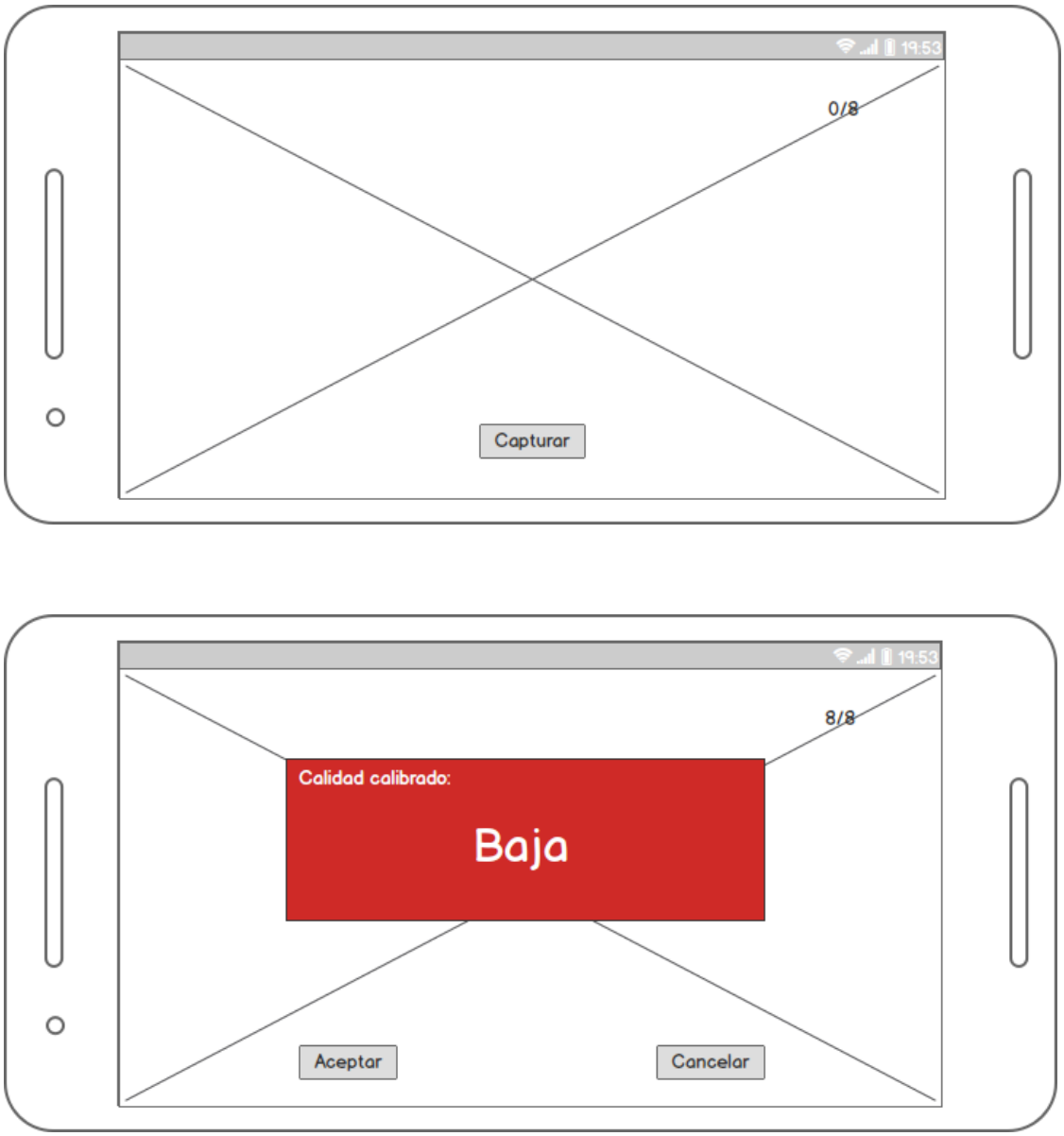

Figura 22: Boceto ventana calibrado.

Por otro lado, si se seleccionamos la opción de paralelismo accederemos a su respectiva ventana, donde a partir de cuatro marcadores se nos indicará si están en línea o paralelo, mediante un sonido y cambiando de color las líneas que conectan los marcadores. El boceto de esta ventana puede ser observado en la Figura 23.

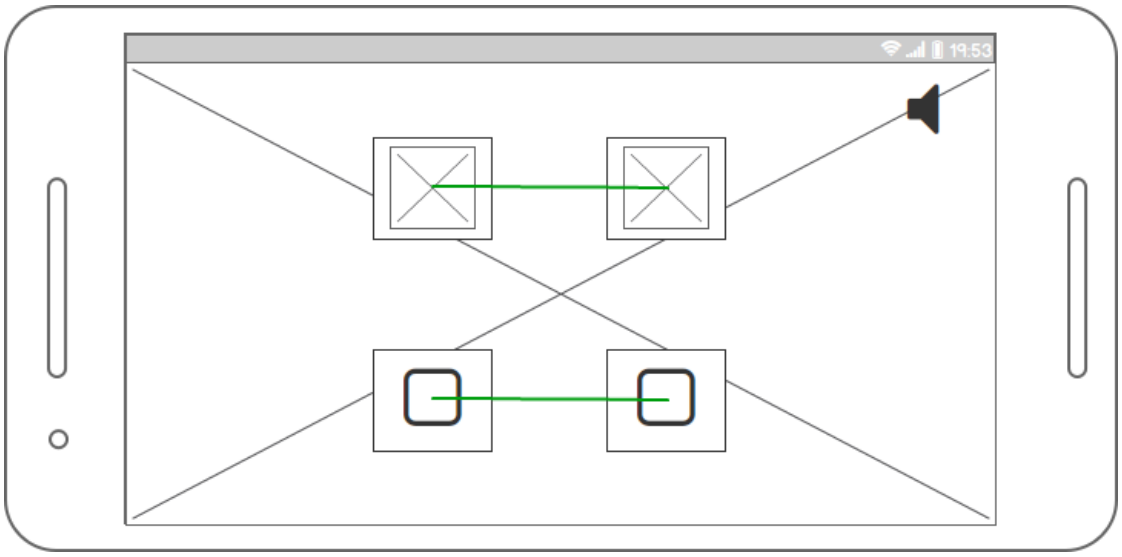

Figura 23: Boceto ventana paralelismo.

De igual modo, seleccionando sus respectivas opciones, se puede acceder desde la ventana principal a las ventanas de distancia y superficie. Estas ventanas nos permitirán observar la distancia y superficie comprendida entre los diferentes marcadores captadas en pantalla, mostrando los resultados en el medio de ellas. En la Figura 24 y 25 podemos observar los bocetos de ambas ventanas.

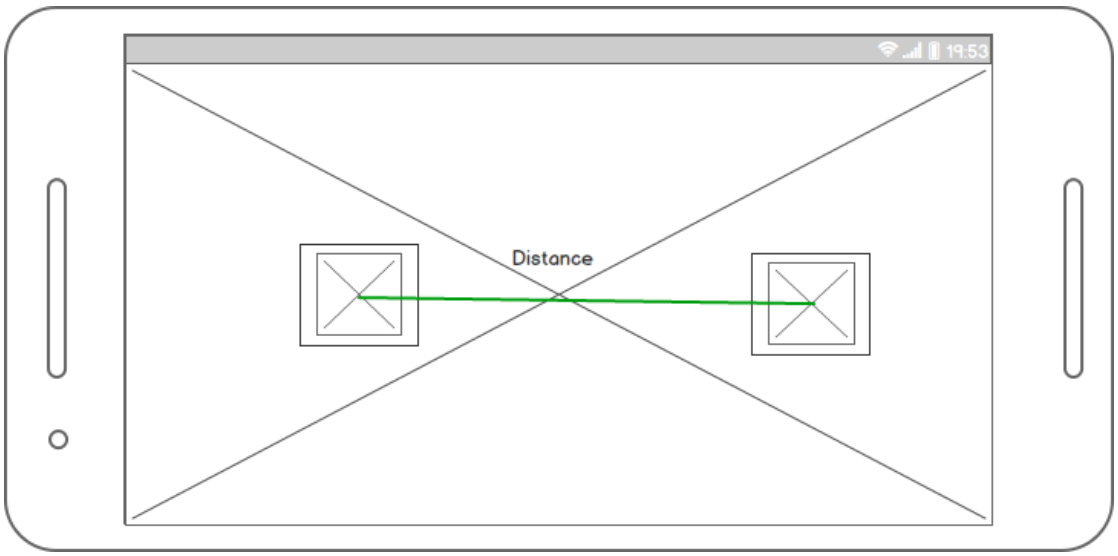

Figura 24: Boceto ventana distancia.

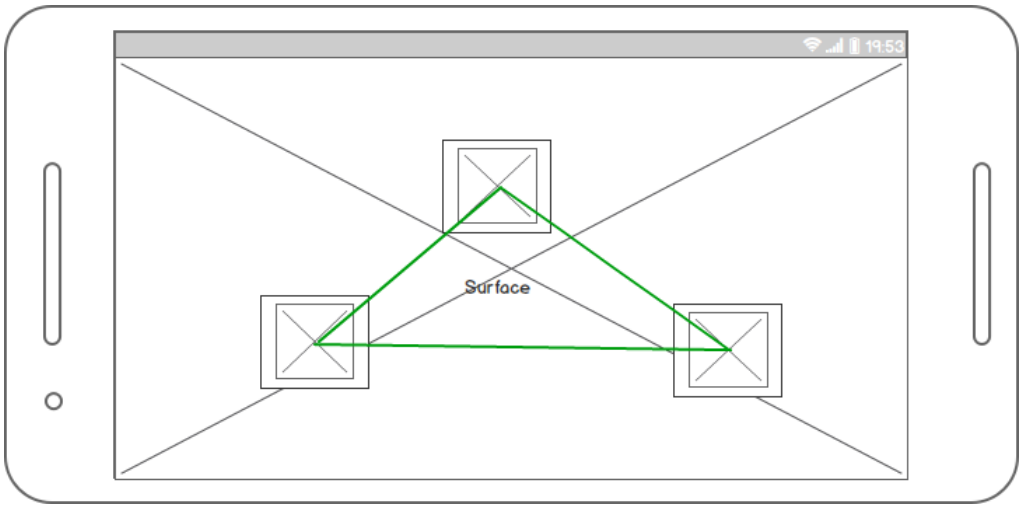

Figura 25: Boceto ventana superficie.

5. Desarrollo de la solución

En este capítulo procederemos a explicar cómo se desarrolló la aplicación, de la forma más breve posible, explicando problemas con los cuales nos encontramos, así como las decisiones que tomamos al respecto. Además, se mostrarán y explicarán algunos fragmentos de código, para que el lector se haga una idea más clara de su funcionamiento.

Para comenzar con el desarrollo de la aplicación debíamos elaborar el proceso de calibrado, para obtener los parámetros de la cámara. Tal y como se indica en [19] existen tanto parámetros intrínsecos como extrínsecos de la cámara. Los primeros son los parámetros que describen su funcionamiento, como pueden ser el centro óptico, el coeficiente de distorsión, la distancia focal, etc. En cambio, los parámetros extrínsecos se encargan de definir la posición y la orientación del cuadro de referencia de la cámara con respecto al mundo real, en otras palabras, proporcionan la orientación externa de la cámara.

Al realizar el calibrado obtenemos los parámetros intrínsecos, los cuales emplearemos para obtener los parámetros extrínsecos, estos serán empleados en las funciones de mediciones de la aplicación, que lo necesitan para obtener el sistema de coordenadas de los marcadores. Para llevar a cabo el calibrado es necesario utilizar un tablero compuesto por marcadores, existen diversos tipos de tableros: Chessboard, tableros ArUco y tableros ChArUco. En la Figura 26 podemos observar cómo es la apariencia de cada uno de ellos.

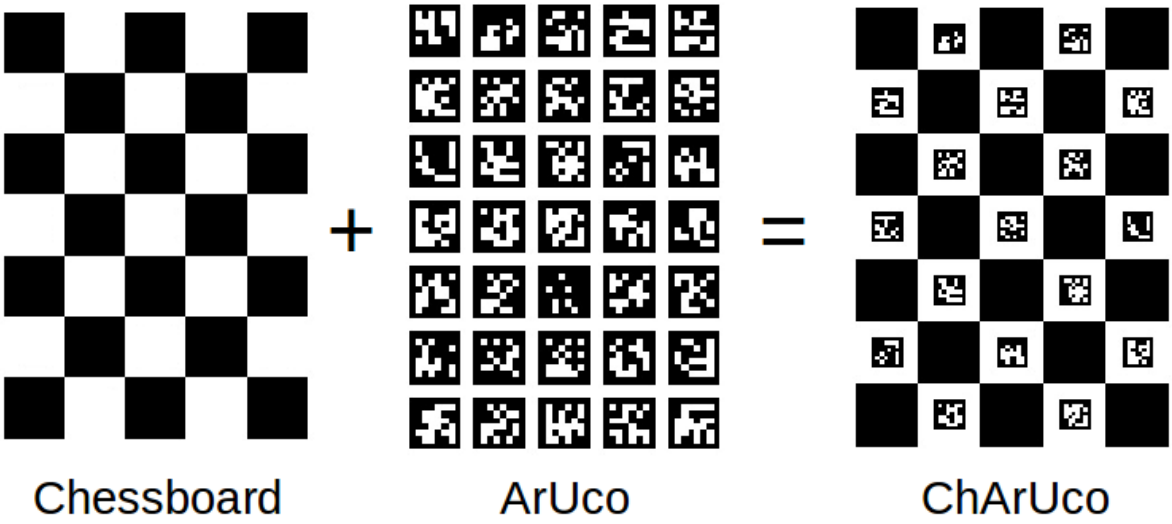

Figura 26: Tableros para calibración de la cámara. Extraída de [7].

Como se comenta en [7] los tableros ArUco son muy útiles debido a su rápida detección y su versatilidad, pero su principal problema es que la precisión de sus posiciones en las esquinas no es demasiado alta. Po otro lado, los Chessboard la

detección de las esquinas tiene mayor precisión ya que cada esquina está rodeada por dos cuadrados negros, pero no es tan versátil como un tablero ArUco, ya que tiene que ser completamente visible y las oclusiones no están permitidas. La alternativa que empleamos en la aplicación son los tableros ChArUco, debido a que combina los beneficios de las otras alternativas, permitiendo que no sea necesario visualizar completamente el tablero.

Para llevar a cabo el calibrado debemos llamar a la función calibrateCameraCharuco, del módulo ArUco. Esto nos devolverá un valor numérico, el cual se corresponderá con el error de proyección. Esto método necesita una serie de parámetros, nosotros en la versión Java empleamos los siguientes:

- CharucoCorners -> lista de matrices con las esquinas de cada marcador que forma el tablero.
- CharucoIds -> lista de matrices con los ids de cada marcador que forma el tablero.
- Board -> ChArUcoBoard creado con anterioridad, mediante la función create de CharucoBoard, a la cual hay que indicarle los parámetros deseados (número de cuadros, tamaño, ...).
- ImageSize ->tamaño de la imagen de entrada, en nuestro caso el tamaño del fotograma.
- CameraMatrix ->matriz 3x3 que contendrá los parámetros intrínsecos de la cámara. Solo necesitamos inicializarla, la llamada al método se encargar de rellenarla con la información necesaria.
- DistCoeffs -> matriz 1x5 que contendrá los coeficientes de distorsión de la cámara. Solo necesitamos inicializarla, la llamada al método se encargar de rellenarla con la información necesaria.

Para realizar el calibrado se necesita tomar diferentes capturas del tablero, desde diferentes perspectivas, almacenando las esquinas y los ids de los marcadores de cada captura, como matrices en listas, estas se corresponderán con las mencionadas en los parámetros de la función. En [19] se aconsejan realizar al menos quince capturas. Como experiencia propia he de indicar que, para obtener mejores resultados de calibrado, conviene situar el tablero de forma perpendicular a la cámara y rotarlo.

Una vez obtenido la matriz de la cámara y los coeficientes de distorsión, debía almacenarlos para utilizarlos en las diversas ventanas que necesitaran de ellos. En este punto encontré una diferencia con las versiones de otros lenguajes, en ellos se empleaba la función fileStorage para guarda los datos obtenidos, pero en Java esta opción no estaba disponible, por lo cual las alternativas eran: usar una base de datos o emplear el SharedPreferences de Android. Opte por la segunda opción, debido a que los datos que teníamos pensado almacenar era datos simples, tan solo

para guardar las preferencias del usuario, por lo que no vimos necesario el uso de base de datos. A continuación, mostraré cómo guardar estos datos en el SharedPreferences y cómo los cargábamos, ya que puede ser de utilidad para lectores que se encuentre con esta situación al emplear Java. Para su realización creamos una clase MySharedPreferences que se encargará de tratar con los todos datos ubicados en el SharedPreferences.

```
38  public boolean saveCameraParams(Mat cameraMatrix, Mat distorsion, List<Camera.Size> resolutionList){
39      cameraArray = new double[CAMERA_COLUMNS*CAMERA_ROWS];
40      distorsionArray = new double[DISTOSION_SIZE];
41
42      try{
43          SharedPreferences.Editor editor = preferences.edit();
44
45          cameraMatrix.get( row: 0, col: 0,cameraArray);
46          for (int i = 0; i<CAMERA_ROWS;i++) {
47              for (int j = 0; j < CAMERA_COLUMNS; j++) {
48                  //Lo almacenaremos con un id que se corresponde con la posicion del elemento en la ma
49                  int id = i * CAMERA_ROWS + j;
50                  editor.putFloat(String.valueOf(id),(float)cameraArray[id]);
51              }
52          }
53
54          distorsion.get( row: 0,  col: 0, distorsionArray);
55          //Los id correspondientes a la matriz de distorsión continuarán a partir del 8, que es el útl
56          int inicio = CAMERA_COLUMNS * CAMERA_ROWS;
57          for (int i = inicio; i < DISTOSION_SIZE + inicio; i++) {
58              editor.putFloat(String.valueOf(i),(float)distorsionArray[i-inicio]);
59          }
60
61          //La lista de resoluciones compatibles con el dispositivo se ubicará a partir de la matriz de
62          inicio = inicio + DISTOSION_SIZE;
63          for (int i = inicio; i < inicio + resolutionList.size(); i++){
64              String keyWidth = "width " + i;
65              String keyHeigth =  "height " + i;
66
67              editor.putInt(keyWidth, resolutionList.get(i-inicio).width);
68              editor.putInt(keyHeigth, resolutionList.get(i-inicio).height);
69          }
70
71          //Por último se almacena el tamaño de la lista de reoluciones comapatibles.
72          editor.putInt("size_list", resolutionList.size());
73
74          editor.apply();
75          return true;
76      }
77      catch (Exception e){
78          Log.i( tag: "prueba", e.getMessage());
79          return false;
80      }
81
82  }
```

El valor de las constantes es el siguiente:

- CAMERA_ROWS = 3. La matriz de la cámara es 3x3.
- CAMERA_COLUMNS = 3.
- DISTOSION_SIZE = 5. La matriz de distorsión es 1x5.

Para realizar el guardado de la matriz de la cámara que habíamos obtenido, la recorríamos y guardábamos cada elemento en el SharedPreferences, dándole como

clave el orden que ocupa en ella, es decir, el orden de la matriz seguiría la siguiente forma: [0 1 2;3 4 5; 6 7 8] (los ";" indican el final de la fila). Realizábamos el mismo proceso con la matriz de distorsión, pero comenzando desde la posición en la que se insertó el último elemento de la matriz de la cámara. También se puede observar que con este método también guardamos en el mismo archivo, la lista de resoluciones compatibles del dispositivo, la cual se puede obtener mediante la línea "mCamera.getParameters().getSupportedPreviewSizes()", la cual es empleada al realizar el calibrado por primera vez.

Para cargar los valores del SharedPreferences el funcionamiento fue similar, aunque tuvimos que almacenarlos previamente en un array, para su inserción en la matriz.

```
122     public boolean loadCameraParams(Mat cameraMatrix, Mat distorsion){
123         cameraArray = new double[CAMERA_COLUMNS*CAMERA_ROWS];
124         distorsionArray = new double[DISTOSION_SIZE];
125
126         try {
127             for (int i = 0; i < CAMERA_ROWS; i++) {
128                 for (int j = 0; j < CAMERA_COLUMNS; j++) {
129                     int id = i * CAMERA_ROWS + j;
130                     cameraArray[id] = preferences.getFloat(String.valueOf(id), defValue: -1);
131                 }
132             }
133             cameraMatrix.put( row: 0,  col: 0, cameraArray);
134
135             int inicio = CAMERA_COLUMNS * CAMERA_ROWS;
136             for (int i = inicio; i < DISTOSION_SIZE + inicio; i++) {
137                 distorsionArray[i - inicio] = preferences.getFloat(String.valueOf(i), defValue: -1);
138             }
139             distorsion.put( row: 0,  col: 0, distorsionArray);
140
141             return true;
142         }
143         catch (Exception e){
144             Log.i( tag: "prueba", e.getMessage());
145             return false;
146         }
147     }
```

Antes de comenzar a comentar las principales funciones de la aplicación, debemos tratar brevemente los objetos Dictionary, del módulo ArUco, ya que todas las funcionalidades de la aplicación que trabajan con la realidad aumentada hacen uso de ellos. Estos objetos contienen la codificación interna de los diversos marcadores que se emplearán. Existen diccionarios predefinidos, los cuales se seleccionan mediante la función getPredefinedDictionary, un ejemplo de estos es: DICT_4X4_50, el cual contiene cincuenta marcadores de 4 bits de tamaño. También existen diccionarios personalizados, que se pueden generar mediante la función generateCustomDictionary. Según el apartado de preguntas frecuentes de [19] es preferible emplear los predefinidos, indicando que es más fácil trabajar con ellos y que el uso de diccionarios personalizados es más adecuado cuando se necesita un diccionario más grande (en términos de cantidad de marcadores o cantidad de bits, que los que proporcionan los predefinidos).

Para el proceso de detección de marcadores, el cual también es empleado por todas las funcionalidades de la aplicación que hace uso de la realidad aumentada, tan solo necesitamos emplear la función detectMarkers, del módulo ArUco. La cual recibirá como parámetros:

- Image -> imagen que analizará, en nuestro caso fotograma.
- Dictionary ->diccionario predefinido o personalizado que hemos seleccionado.
- Corners -> lista de matrices, que contendrá las esquinas de cada marcador detectado en la imagen. La lista estará compuesta por una matriz por cada marcador detectado, cada matriz contendrá las cuatro esquinas de su marcador. Tan solo debemos inicializarla, la función se encargará de rellenarla con la información detectada.
- Ids ->Matriz que contendrá los ids de todos los marcadores detectados en la imagen. Tan solo debemos inicializarla, la función se encargará de rellenarla con la información detectada.

La función, internamente, se encargará de aplicar diferentes técnicas de visión por computador sobre la imagen, las cuales no veo necesario profundizar (si se desea hay un apartado en [19] que trata sobre ello), que determina que objetos de la imagen son marcadores correctos y cuáles se deben desechar.

En cuanto a las funcionalidades principales, hubo algunas decisiones y cambios que tuvimos que llevar a cabo debido a que la idea que teníamos no se podía aplicar correctamente. Las situaciones que no se pudimos solucionar durante el desarrollo fueron añadidas como consejos, en la ventana de más información, para que el usuario pudiera realizar algo al respecto.

Al principio del desarrollo de la aplicación, la funcionalidad de comprobación de paralelismo y alineación iba a ser realizado con solo dos marcadores, pero nos dimos cuenta de que de esa forma solo comprobábamos que los marcadores estuvieran en paralelo a la cámara, en cuanto inclinábamos el dispositivo se nos indicaba que el objeto ya no estaba en la posición adecuada. A causa de esto, decidimos añadir dos marcadores más, los cuales hicieran de referencia para los otros dos. El funcionamiento básico de esta funcionalidad acabo consistiendo en calcular las pendientes de las rectas formadas entre los pares de marcadores y cuando coincidieran se marcaría como paralelo o en línea, produciendo un sonido (para el caso de usar el dispositivo en alguna plataforma, como un trípode) y cambiando el color de las líneas que unían los marcadores, de rojo a verde. Para el caso de objetos verticales, que no tengan pendiente, se recomienda rotar el dispositivo, poniéndolo en una posición vertical, esto fue añadido en la aplicación como consejo. También decidimos proyectar iconos sobre los marcadores de referencia, para que el usuario distinguiera cuales iban a ir unidos y no los cruzase, provocando que tuviera que cambiarlos posteriormente de posición.

El proceso de proyección de una imagen me resultó bastante problemático en la fase previa y hasta que no comprendí el procedimiento no conseguí realizarlo. Para proyectar una imagen normal sobre un marcador aconsejo seguir los siguientes pasos:

- Obtener la matriz de proyección (emplear la función getPerpectiveTransform) mediante las esquinas del marcador y los puntos de las esquinas de la imagen a proyectar.

- Crear una matriz en blanco, toda a 255, del tamaño de la imagen. Para ello emplear función bitwise_not en una matriz de ceros.

- Adaptar la imagen y la matriz blanca, al fotograma con la matriz de proyección obtenida (emplear función warpPerspective). El adaptar la matriz blanca permite obtener la matriz en negro y solo habrá valores en la parte de la matriz que se corresponda con el lugar del marcador.

- Inviertir el resultado de adaptar la matriz blanca, para que los pixeles correspondientes al marcador se queden a 0 y 255 el resto.

- Emplear la función bitwise_and de esta matriz con el frame, obteniendo una copia del fotograma con el espacio del marcador vacío.

- Hacer uso de la función bitwise_or de la matriz obtenida en el paso anterior y la imagen adaptada (del paso 2), remplazando el negro del marcador por los bits de la matriz de la imagen.

- Mostrar el fotograma.

En nuestro caso, al emplear una imagen con trasparencias, la matriz blanca debe ser solo de ese color en las zonas de la imagen a proyectar que no tuviera transparencias, en caso contrario aparecerá un fondo negro cubriendo todo el marcador. La proyección de nuestro archivo PNG puede observarse en los siguientes fragmentos de código, siendo i la posición del marcador en la lista.

```
public void displayImage(int i){
    Mat projection = Imgproc.getPerspectiveTransform(Converters.vector_Point2f_to_Mat(imageQuadrangle), corners.get(i));

    Imgproc.warpPerspective(image, image_adapted, projection, frame4Chanels.size());
    Imgproc.warpPerspective(blank, blank_adapted, projection, frame4Chanels.size());

    Core.bitwise_not(blank_adapted, blank_adapted);
    Core.bitwise_and(blank_adapted, frame4Chanels, blank_adapted);
    Core.bitwise_or(blank_adapted, image_adapted, frame4Chanels);
}
```

En los siguientes dos fragmentos se puede observar la inicialización de los componentes empleados en el paso anterior.

```
imageQuadrangle = new ArrayList<>();
try {
    image = Utils.loadResource(getApplicationContext(), R.drawable.anchor);
    //OpenCV trabaja con las imagenes en BGR, si no se convierte saldrán los colores de forma erronea
    Imgproc.cvtColor(image,image,Imgproc.COLOR_RGBA2BGRA);
    imageQuadrangle.add(new Point( x: 0, y: 0));
    imageQuadrangle.add(new Point(image.cols(), y: 0));
    imageQuadrangle.add(new Point(image.cols(),image.rows()));
    imageQuadrangle.add(new Point( x: 0,image.rows()));
} catch (IOException e) {
    Toast.makeText(getBaseContext(),"Error inesperado. Contacte con pablohe2@inf.upv.es",Toast.LENGTH_LONG).show();
}
```

```
List<Mat> channels = new ArrayList<>();
//Dividimos los canales en 4 matrices
split(image,channels);
//Matriz blanca, para las comparaciones.
blank = Mat.zeros(image.size(),image.type());
//Cambiamos los pixeles a blanco, en las zonas de no transparencia, obteniendo la figura png en blanco.
blank.setTo(new Scalar(255,255,255,1),channels.get(3));
```

```
@Override
public void onCameraViewStarted(int width, int height) {
    frame = new Mat(height,width, CvType.CV_8UC3);
    frame4Chanels = new Mat(height,width, CvType.CV_8UC4);
    image_adapted = Mat.zeros(frame.size(),frame4Chanels .type());
    blank_adapted = Mat.zeros(frame.size(),frame4Chanels .type());
}

@Override
public void onCameraViewStopped() {
    frame.release();
    frame4Chanels.release();
    image_adapted.release();
    blank_adapted.release();
}
```

En cuanto a la funcionalidad de distancia y perímetro, funcionan empleando la fórmula de distancia entre puntos, indicándole los puntos de los diversos marcadores en pantalla. Si se activa la opción de perímetro pasa de mostrar las

distancias, entre los diferentes marcadores, a mostrar una única medida en el centro de la figura formada por ellos. Tanto en esta como en la funcionalidad de superficie necesitamos emplear los parámetros extrínsecos de la cámara, para ello necesitábamos cargar los parámetros intrínsecos guardados en el SharedPreferrences y emplear la función estimatePoseSingleMarkers, proporcionándonos el vector de rotación y translación, los cuales serán empleados para calcular la distancia y superficie de los marcadores. Para el uso de esta función es necesario proporcionar una serie de parámetros:

- Corners -> lista de matrices, que contendrá las esquinas de cada marcador detectado en la imagen. Tan solo debemos inicializarla, la función detectMarkers se encargará de rellenarla con la información detectada.

- MarkerLength -> Tamaño de los marcadores.

- CameraMatrix -> matriz 3x3 que contendrá los parámetros intrínsecos de la cámara.

- DistCoeffs -> matriz 1x5 que contendrá los coeficientes de distorsión de la cámara.

- Rvecs -> Matriz con el vector de rotación. Tan solo debemos inicializarla, la función se encargará de rellenarla con la información obtenida.

- Tvecs -> Matriz con el vector de translación. Tan solo debemos inicializarla, la función se encargará de rellenarla con la información obtenida.

Como no podíamos asegurar que todos los usuarios emplearan el mismo tamaño de marcador, decidimos añadirlo como una opción configurable en ajustes. En este caso no proporcionaré código de ejemplo debido a que es más extenso que los casos anteriores.

Por último, en la funcionalidad de superficie, decidimos emplear la fórmula del área de Gauss[13], debido a que no podríamos saber qué figura haría el usuario con los marcadores y esta fórmula nos permite calcular el área de un polígono simple. La fórmula consiste básicamente en formar una matriz con las coordenadas de los puntos y tras obtener su determinante, dividirlo entre dos. A continuación, se puede observar cómo fue adaptada a código Java.

[13] https://es.wikipedia.org/wiki/F%C3%B3rmula_del_%C3%A1rea_de_Gauss

```java
/**
 * Calcula la superficie de un polígono.
 * @param points puntos que delimitan el poígono.
 * @param accuracy precisión deseada por le usuario.
 * @return Superficie del poligono.
 */
public static double calculateSurface(List<Point3> points,double accuracy){
    double positive = 0;
    double negative = 0;

    points.add(points.get(0));
    for (int i = 0; i < points.size(); i++) {
        if (i + 1 < points.size()) {
            positive += points.get(i).x * points.get(i + 1).y;
            negative += points.get(points.size()-1 - i).x * points.get(points.size()-1 - (i + 1)).y;
        }
    }

    double determinant = Math.abs(positive - negative)/2;
    //Lo convertimos en cm2
    double value = determinant * 10000;

    return  Math.round(value * accuracy)/accuracy;
}
```

Como puede observarse, le proporcionamos un parámetro con la precisión. Esto es la solución que utilizamos para el problema que nos encontramos en todas las funcionalidades de medición, era difícil mantener los resultados estables, incluso siendo afectado por el propio pulso del usuario, por ello se decidió limitar la precisión de las operaciones. Se opto por permitir que el usuario escogiera entre tres opciones, desde la ventana de ajustes, las cuales limitarían el número de decimales (baja precisión elimina los decimales, media precisión permite un decimal y alta precisión emplea dos decimales). No conseguimos encontrar una mejor solución y dejamos abierta la posibilidad de en futuras mejoras solucionarlo de alguna otra manera.

También he de comentar que OpenCV ofrece funciones de dibujado bastante fáciles de utilizar. Empleamos la función line, del módulo Imgproc, para dibujar las diferentes líneas, tanto en la función de distancia como en la de paralelismo. En cambio, en superficie se empleó la función fillPoly, del mismo módulo, para dibujar un polígono relleno.

Por último, he de mencionar que la aplicación también se adaptó a tamaños de pantalla considerados xlarge, para un uso correcta de ella en tablets. Además de crear varios archivos string.xml para cada uno de los idiomas que indicamos en los requisitos, permitiendo al usuario elegir cual deseaba emplear de las tres opciones disponibles.

6. Pruebas

Para comprobar el correcto funcionamiento de la aplicación, hicimos uso de ella, poniéndonos en la piel de usuario. Además, calculamos la incertidumbre de las mediciones, comprobando los márgenes de precisión de la aplicación.

El correcto funcionamiento de las opciones de configuración fue comprobado al modificarlas y asegurar que se guardaban exitosamente, observando que la nueva configuración se emplea en las diferentes ventanas.

También se comprobó que al iniciar por primera vez la aplicación se generaran correctamente las imágenes de los marcadores, en la carpeta descargas del dispositivo. Además de la posibilidad de descargarlas de nuevo desde la ventana de descargas de la aplicación, comprobado que aparecían en la carpeta tanto la marca indicada, mediante su identificador, como todas las marcas necesarias. Asimismo, se comprobó que el enlace a Google Drive funcionará, descargándose el archivo comprimido con el material necesario.

En cuanto al calibrado, realicé diversos intentos, observando que al recalibrar se veía plasmado en el dibujado de los ejes, en la opción de validar calibrado. Para acabar de verificar su correcto funcionamiento, comprobé que las otras funciones, las cuales emplean los parámetros de la cámara obtenidos en esta función, proporcionaban valores adecuados.

Por otro lado, para comprobar que la funcionalidad de comprobación de paralelismo funcionaba adecuadamente, se buscó un objeto que estuviera nivelado y se comparó con el suelo, asegurando antes de que este también estuviera nivelado. Posteriormente, se comprobó que la aplicación indicara que sí que eran paralelos. Para comprobar el caso contrario se añadió un poco de altura a uno de los marcadores, y se comprobó que esta vez indicara lo contrario. Esta situación se puede observar en la Figura 27.

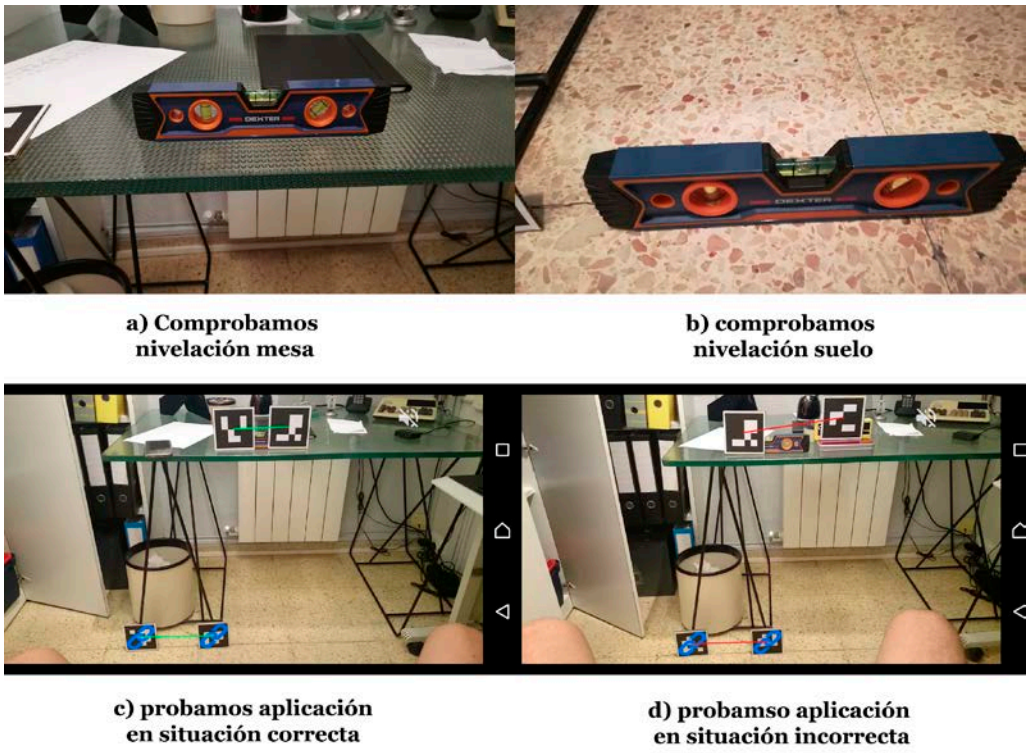

Figura 27: Prueba de paralelismo

Para averiguar sobre cuánto podían variar los resultados, indicando soluciones erróneas, situé los marcadores en paralelo y procedí a desplazarlos lentamente hasta el punto en que dejara de indicar que estaba en paralelo, midiendo la distancia en la que lo había desplazado. Con ello observe que:

- Con baja precisión me permitía desplazarlo más de 5 centímetros sin que detectará que ya no era paralelo.

- Con media precisión permitía desplazarlo hasta 0'8 centímetros. Siendo esta la mejor alternativa.

- Con alta precisión no se permite desplazarlo, podríamos estar hablando de 0.01 centímetros, debido a que al mínimo movimiento deja de detectar que son paralelos. Incluso pudiendo indicar este resultado estando sobre una superficie estable, debido a fallos en la detección de los marcadores.

A continuación, comprobamos el funcionamiento de las mediciones de distancia, para ello medimos con el metro una distancia y comprobamos que los resultados proporcionados. En la Figura 28 podemos observar como medimos el largo de un televisor, dando 62'2 centímetros y con la aplicación obtenemos 62'61 centímetros.

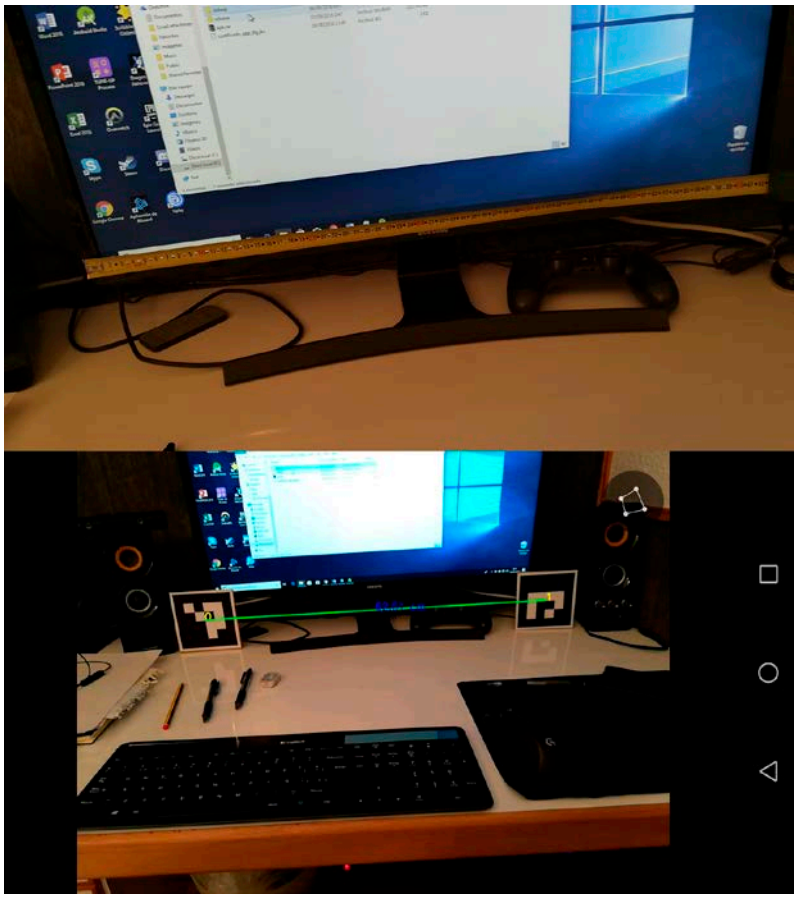

Figura 28: Prueba de distancia.

Esta prueba también fue realizada con los diversos tipos de precisión. Tras comprobar cuanto solía variar el resultado con respecto a lo indicado con el metro, permaneciendo lo más inmóvil posible, se obtuvo que:

- Con precisión baja el resultado es de ±1 centímetro. Aunque los resultados que se muestran permanecen más estable, incluso si nos desplazamos.

- Con precisión media es de ±0'6 centímetros. Aunque la estabilidad soporta peor los desplazamientos

- Con precisión alta es de ±0'45 centímetros, pero siendo muy sensible a cualquier movimiento.

También se comprobó el funcionamiento de la medición de distancia con profundidad, para ello se midió la diagonal de un folio y se comprobó los resultados obtenido con el uso de la aplicación. Esta situación se puede observar en la Figura 29, la medida real era de 36.2 centímetros y la obtenida fue de 35.31 centímetros, en este caso la precisión de la aplicación se redujo respecto a la medición de distancia sin profundidad y para obtener resultados más acercados al real debíamos captar una perspectiva desde la cual se apreciaría la separación entre los marcadores, desde una perspectiva frontal se obtenían peores resultados.

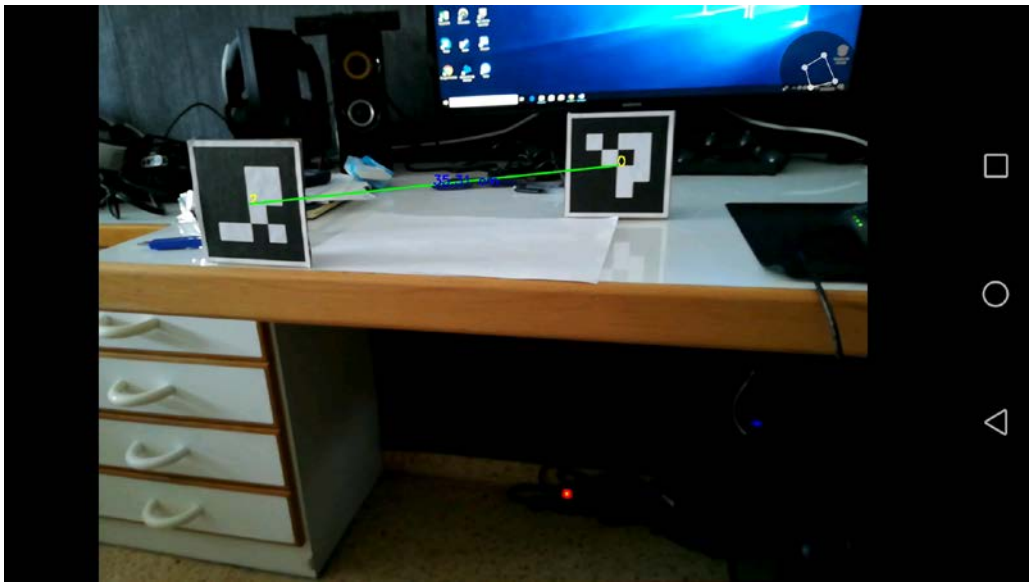

Figura 29: Prueba distancia con profundidad.

El funcionamiento del perímetro se comprobó observando si sumaba correctamente todas las distancias que había entre los diferentes marcadores que había en pantalla. Esta prueba se realizado con precisión baja, facilitando la comprobación de resultados. Se puede observar su realización en la Figura 30.

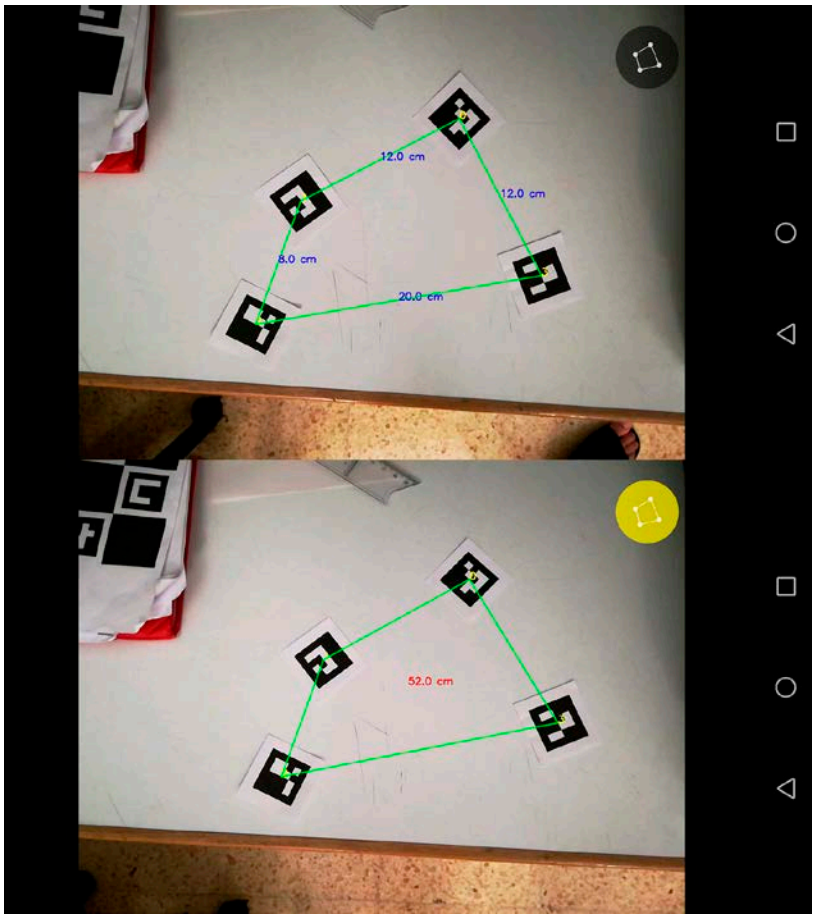

Figura 30: Prueba de perímetro.

Por último, se corroboró el correcto funcionamiento del cálculo de la superficie, de la figura formada por los marcadores. Para esta prueba se realizó un dibujo de un triángulo de altura 20’9 centímetro y base de 19’4 centímetros, obteniendo como superficie real un valor de 202’73 centímetros cuadrados. En esta prueba, los resultados se ven bastante afectados por el movimiento, independientemente de la precisión seleccionada. La realización de esta prueba puede ser observada en la Figura 31.

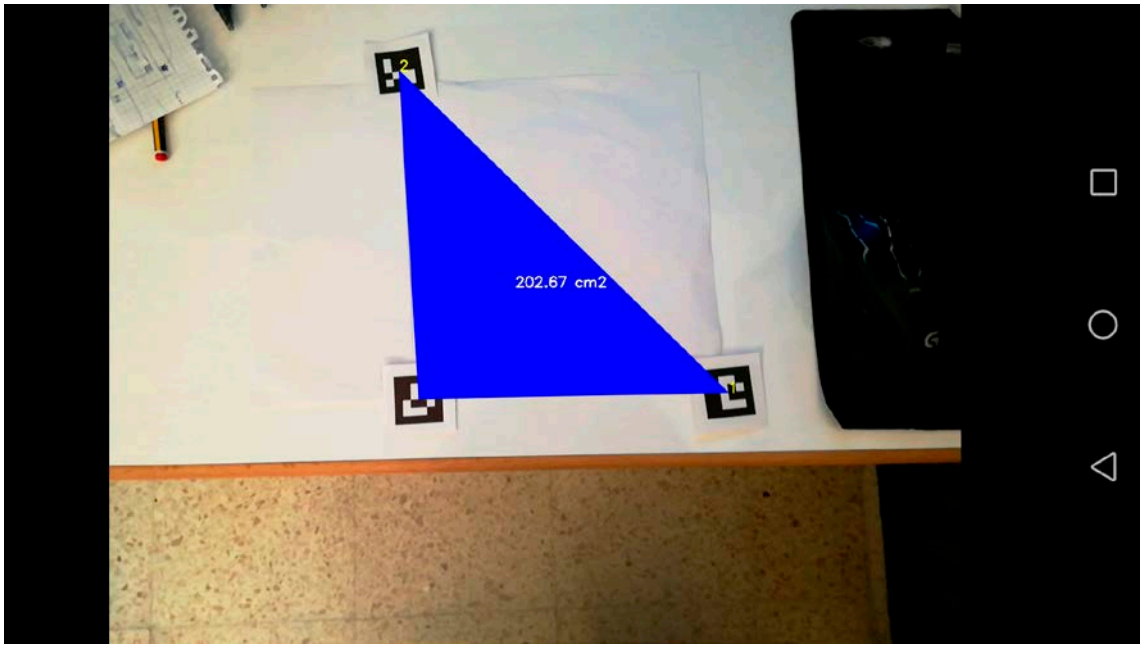

Figura 31: Prueba superficie.

Aunque el resultado se asemeja bastante al real, suele ser poco estable, esto es causado por realizar la fórmula del área de Gauss con puntos que pueden variar fácilmente, debido al pulso del usuario o fallos en la detección del marcador. Con los resultados observados podemos indicar que:

- Con precisión baja: el resultado puede variar ±4 centímetros cuadrados.
- Con precisión media: el resultado puede variar ±3’5 centímetros cuadrados.
- Con precisión alta: el resultado puede variar ±2’75 centímetros cuadrados.

Cabe mencionar que no se pueden asegurar con totalidad las incertidumbres indicadas, debido a que depende de diversos factores: luz ambiente, resolución, calidad de cámara, correcta impresión de los marcadores, etc.

7. Conclusiones

En este último apartado vamos a tratar las conclusiones que hemos obtenido tras desarrollar el proyecto, realizando una valoración personal de él, y ofreceremos una serie de mejoras que podrían desarrollarse en un futuro próximo.

Antes de comenzar con ello, debemos analizar si los objetivos que se plantearon al inicio del proyecto fueron cumplidos satisfactoriamente. El proyecto se centró, en los primeros capítulos, en definir en qué consistía la realidad aumentada, exponiendo diferentes herramientas para el desarrollo de aplicaciones de esta índole, cumpliendo el primer y segundo objetivo.

Dentro de la aplicación se estipuló que funcionara en dispositivos cuya versión del sistema operativo fuera superior a la 5.0. y se comprobó su funcionamiento en varios dispositivos, concretamente dos teléfonos móviles (Huawei P10 Lite 8.0, Xperia Z2 6.0.1) y una *tablet* (Samsung Galaxy Tab S2 7.0), cumpliendo el tercer objetivo.

Además, la aplicación fue desarrolla de tal forma que el usuario solo necesita su dispositivo móvil para emplear la aplicación, pudiendo realizar cálculos de distancia, perímetro, superficie y comprobaciones sobre el paralelismo o alineamiento de marcadores, cumpliendo el cuarto y quinto objetivo.

Como podemos observar todos los objetivos que se propusieron han sido cumplidos, por lo cual podemos determinar que el proyecto ha sido un éxito.

7.1 Conocimientos adquiridos

El desarrollo de este proyecto me ha permitido adquirir ciertos conocimientos que no se impartieron durante la carrera o se trataban en asignaturas optativas, las cuales no cursé. He aprendido a desarrollar aplicaciones Android desde cero, aprendiendo el funcionamiento de sus diferentes componentes y cómo adaptarlos a los distintos tamaños de dispositivos, conocimientos que me serán de gran utilidad en el futuro, debido a la gran importancia que han adquirido hoy en día este tipo de aplicaciones.

De igual modo he aprendido sobre la librería OpenCV y Aruco, permitiéndome experimentar con la visión por computador.

Durante el desarrollo del proyecto también descubrí el servicio web de control de versiones Gitlab, el cual empleé para mantener mi proyecto seguro.

Por otro lado, he adquirido conocimientos sobre las nuevas tecnologías, como la RA, RV y RM, aprendido a diferenciar entre ellas. Me ha posibilitado conocer las diferentes herramientas que se emplean en el desarrollo de las aplicaciones de RA,

conociendo ciertas de sus características y descubrir los proyectos futuros de Google y Apple respecto a esta tecnología.

También me ha permitido emplear conocimientos adquiridos en la carrera, como la programación en Java, especificación requisitos y uso del metalenguaje XML, tanto para el desarrollo de los *layouts* de Android como para almacenar la información de los textos que iba a emplear. Además de realizar la creación de mockups mediante el programa Balsamiq Mockups 3, el mism0 que emplee en las asignaturas de Análisis y Especificación de Requisito y Proyecto de Ingeniería del Software. También llevé a cabo algunos *refactorings* para obtener código limpio y facilitar el mantenimiento.

Por último, me ha permitido repasar conceptos de matemáticas que no recordaba, como el cálculo de la pendiente de una recta o la fórmula del área de Gauss.

7.2 Trabajos futuros

Aunque el proyecto haya finalizado, quedan pendientes muchas mejoras que podrían desarrollarse, algunas de las cuales tenía pensado realizar, pero debido al tiempo limitado y la falta de recursos quedaron en un segundo plano:

- Añadir la posibilidad de emplear ARCore a los usuarios que dispongan de un dispositivo compatible.
- Permitir cambiar el tipo de unidades de las mediciones.
- Añadir más idiomas.
- Adaptar la aplicación a un mayor número de tamaños, permitiendo una visualización más atractiva para los distintos dispositivos.
- Añadir más funciones a la aplicación, como cálculo del ángulo que se forma entre los marcadores o proyectar cualquier imagen con el tamaño que deseemos sobre el marcador, permitiendo visualizar como quedaría un cuadro con esa imagen.
- Permitir mayor configuración, permitiendo seleccionar el color de dibujado al usuario o la posibilidad de elegir desde qué parte de la marca desea realizar la medición: esquina superior derecha, centro, esquina inferior izquierda...
- Mejoras en la precisión y estabilidad de los resultados.

7.3 Valoración personal

A pesar de haber cumplidos los objetivos y estar conforme con la aplicación desarrollada, tras realizarla empleando marcadores e investigar sobre las diferentes herramientas de desarrollo de aplicaciones de RA, me quedó la sensación de que el uso de marcadores puede quedar en un segundo plano, debido a que, si ARCore y ARKit siguen aumentando el número de dispositivos compatibles, los marcadores ya no serán necesarios para ubicar los objetos virtuales de forma precisa. Esto es bastante probable y es cuestión de tiempo, debido al avance de los propios dispositivos, que mejoran continuamente su tecnología, llegando un punto en que todos los productos del mercado sean compatibles con las nuevas formas de RA. Si tuviera que realizar el proyecto de nuevo y no estuviera limitado a realizarlo en un tiempo determinado o emplear OpenCV, optaría por emplear ARCore, concretamente usando la última versión de Vuforia junto a Unity.

8. Bibliografía

[1] *Apple desvela ARKit 2*. (4 de junio de 2018). Obtenido de Apple - Newsroom: https://www.apple.com/es/newsroom/2018/06/apple-unveils-arkit-2/

[2] *ArKit vs ArCore: La realidad aumentada móvil de Apple y Google*. (24 de octubre de 2017). Obtenido de Neosentec: https://www.neosentec.com/arkit-vs-arcore-realidad-aumentada-movil/

[3]*ArUco: una biblioteca mínimo para aplicaciones de realidad aumentada basada em OpenCV*. (s.f.). Obtenido de Realidad Aumentada Perú: http://realidadaumentadaperu.blogspot.com/2013/07/aruco-una-biblioteca-minimo-para.html

[4]Christina. (4 de octuber de 2017). *5 Best ARKit Apps for iPhone and iPad to Use in iOS 11*. Obtenido de Unlockboot: https://www.unlockboot.com/best-arkit-apps-iphone/

[5]Compilar el SDK OpenCV para Android. (16 de febrero de 2018). Obtenido de Tutor de Programación: http://acodigo.blogspot.com/2018/02/compilar-el-sdk-opencv-para-android.html

[6]Continuo de la virtualidad. En Wikipedia. Recuperado el 17 de agosto de 2018 de https://es.wikipedia.org/wiki/Continuo_de_la_virtualidad

[7]*Detection of ChArUco Corners*. (s.f.). Obtenido de OpenCV Docs: https://docs.opencv.org/3.1.0/df/d4a/tutorial_charuco_detection.html

[8]Gleb, B. (2011-2018). *rubygarage*. Obtenido de https://rubygarage.org/blog/difference-between-ar-vr-mr

[9]*Global revenue of the consumer mobile augmented reality app market (standalone/embedded) from 2016 to 2022 (in million U.S. dollars)*. (2016). Obtenido de statista: https://www.statista.com/statistics/608990/mobile-ar-applications-installed-base-worldwide-by-type/

[10]González Morcillo, C., Vallejo Fernández, D., Albusac Jiménez, J. A., & Castro Sánchez, J. J. (2012). *Realidad Aumentada. Un Enfoque Práctico con ARToolKit y Blender*. Ciudad Real: IdenTIC.

[11]Gracia, L. M. (9 de octubre de 2013). *¿Qué es OpenCV?* Obtenido de Un Poco De Java: https://unpocodejava.com/2013/10/09/que-es-opencv/

[12]Imascono Team. (19 de junio de 2017). *Tipos de Realidad Aumentada según sus formas de utilización*. Obtenido de imascono: http://imascono.com/es/magazine/realidad-aumentada-segun-utilizacion

[13]Inglobe. (25 de mayo de 2011). *ARToolKit Professional for Android*. Obtenido de AR Blog: http://arblog.inglobetechnologies.com/?p=421

[14]Mainelli, T. (mayo de 2018). How Augmented Reality Drives Real-World Gains in Services,. Framingham, Massachusetts, USA: IDC. Obtenido de IDC: https://www.ptc.com/en/resources/iot/webcast/ar-drives-real-world-gains

[15]Megali, T. (12 de septiembre de 2016). *Realidad Aumentada del Estilo de Pokémon GO con Vuforia*. Obtenido de tutsplus: https://code.tutsplus.com/es/tutorials/introducing-augmented-reality-with-vuforia--cms-27160

[16]Molina, M. (30 de agosto de 2017). *Ejemplos de apps usando ARKit iOS 11*. Obtenido de Poder pda: https://www.poderpda.com/plataformas/android/apps-usando-arkit-ios11/

[17]Monica. (21 de enero de 2015). *Arloon. App educativas de ciencias con Realidad Aumentada*. Obtenido de Moncadele: http://mocadele.net/arloon-app-educativas-de-ciencias-con-realidad-aumentada/

[18]Muñoz Salinas, R. (s.f.). *ArUco: a minimal library for Augmented Reality applications based on OpenCV*. Obtenido de UCO (Universidad de Córdoba): https://www.uco.es/investiga/grupos/ava/node/26

[19]Muñoz Salinas, R. (s.f.). *ArUco: An efficient library for detection of planar markers and camera pose estimation*. Obtenido de Google Docs: https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit#

[20]PlayStation Mobile Inc. (5 de diciembre de 2013). *Invizimals™: Desafíos Ocultos APK*. Obtenido de Google Play: https://play.google.com/store/apps/details?id=com.sony.novarama.invizimalstcg&hl=es

[21]Project Tango. En Wikipedia. Recuperado el 17 de agosto de 2018 de https://es.wikipedia.org/wiki/Project_Tango

[22]Priyadarshini, M. (21 de marzo de 2018). *Google Goes Big On ARCore With Over 60 New Apps In Play Store: Here's A Glimpse*. Obtenido de Fossbytes: https://fossbytes.com/google-augmented-reality-apps/

[23]Vera, R. (27 de junio de 2018). *ARCore: Así es como se usa la realidad aumentada en Android*. Obtenido de Inmersys: https://www.inmersys.com/blog/arcore-como-se-usa-realidad-aumentada-en-android

[24]Wilson, B. (s.f.). *Cómo la realidad aumentada puede revolucionar la industria de la Hospitalidad*. Obtenido de Realidad Aumentada Perú: http://realidadaumentadaperu.blogspot.com/2014/09/como-realidad-aumentada-puede.html

[25]Zamrath, N. (17 de marzo de 2018). *Building OpenCV & OpenCV Extra Modules For Android From Source*. Obtenido de You, Myself and Community: https://zami0xzami.wordpress.com/2016/03/17/building-opencv-for-android-from-source/

9. Anexos

9.1 Instalación

Antes de proceder con el proceso de instalación, cabe mencionar que para llevarla a cabo revisé los pasos de guías como las que se encuentran en [5] y [25], facilitándome realizar la instalación de OpenCV en Android Studio con los módulos extra, permitiendo el uso de Java. A continuación, comentaré los procesos que seguí personalmente y qué problemas encontré durante el proceso.

Para comenzar necesitamos realizar los siguientes pasos:

- Descargar y descomprimir los archivos fuente de OpenCV, en nuestro caso hemos empleado la versión 3.4.0. Lo puedes obtener desde el siguiente enlace: https://opencv.org/releases.html

- Descargar y descomprimir los módulos extra de OpenCV, la versión debe coincidir con el archivo descargado en el anterior punto. Se puede obtener desde: https://github.com/opencv/opencv_contrib/tree/3.4.0

- Descargar e instalar CMake, para generar los archivos necesarios del proyecto. Puede ser descargado desde el siguiente enlace: https://cmake.org/download/

- Descargar e instalar MinGW, asegúrate de agregar la carpeta <mingw>/bin a la variable de entorno, concretamente la variable de sistema, PATH. Puede ser descargado desde el siguiente enlace: https://sourceforge.net/projects/mingw/?source=typ_redirect

- Descargar e instalar Apache Ant. Deberemos agregar la variable de sistema ANT_HOME apuntando a la carpeta descomprimida <ant> y agregar la carpeta <ant>/bin a la variable de sistema PATH, igual que en el anterior punto. Puede ser obtenido desde el siguiente enlace: http://ant.apache.org/bindownload.cgi?Prefer

- Descargar e instalar Python 2.x. De igual modo debemos agregar la ruta a la variable de sistema PATH. Se puede descargar desde la página de Python: https://www.python.org/downloads/

- Debemos tener instalado Java JDK y agregar su ruta a la variable de sistema PATH. Podemos obtenerlo desde la página de Oracle: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

- Descargar y descomprimir Android NDK. Agregando la ruta de la carpeta descomprima a la variable del sistema PATH. Aunque se puede obtener desde el propio Android Studio, desde la estructura del proyecto, también es posible descargarlo desde el siguiente enlace: https://developer.android.com/ndk/downloads/?hl=es-419

Tras realizar estos preparativos, debemos ejecutar CMake debemos indicar la carpeta de la cual extraerá el código fuente de la librería OpenCV y la carpeta donde se guardarán los archivos generados por el programa. En la Figura 32 se puede observar este paso.

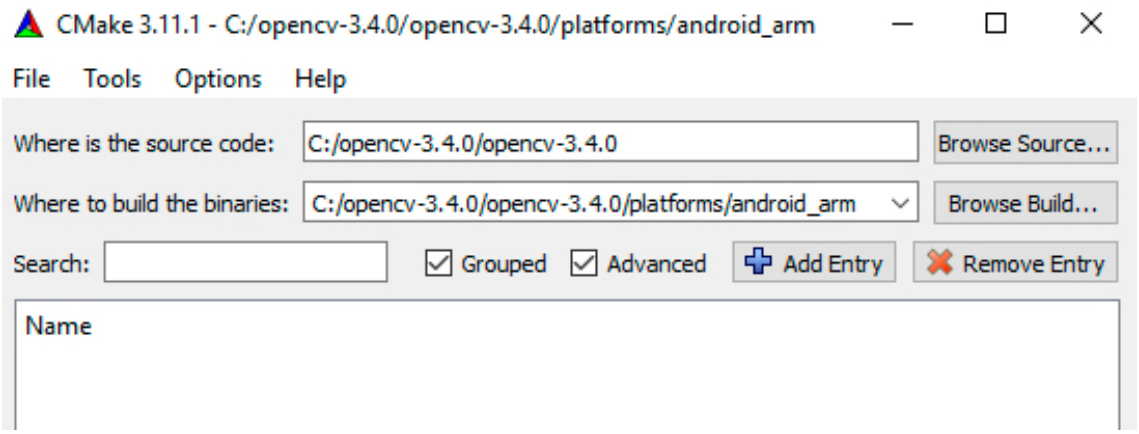

Figura 32: Selección de carpetas CMake.

Una vez seleccionadas las carpetas, presionamos el botón Configure, seleccionamos el generador MinGW Makefiles y activamos la opción Specify toolchain file for cross-compiling. Tras presionar en el botón Next, nos aparecerá una ventana, donde introduciremos el Toolchain a utilizar, se encuentra dentro de la carpeta descargada de OpenCV, en <opnecv>/platforms/android. Ambas ventanas pueden ser visualizadas en la Figura 33.

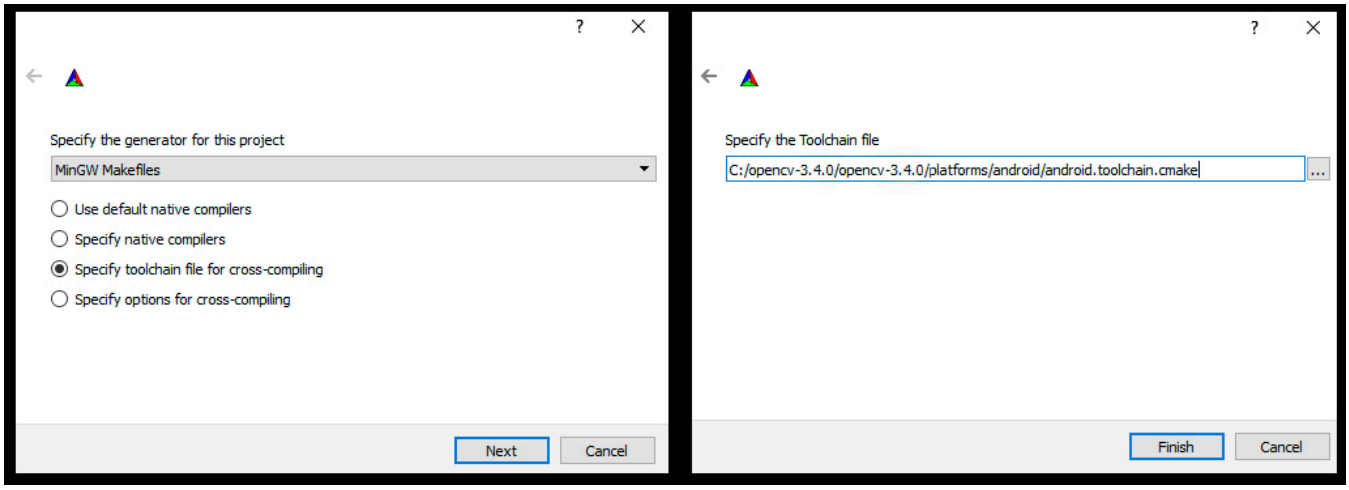

Figura 33: Especificando Toolchain en CMake

Tras presionar el botón Finish, se generar una serie de entradas en la lista de configuración de CMake. En caso de producirse algún problema, suele ser debido a que no encuentra el directorio de alguna de las herramientas de los pasos previos, en el mensaje de error suele indicarse cuál es el fallo. Si el fallo se debe a esto, configure la entrada de la lista, con el path de la herramienta que no detecta. También puede solucionarse al añadirla a las variables de entorno del sistema.

Antes de darle a generar, es aconsejable revisar que se han detectado correctamente los directorios de las herramientas indicadas al principio del capítulo. También debemos comprobar que las entradas relacionadas con Java quedan de una manera similar a la Figura 34, debido a que no detectaba correctamente el jdk al generar los java wrappers, produciendo un error.

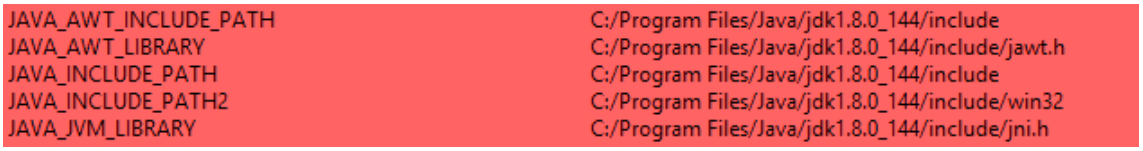

Figura 34: Configuración java CMake.

Además, debemos de configurar o añadir, en caso de que no estén, una serie de entradas:

- Nombre: OPENCV_EXTRA_MODULES_PATH
  Tipo: PATH
  Valor: <opencv-contrib-source>/modules

- Nombre: ANDROID_SDK_TARGET
  Tipo: STRING
  Valor: la versión de Android que desees, por ejemplo, android-21.

- Nombre: ANDROID_NATIVE_API_LEVEL
  Tipo: STRING
  Valor: nivel de API del NDK que deseemos, por ejemplo, 19. En el siguiente enlace se muestra una lista de las diferentes API: https://developer.android.com/ndk/guides/stable_apis?hl=es-419

- Nombre: ANDROID_ABI
  Tipo: STRING
  Valor: la arquitectura de procesador deseado, por ejemplo, armeabi-v7a

- Nombre: BUILD_JAVA
  Tipo: BOOLEAN
  Valor: true

La primera palabra de su nombre indica en que apartado de la lista se encuentra, facilitando su búsqueda. Una vez hecho todos estos pasos, volvemos a

pulsar sobre Configure, para actualizar los valores y si no se produce ningún pulsamos en Generate. Si se generaron correctamente los java wrappers se observará que en la consola de CMake aparece un mensaje indicándolo, Java wrappers: YES. Si no se produjo ningún error durante este proceso, procederíamos a emplear MinGW.

Abrimos la terminal de Windows y accedemos a la carpeta donde fueron guardados los archivos que se generaron en el paso anterior. Una vez hecho esto, ejecutamos el siguiente comando: mingw32-make y procederá la ejecución mostrando una ventana similar a la Figura 35.

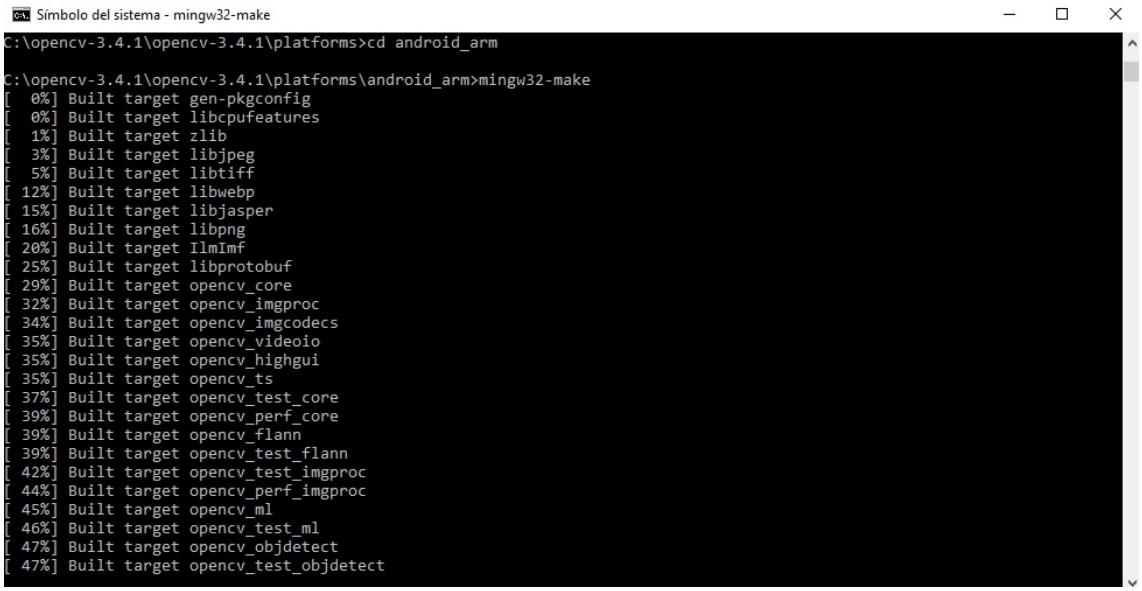

Figura 35: ejecución mingw32-make.

Si se produjo un error será notificado mediante un mensaje de fallo. Antes de comenzar la siguiente fase, mostraré un par de errores con los que me encontré y como los solucioné, por si mi experiencia puede ser de utilidad a las personas que estén intentado realizar la instalación:

- **Error**: C:\opencv_contrib-3.4.1\opencv_contrib-3.4.1\modules\xfeatures2d\src\boostdesc.cpp: 653:37: fatal error: boostdesc_bgm.i: No such file or directory #include "boostdesc_bgm.i"

  Solución: Acceder al repositorio de opencv3rdparty y descargar opencv_3rdparty-contrib_xfeatures2d_boostdesc_20161012 y ubicar los archivos bootdesc en la carpeta. Mismo error con vgg, por lo que hice lo mismo, pero descargando opencv3rdparty-contrib-xfeature2d-vgg-20160317.

- **Error**: The "android" command is deprecated.

  Solución: Realizar un *downgrade* del sdk.

- **Error**: SyntaxError: Missing parentheses in call to 'print'. Did you mean print ("Usage:\n", \)?

  Solución: Esto me ocurrió al emplear Python 3, la solución que encontré era emplear en su lugar Python 2 o poner los paréntesis en los lugares que indicaba el fallo.

En cambio, si se completó de manera exitosa, se habrá generado una carpeta install dentro de la carpeta indicada. Ahora debemos abrir Android Studio, acceder al proyecto que hayamos creado y llevar a cabo los siguientes pasos:

- Desde el menú principal, seleccionamos File –> New –> Import Module. En este punto introducimos la ubicación de la carpeta generada, .../install/sdk/java. Si la carpeta java no aparecer, significa que no se ha generado correctamente los java wrappers.

- Debemos editar el archivo build.gradle del módulo que acabamos de importar, haciendo que coincidan las versiones con el build.gradle de la aplicación. El archivo debe de quedar similar a la Figura 36.

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

android {
    compileSdkVersion 25
    buildToolsVersion "27.0.3"

    defaultConfig {
        minSdkVersion 21
        targetSdkVersion 25
    }

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

Figura 36: Gradle del módulo OpenCV.

- También debemos añadir el módulo a las dependencias del proyecto. Desde el menú principal, seleccionamos File -> Project Structure - > app -> Dependencies. En este punto, pulsamos sobre el botón para añadir e indicamos Module Dependency, seleccionamos el módulo de OpenCV y aceptamos.

- Por último, debemos crear la carpeta jniLibs en el proyecto, quedando de tal manera ...\app\src\main\jniLibs. Una vez creada la carpeta, debemos copiar el contenido de \install\sdk\native\libs en la nueva carpeta. Estos archivos son necesarios para que funcione correctamente en dispositivos con esas arquitecturas de procesador, solo nos interesa los archivos .so, podemos eliminar el resto.

9.2 Guía de usuario

En la ventana de carga se le solicitarán los permisos necesarios para poder utilizar la aplicación. Además de descargarán los marcadores necesarios para poder hacer uso de todas las funciones de la aplicación y se cargará la configuración por defecto. En la Figura 37 podemos observar una imagen de esta ventana.

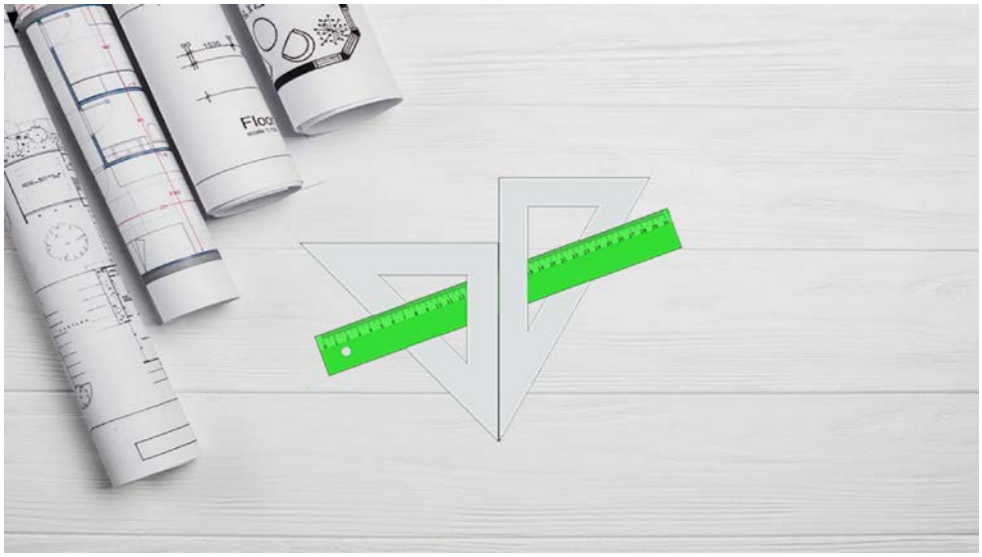

Figura 37: Ventana de carga

Si es la primera vez que iniciamos la aplicación aparece la ventana de calibrado, pero con un botón que permite visualizar la ventana de más información, debido a que al no acceder desde la ventana principal no dispondríamos de esta opción.

En cambio, si no es la primera vez que iniciamos la aplicación, tras la ventana de carga, accederemos a la ventana principal. En dicha ventana se encuentran las principales funciones: comprobación de alineamiento y paralelismo medición de distancia, perímetro y superficie. Vienen agrupadas en tarjetas, que contienen el título de la función que desempeñan y una descripción de ella. Además, la tarjeta posee un botón superior el cual nos llevará a una ventana con más información sobre el funcionamiento, como que marcadores necesitaremos o consejos que le pueden ser de utilidad. Con las fechas, que se encuentran en encima y debajo de la tarjeta, podremos cambiar entre las distintas funciones principales. Abajo se indica el número de página en la cual nos encontramos. Si deslizamos el dedo de izquierda a derecha, pasaremos a la página siguiente, la cual contiene las funciones de ayuda: recalibrar, validar calibrado y descargar marcadores. Está formada por los mismos componentes que la ventana principal. En la Figura 38 se muestra ambas páginas de la ventana principal, con la función de sus diversos componentes.

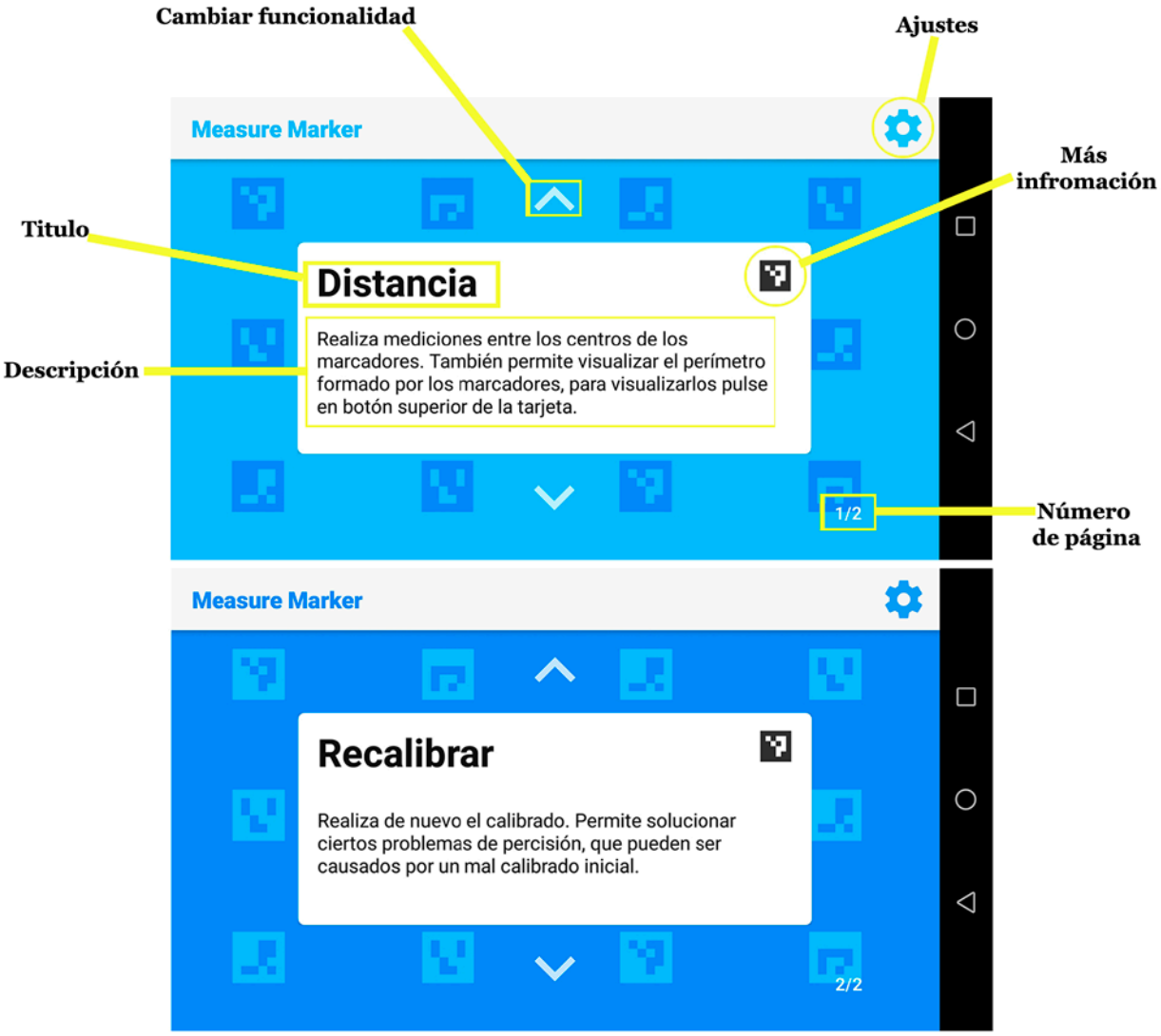

Figura 38: Ventana principal.

Si accedemos a la ventana de más información, podremos observar las imágenes de los marcadores que necesitaremos para desempeñar esa función, con los botones laterales podremos recorrer cada una de ellas. Su identificador está ubicado debajo de la imagen y nos será de utilidad en la ventana de descargas, si deseamos descargar un marcador en concreto. Desde esta misma ventana podemos visualizar una lista de consejos para el desempeño de esa función, que nos permiten comprender mejor el uso de la aplicación y solucionar posibles dudas que les surjan durante el uso de esta. En la Figura 39 podemos visualizar está ventana con las dos opciones.

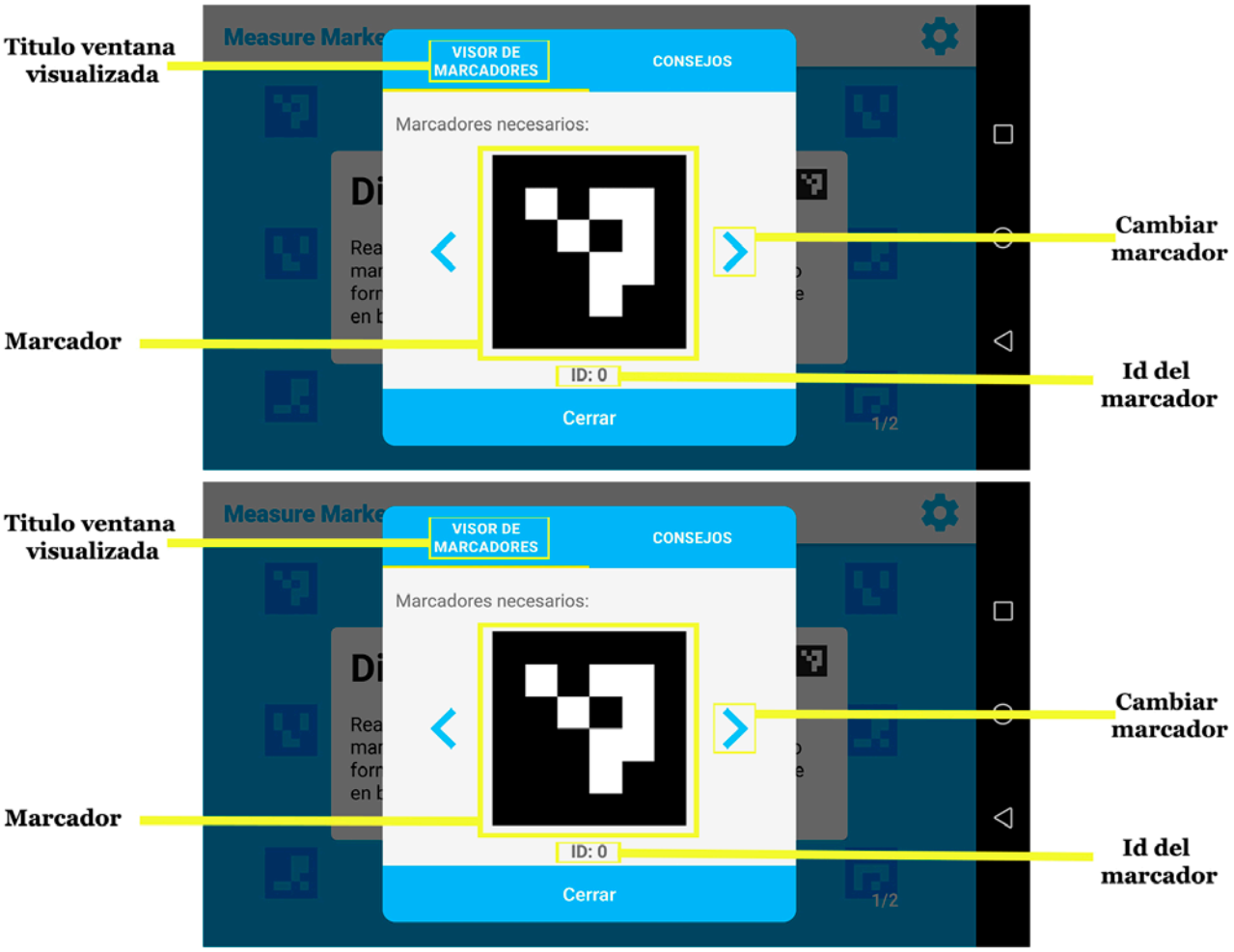

Figura 39: Ventana más información.

Desde la barra superior podremos acceder a la ventana de ajustes, la cual posibilita realizar una configuración de nuestras preferencias generales: como el tamaño de los marcadores que vamos a emplear y la precisión con la cual queremos que se nos muestren los resultados. También nos permite seleccionar la resolución, que emplearemos en las funciones hacen uso de la cámara, y nos permite elegir el idioma que deseamos utilizar, entre tres opciones: español, inglés y francés. Además, desde la configuración general, si pulsamos el botón de información podremos visualizar un empleo de cómo se realiza la medición correcta del tamaño de la marca, debido a que se necesitaba eliminar cualquier duda sobre ello, ya que el valor introducido en este apartado es de gran importancia, pudiendo causar resultados incorrectos en las demás funciones, en caso de ser introducido de forma errónea. En la Figura 40 podemos observar las diferentes opciones de esta ventana.

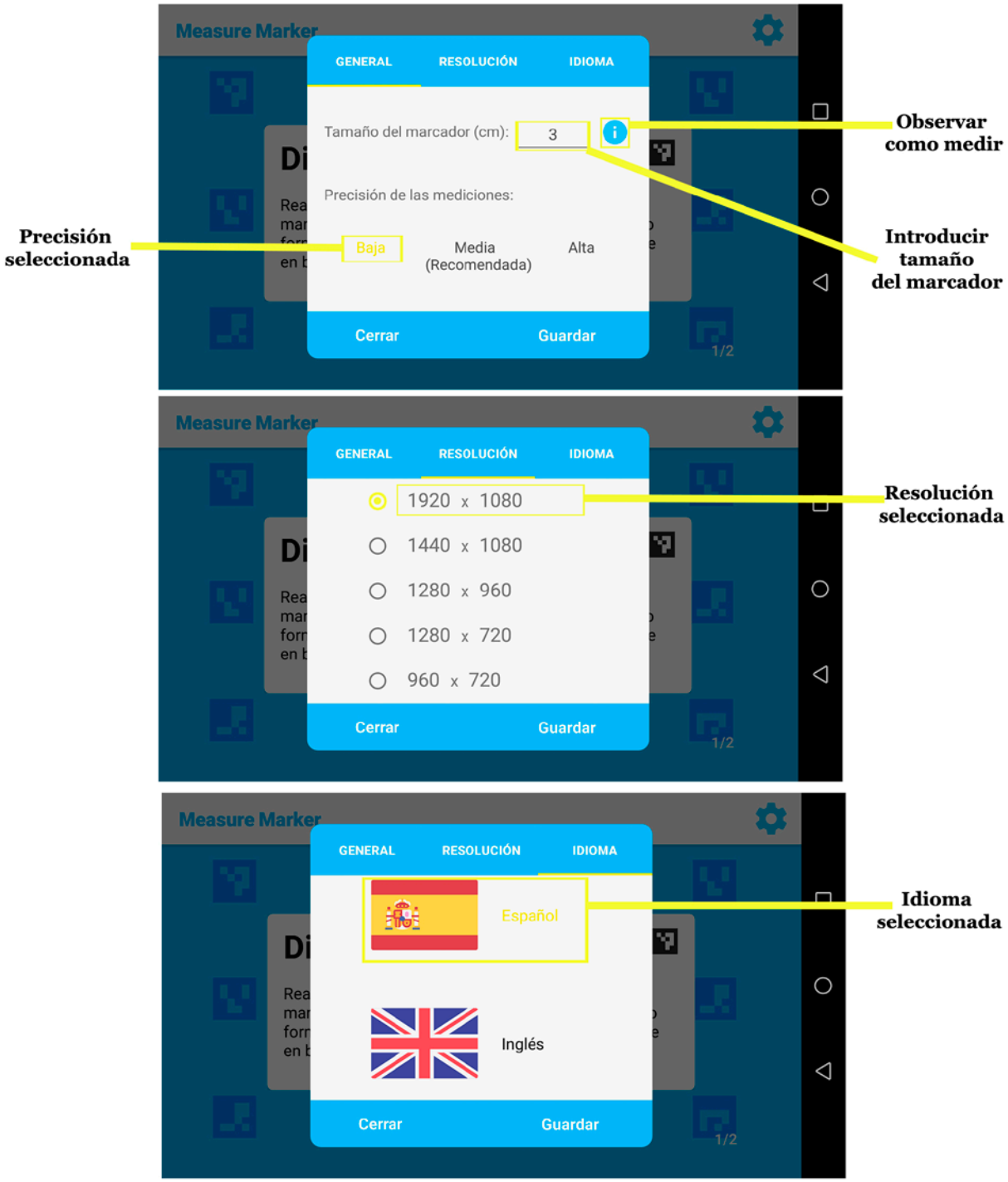

Figura 40: Ventana ajustes.

En la Figura 41 podemos observar la ventana que muestra cómo realizar la medición del marcador.

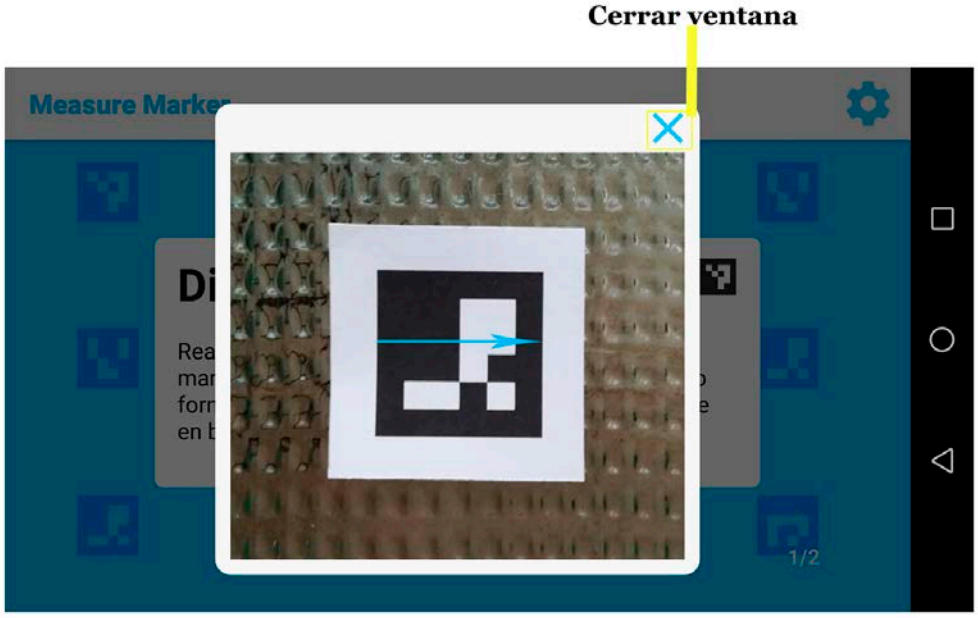

Figura 41: Ventana muestra de medición

La primera función que debemos llevar a cabo es el calibrado, para obtener los parámetros de la cámara, posteriormente disponemos de la posibilidad de realizar el recalibrado, desde la página de funciones de ayuda, de la ventana principal. La ventana de calibrado dispone de un contador de capturas en la parte superior, es necesario realizar quince para llevar a cabo el cálculo de los parámetros de la cámara, y un botón en la parte inferior para realizar las capturas. Una vez realizado el número necesario de capturas, aparecerá un cartel con la información del calibrado. La Figura 42 se puede observar la ventana de calibrado y de cómo se muestran los resultados.

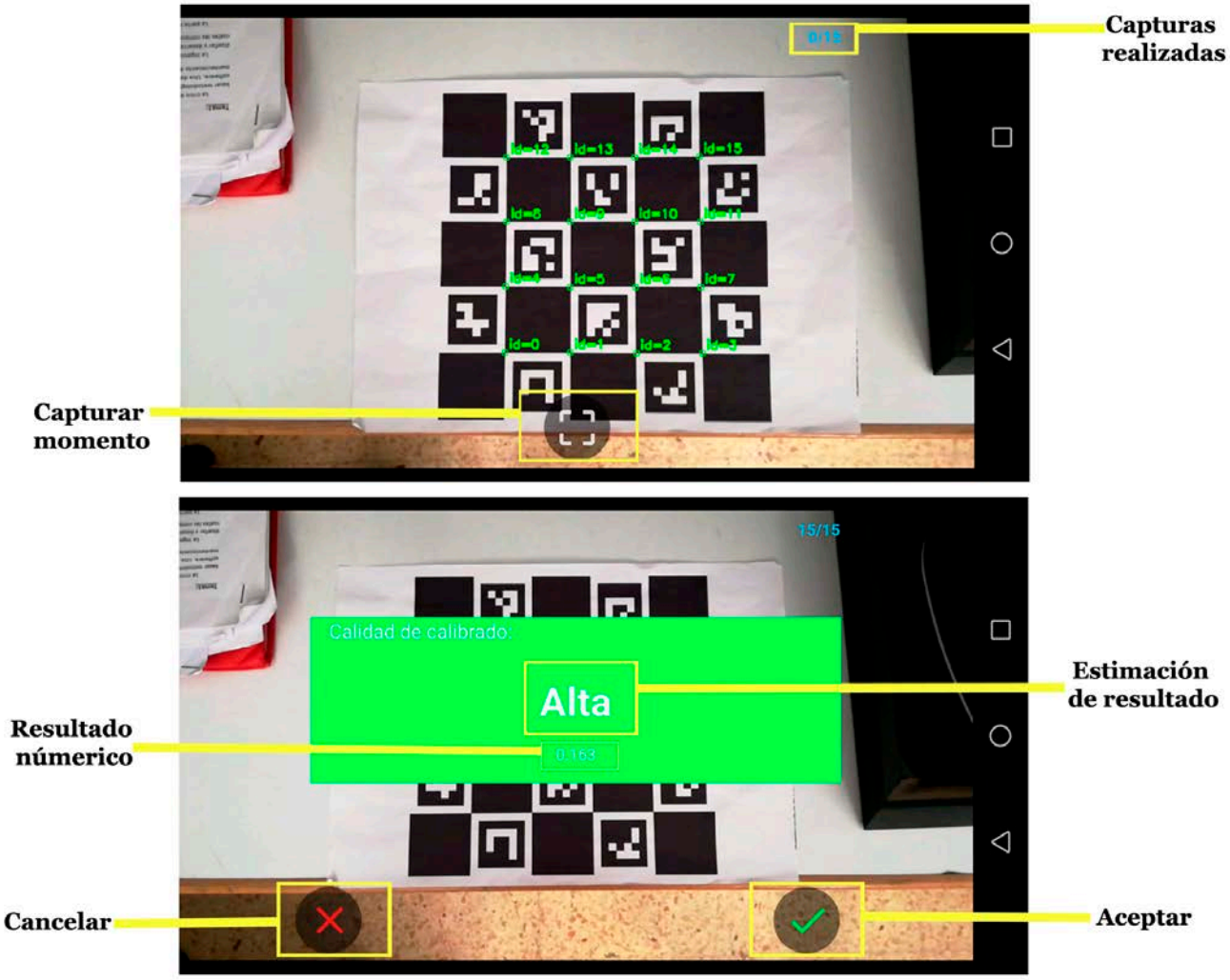

Figura 42: Ventana calibrado.

Tras realizar el calibrado es necesario comprobar que en realidad se ha realizado correctamente, para ello se dispone de la función validar calibrado, la cual proyectará ejes de coordenadas sobre los marcadores, podemos emplear cualquiera de los marcadores o el tablero. Debemos comprobar que los ejes se dibujan sin desmontarse. Si se dibujan correctamente, el calibrado ha sido un éxito. En caso contrario, debemos llevar a cabo el recalibrado y volver a validar el resultado en esta ventana. En la Figura 43 podemos visualizar un ejemplo de esta ventana.

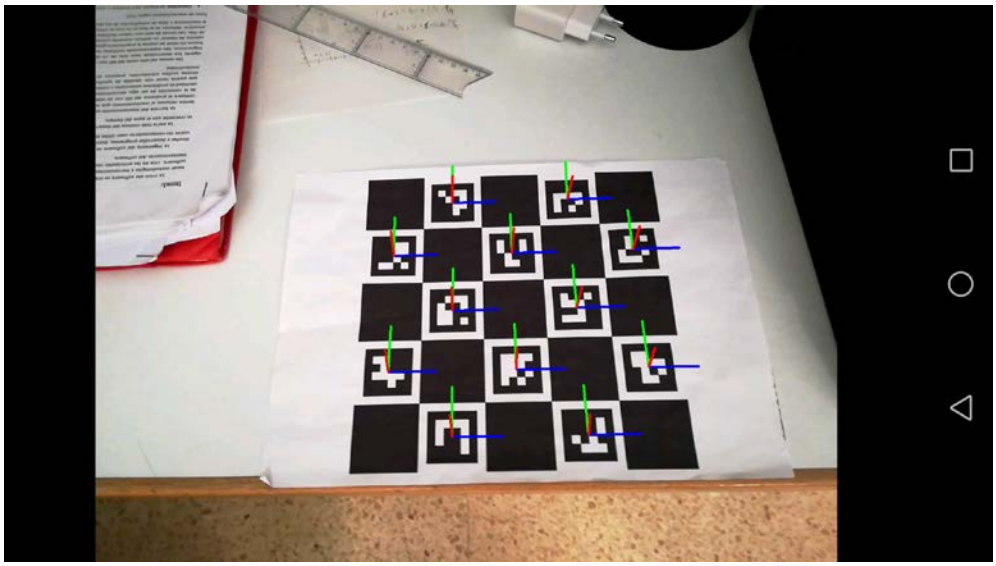

Figura 43: Ventana validar calibrado.

Una vez realizado el calibrado inicial, tendremos acceso al resto de funciones de la aplicación, desde la ventana principal.

La función de paralelismo comprueba si dos objetos están alineados o en posición paralela, uno respecto al otro. Si lo están la línea que enlaza ambas marcas cambiará de color, de amarillo a verde. Además, disponemos de la posibilidad de habilitar el sonido, para aquellos casos en los que se tenga la opción de colocar el dispositivo en una zona estable mientras movemos los marcadores, realizando un sonido cuando estos estén alineados o en paralelo. Para llevar a cabo esta función es necesario emplear cuatro marcadores, dos para el objeto de referencia, aparecen marcadas con un icono, y dos marcas para el objeto deseado. No es necesario que sean dos objetos diferentes, podemos emplearlo en los extremos de un mismo objeto, por ejemplo, para comprobar que al recortar un papel no nos hemos desviado. En la Figura 44 podemos visualizar esta ventana.

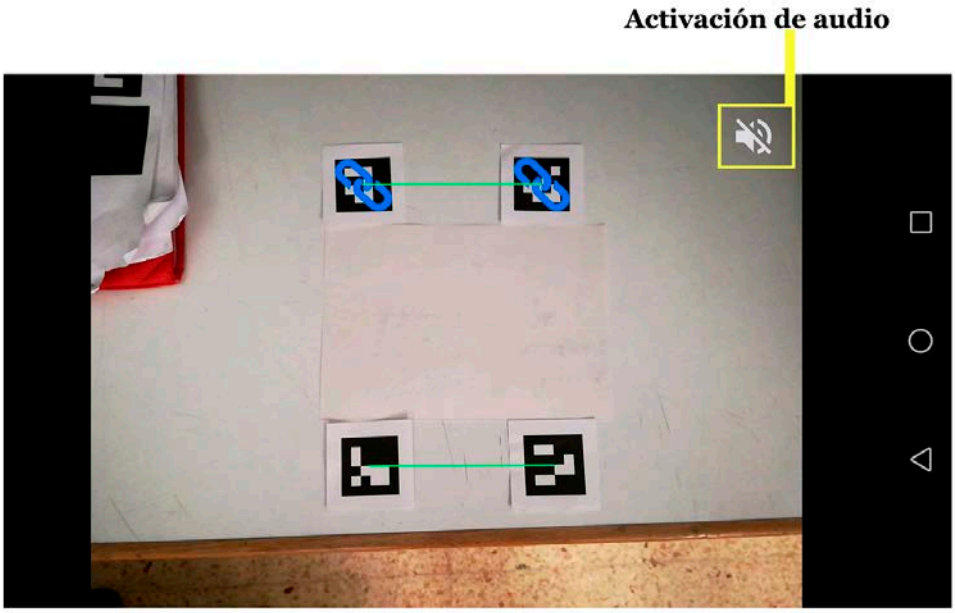

Figura 44: Ventana de paralelismo.

En la ventana de distancia podemos obtener tanto la distancia individual de cada línea, que conecta los centros de los marcadores en pantalla, como el perímetro total de la figura formada por los marcadores. Para activar este modo se dispone de un botón en la parte superior de la ventana. En esta función se pueden emplear hasta cuatro marcadores, pero no es necesario emplear todos ellos, con usar más de un marcador ya obtendremos los valores de distancia entre ellos, posibilitando emplear cualquiera de los cuatro marcadores indicados en su ventana de más información. Los números encima de los marcadores son su orden, es decir, las líneas se irán dibujando desde los marcadores con menor número hacía las marcas con mayor número, se emplea para poder hacernos una idea de cómo situarlos. En la Figura 45 podemos observar la ventana de distancia, con el modo perímetro desactivado y activado.

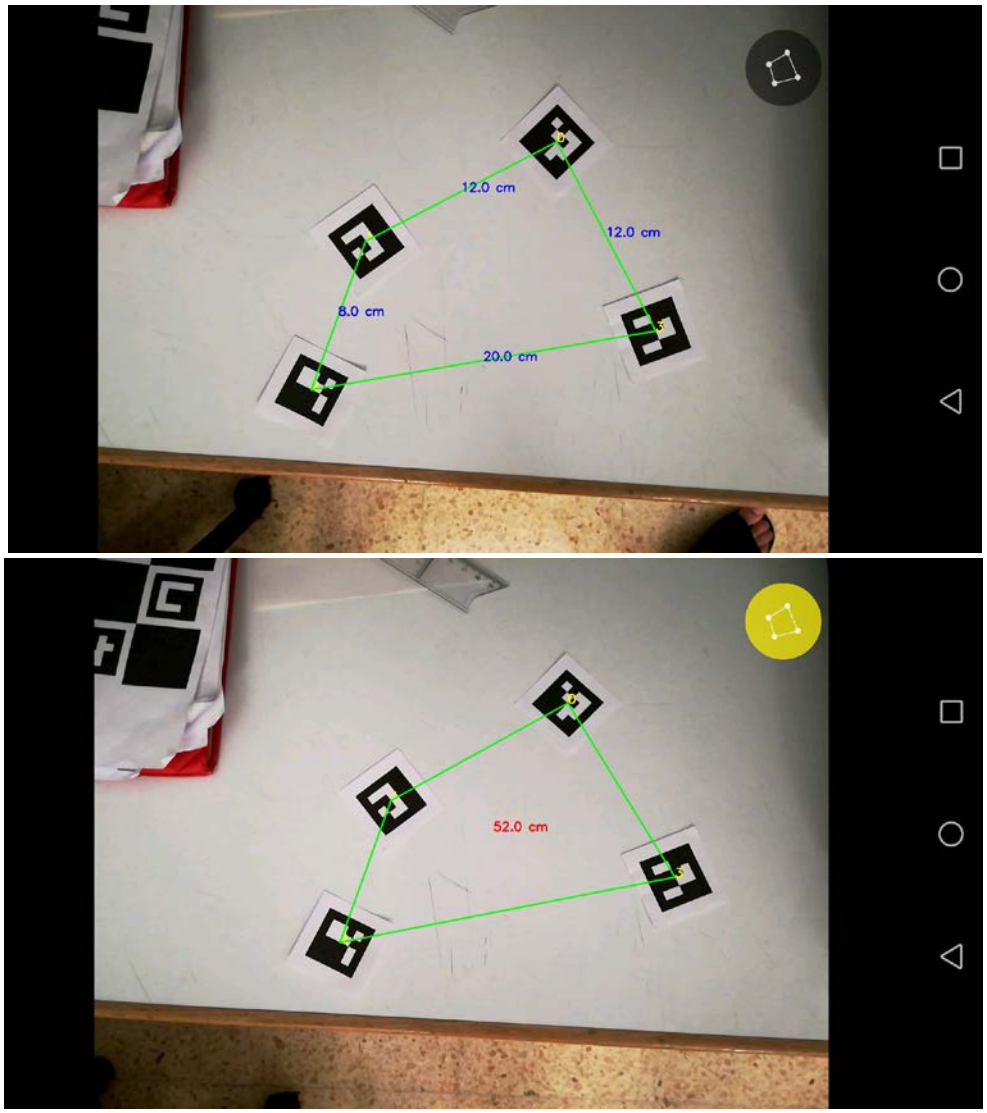

Figura 45: Ventana distancia

La función de calcular superficie nos permite obtener la superficie que posee la figura delimitada por los centros de los marcadores. En este caso podemos emplear hasta 5 marcadores, teniendo también la posibilidad de emplear una menor cantidad, pero utilizando como mínimo tres de ellos. Para obtener resultados correctos la figura que formemos debe de ser un polígono simple. Podemos observar esta ventana y el comportamiento al retirar uno de los marcadores en la Figura 46.

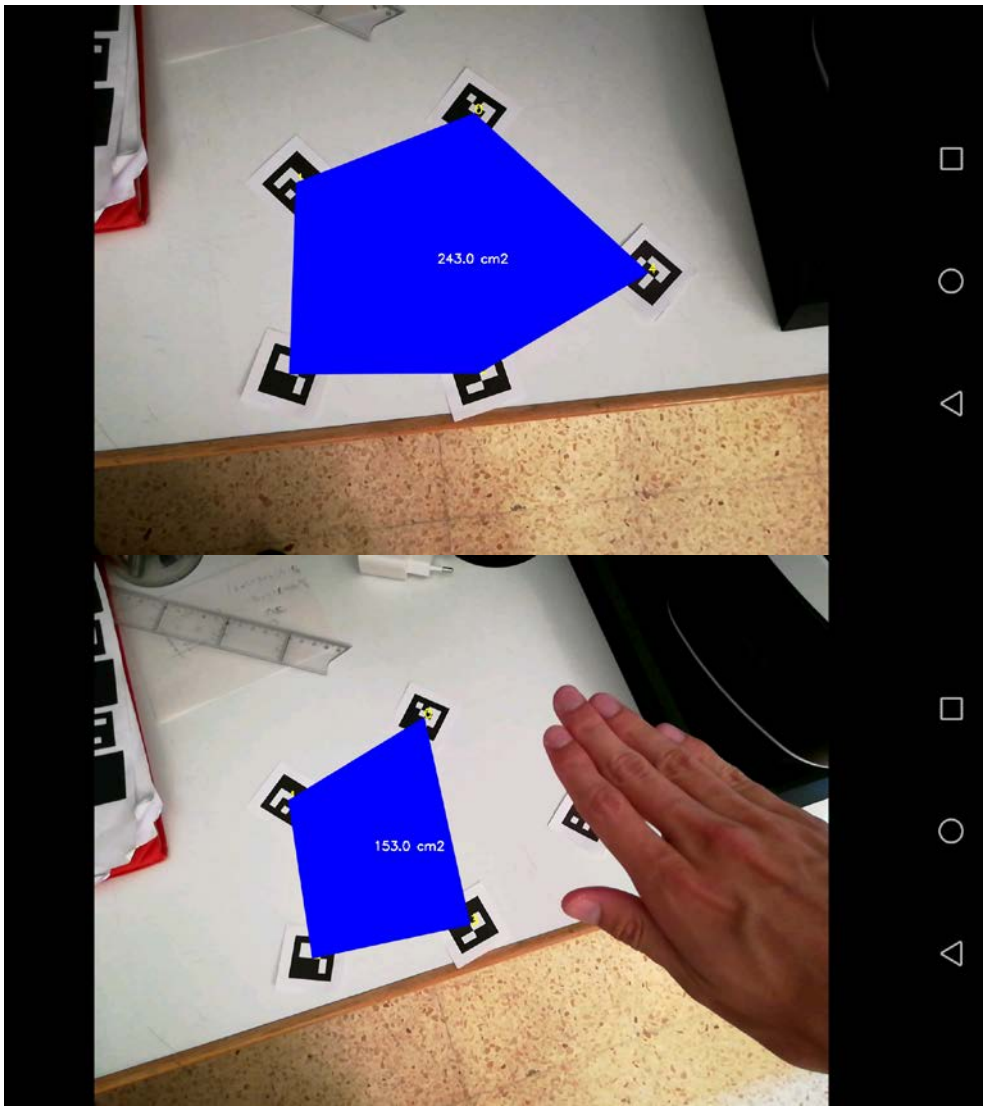

Figura 46: Ventana de superficie.

Por último, disponemos de la ventana de descarga, la cual posibilita descargar de nuevo los marcadores. Podemos descargar el que deseemos introduciendo su identificador o directamente descargar todos ellos. También se aporta un enlace a Google Drive, para prevenir posibles incidencias con las otras opciones o si se desea apuntar en enlace al que direcciona y descargarlas desde el ordenador. En la Figura 47 podemos observar esta ventana.

Figura 47: Ventana descargas.

10. Glosario

Las definiciones de este glosario han sido obtenidas de la enciclopedia de contenido libre, Wikipedia.

Brainstorm: es una herramienta de trabajo grupal que facilita el surgimiento de nuevas ideas sobre un tema o problemática determinada. La principal regla de este método es suspender o aplazar el juicio, ya que en un principio toda idea es válida y ninguna debe ser rechazada.

Código abierto: Es un modelo de desarrollo de *software* basado en la colaboración abierta. Enfocándose más en los beneficios prácticos (acceso al código fuente) que en cuestiones éticas o de libertad que tanto se destacan en el *software* libre.

Fiduciario: objeto utilizado para la observación de sistemas de imágenes, el cual aparece en la imagen para ser usado como punto de referencia o de medida. Además, puede ser ubicado como una marca o grupo de marcas en un instrumento óptico.

Java Wrapper: software que permite que ciertas clases trabajen juntas, lo que no sería posible de otra forma.

Librería: conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que invoca. El comportamiento que implementa una librería no espera ser utilizada de forma autónoma, sino que su fin es ser utilizada por otros programas.

Licencia Apache: es una licencia de *software* libre permisiva. No es una licencia copyleft, ya que no requiere la redistribución del código fuente cuando se distribuyen versiones modificadas.

Licencia BSD: es una licencia de *software* libre permisiva como la licencia de OpenSSL o la MIT License. Esta licencia tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre.

Licencia GNU GPL: es una licencia de derecho de autor ampliamente usada en el mundo del software libre y código abierto, y garantiza a los usuarios finales la libertad de usar, estudiar, compartir y modificar el *software*. Declara que el software cubierto por esta licencia es libre, y lo protege, mediante *copyleft*, de intentos de apropiación que restrinjan esas libertades a nuevos usuarios cada vez que la obra es distribuida, modificada o ampliada.

Licencia MIT: esta licencia es una Licencia de software libre permisiva. En cuanto a efectos es muy parecida a la licencia BSD.

Machine learning: es una variante dentro de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que

permitan que las computadoras aprendan. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos.

Odometría: es el estudio de la estimación de la posición de vehículos con ruedas durante la navegación. Para realizar esta estimación se usa información sobre la rotación de las ruedas para detectar cambios en la posición a lo largo del tiempo.

SceneKit: combina un motor de renderizado de alto rendimiento con una API para importar, manipular y renderizar objetos de tres dimensiones.

SDK: un kit de desarrollo de *software* es generalmente un conjunto de herramientas de desarrollo de *software* que le permiten al programador o desarrollador de *software* crear una aplicación informática para un sistema concreto.

SprintKit: es una infraestructura de renderizado y animación de gráficos que puedes utilizar para animar *sprites*.

Unity: Motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, OS X y Linux.

UWP: plataforma universal de Windows, es una plataforma común de aplicaciones presentes en todos los dispositivos que cuentan con Windows 10 y sus variantes.

Ventana modal: es un elemento de control gráfico subordinado a la ventana principal de una aplicación.