UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Aplicación móvil para la captura guiada de imágenes de un objeto previamente modelado

Adrián Aranda Rebollo
Tutor: Jesús Bescós Cano

Junio 2019

Aplicación móvil para la captura guiada de imágenes de un objeto previamente modelado

AUTOR: Adrián Aranda Rebollo
TUTOR: Jesús Bescós Cano

Dpto. Tecnología Electrónica y de Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2019

Resumen

Cada vez más los sistemas de visión por computador están acaparando más protagonismo en nuestro día a día. Durante los últimos tiempos, gracias a los avances en tecnologías, se ha convertido en una de las disciplinas más importantes y con mayor desarrollo en el mundo. Encontramos este tipo de sistemas en aplicaciones militares, aplicaciones en seguridad de detección de movimiento, en el mundo de la industria como en controles de calidad, seguridad en medios de transporte, etc.

Por otra parte, los "smartphones" o teléfonos inteligentes han ido desempeñando, cada vez en mayor medida, un papel fundamental en nuestras vidas, y el uso que le damos a estos dispositivos los ha convertido en verdaderos asistentes personales capaces de simplificar las tareas realizadas en nuestro día a día.

Es por esto que únicamente era cuestión de tiempo que ambos campos se uniesen y que los dispositivos inteligentes comenzasen a ofrecer aplicaciones con funcionalidades de visión por computador que hiciesen uso de la cámara para obtener información de nuestro alrededor.

Es en este contexto de aplicaciones emergentes donde se desarrolla este trabajo de fin de grado, un campo poco explorado, con mucho potencial pero igualmente con mucho trabajo por delante.

Este trabajo se centra en la creación de una app centrada en guiar a un usuario a la captura de imágenes de un objeto en concreto. Dicho objeto se trata de un recipiente translúcido lleno de "cosas". Para su implementación, se hará uso de los algoritmos proporcionados por las librerías de OpenCV permitiendo desarrollar un algoritmo básico de detección de objetos y explorando las posibilidades del uso de la cámara.

Abstract

Computer Vision Systems are gaining prominence in our daily lives. During recent times, thanks to technological advances, it has become one of the most important and most developed disciplines in the world. We find these systems in military applications, security applications for motion detection or intrusion, industrial applications such as quality control, security in transport, etc.

On the other hand, smartphones have also acquired an increasingly important role in our lives, and the use we make of these devices has transformed them into personal assistants capable of simplifying our everyday tasks.

This is why it was only a matter of time before both fields unified and smartphones began to offer applications with computer vision features using the camera to obtain information about our surroundings. This project is carried out in this context of emerging applications, a field that is little explored, but has a lot of potential, as well as a lot of work ahead.

This paper focuses on the creation of an app capable of detecting a transparent container full of "things" by using the camera integrated in Android smartphones. For that purpose, the algorithms provided by OpenCV libraries will be used to develop a basic object detection algorithm and to explore the possibilities the camera offers.

Palabras clave

Android, aplicación, teléfono inteligente, visión por computador, máscara, imagen guía, entorno de trabajo.

Keywords

Android, application, smartphone, computer visión, mask, guide image, framework.

Agradecimientos

Agradecer a mis padres ante todo y a Jesús Bescós por prestarme su ayuda en todos estos años de carrera cuando la he necesitado.

Agradecer también a mi hermana, a Maricela por aguantarme sobretodo el último año y especial agradecimiento a Yolanda.

ÍNDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

En la actualidad, somos testigos de una auténtica revolución tecnológica. Gracias al avance en las comunicaciones durante los últimos tiempos de la historia reciente, se han ido desarrollando dispositivos, cada vez más sofisticados, que impulsan el crecimiento y desarrollo de esta materia.

La aparición de los teléfonos inteligentes en nuestras vidas (conocidos también como "Smartphones", palabra inglesa), han contribuido no solamente a que podamos comunicarnos entre nosotros de un modo más directo, sino que también han simplificado muchas de las tareas que realizamos en nuestro quehacer cotidiano.

La continua mejora en las especificaciones técnicas de estos dispositivos permite realizar actividades cada vez más complejas. Uno de los campos que se encuentra actualmente en auge lo representa el de los sistemas de visión por computador (CV) y de realidad aumentada. Este tipo de sistemas capta imágenes del exterior a través de los sensores ópticos del dispositivo electrónico y las procesa con el fin de obtener información más detallada del mundo que nos rodea. Este aspecto, lógicamente, proyecta un gran potencial de perfeccionamiento el cuál suscita un importante interés por parte de los desarrolladores de aplicaciones, quienes no han tardado en crear y compartir una serie de herramientas informáticas orientadas hacia el trabajo con sistemas de visión por computador y realidad aumentada.

Una de las premisas de los programadores para el desarrollo y crecimiento de las aplicaciones se centra en que los recursos puedan ser lo más fácilmente accesibles para aquellas personas interesadas en su mejora. Es por esto que OpenCV (Open SourceComputerVision) pone a disposición una serie de librerías que contienen cierto tipo de clases y algoritmos para su uso específico en la visión por computador y la realidad aumentada. Siguiendo con esta idea de máxima accesibilidad, Android, con más de 2000 millones de usuarios repartidos en todo el mundo (según datos de Google en 2017) es el sistema operativo más utilizado por los usuarios de dispositivos móviles junto con IOS (de Apple). La gran ventaja de las librerías antes citadas y del sistema operativo Android radica en que son proyectos de código abierto, por lo que se encuentran en constante crecimiento y tienen detrás una gran comunidad de usuarios y programadores muy activos en el intercambio de información para impulsar el progreso de ambos proyectos.

Uno de los espacios donde se están probando y creando nuevas técnicas y métodos de visión por computador lo encontramos en el de reconocimiento de objetos. Es en este marco de investigación en el que se encuentra el Trabajo de Fin de Grado que esta memoria detalla.

1.2 Objetivos

Para comenzar, es necesario destacar el marco donde se sitúa este Trabajo de Fin de Grado, y es que este TFG queda dentro de un proyecto más amplio impulsado por la Universidad Autónoma de Madrid, que tiene como propósito realizar una aplicación para dispositivos inteligentes que permita al usuario conocer el nivel de llenado de un recipiente translúcido de estructura conocida haciendo uso de su cámara integrada.

En dicho proyecto se distinguen dos tareas: una es la de detectar el recipiente deseado para captar una imagen de él en las mejores condiciones tanto de orientación como de contraste con la cámara del dispositivo; y la otra es la que atañe al trabajo de cuantificación del nivel de llenado del recipiente a partir de su imagen dada y, de este modo, comunicárselo al usuario.

La labor que a este TFG le corresponde está relacionada con la primera tarea ya citada. Específicamente, se pretende poner en marcha una aplicación móvil orientada al sistema operativo Android, en donde se guíe al usuario hacia la captura de imágenes de un objeto concreto. Para ello será necesario establecer un modelo previo del objeto y a partir de éste, junto con un análisis básico de la imagen enfocada, se creará un proceso interactivo, dando indicaciones gráficas al usuario, hasta obtener la imagen deseada. No se ha pretendido desarrollar una aplicación destinada a un usuario final, sino que con la elaboración de este trabajo se ha buscado asentar unas bases de diseño y procedimiento que permitan en futuros trabajos ampliar y optimizar los resultados obtenidos para alcanzar los propósitos del proyecto.

Por lo tanto, el objetivo específico perseguido por este TFG, en esencia, ha sido la de verificar la viabilidad de la aplicación. Para ello, lo que se buscaba desde el primer momento era familiarizarse con el entorno de programación de aplicaciones Android y las herramientas y funciones de las que se disponen para llevar a cabo la aplicación únicamente a un nivel funcional, así como elaborar una serie de etapas con el fin de obtener un modelo del objeto en cuestión que sea usado como referencia para su detección. Y que todo ello quedara documentado y reflejado en la memoria que este TFG detalla, además de la entrega del código de la aplicación al tutor de este TFG para, en un futuro, continuar con el desarrollo de esta parte del proyecto desde este punto.

1.3 Organización de la memoria

Para exponer el trabajo desarrollado en este Trabajo de Fin de Grado se seguirá la siguiente estructura:

•Capítulo 1: Introducción.
•Capítulo 2: Tecnologías de base.
•Capítulo 3: Descripción del sistema.
•Capítulo 4: Implementación.
•Capítulo 5: Evaluación: pruebas, resultados y limitaciones.
•Capítulo 6: Conclusiones y trabajo futuro.

2 Tecnologías de base

Para una correcta identificación del marco tecnológico en que nos encontramos, a continuación se expondrán una serie de herramientas y disciplinas puestas en práctica a lo largo del desarrollo de la aplicación que servirán para situarse en la misma línea de trabajo que el proyecto pretende desempeñar.

2.1 Android

Android es un sistema operativo y una plataforma software basado en el kernel de Linux. Se trata de una plataforma de código abierto, lo que permite a fabricantes, operadores y desarrolladores dar mayor funcionalidad a sus dispositivos integrados con este sistema operativo.

Fue creado en el año 2003 por la empresa Android Inc. y la intención inicial era desarrollar un sistema operativo avanzado para cámaras digitales. Sin embargo, la Compañia se dio cuenta de que no era un mercado lo suficientemente amplio y redirigió sus esfuerzos en elaborar un software que pudiera competir con otros sistemas operativos para teléfonos móviles de esos años. En 2005, percatándose de su gran potencial, la citada empresa fue comprada por Google, que integró Android como sistema operativo en sus primeros teléfonos móviles, extremo que ha contribuido a aumentar su importancia en el mundo de la tecnología. En la actualidad, podemos encontrarnos este sistema operativo en infinidad de dispositivos electrónicos: tablets, relojes, coches, auriculares, portátiles, televisores…
Para entender un poco más este sistema operativo y su funcionamiento, analizaré de forma breve su estructura y los niveles en los que se compone:

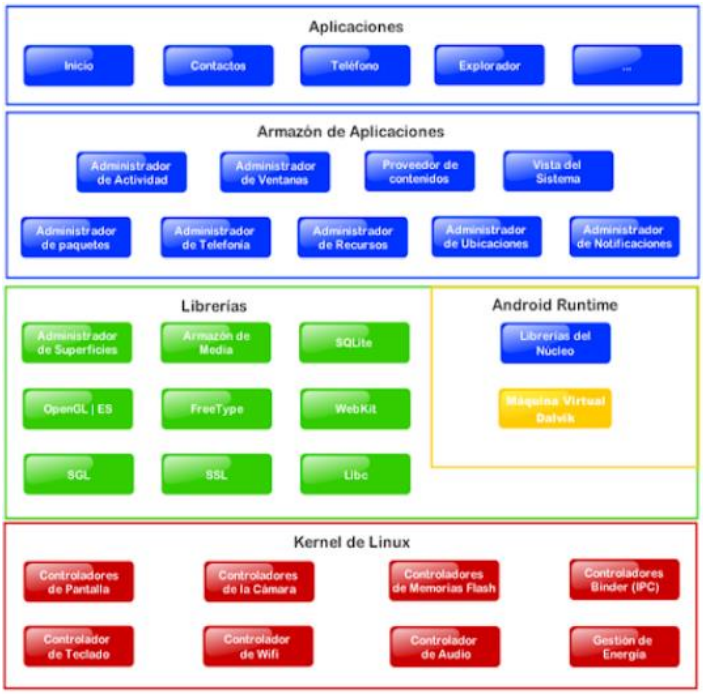

Figura 2-1: Estructura Android.

• **Aplicaciones:**
Este nivel contiene tanto las aplicaciones por defecto incluidas en Android como aquellas desarrolladas por terceros o por el propio usuario.

• **Armazón de Aplicaciones ("Framework" de Aplicaciones):**
Representa el conjunto de herramientas de desarrollo de cualquier aplicación necesarias para administrar su ciclo de vida, recursos, paquetes, sensores y otros complementos necesarios para su funcionamiento. Toda aplicación desarrollada para Android utiliza el mismo conjunto de APIs ("ApplicationProgramming Interface") y el mismo "framework", representado por este nivel.

• **Librerías nativas de Android:**
Se corresponde con las Llibrerías de Android que han sido escritas usando los lenguajes de programación C/C++ y accesibles a ellas a través del uso de las Interfaces Nativas de Java (JNI).

• **Tiempo de ejecución de Android:**
Se sitúan en el mismo nivel que las Librerías. Nos encontramos con la máquina virtual ART ("Android Runtime") que ha sustituido recientemente a su predecesora Dalvik. Diseñada para reducir la memoria utilizada por las aplicaciones y poder ejecutar varias instancias de la máquina virtual.

• **Núcleo:**
Android usa el kernel (núcleo) de Linux como una capa de abstracción para el hardware disponible en los dispositivos móviles. Su cometido es el de administrar controladores, memoria, procesos y redes.

2.2 OpenCV

OpenCV se define como una librería de código libre desarrollada originalmente por la empresa tecnológica Intel y orientada específicamente a la visión por computador (CV), o visión artificial, en tiempo real y al "Machine Learning" (aprendizaje automático).

Su objetivo fundamental se centra en proporcionar un marco de trabajo común para las aplicaciones ligadas a este ámbito y acelerar el uso de los sistemas de percepción por computador en sistemas comerciales.

Esta librería dispone de más de 2500 algoritmos optimizados que pueden ser usados para detectar y reconocer rostros, seguimiento de objetos, extracción de modelos 3D, mejora en la resolución de imágenes de una escena, comparación de imágenes en base de datos, etcétera. Todo ello continúa desarrollándose con el avance de la tecnología, las cámaras y las tarjetas de procesamiento.

Asimismo, hay que añadir que es una librería multiplataforma compatible con sistemas Linux, Windows, Android... y puede ser implementada desde múltiples lenguajes de programación como C/C++, Python, Java y MATLAB.

2.3 Matlab

Matlab se configura como de un software de ámbito matemático que ofrece su propio entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M).
Este lenguaje de programación fue desarrollado en 1970, con la idea de proporcionar un acceso sencillo al software de matrices LINPACK y EISPACK y ello sin la necesidad de usar otro lenguaje de programación común en ese momento para este tipo de tareas como era el Fortran. El programa, propiamente dicho, fue creado en 1984 por Cleve Moler con la finalidad de crear subrutinas escritas en Fortran en los cursos de álgebra lineal y análisis numérico obviando crear programas en dicho lenguaje.
Se trata de un software muy común en universidades y centros de investigación y desarrollo. Entre sus prestaciones básicas, se hallan la manipulación de matrices, representación de datos y funciones, implementación de algoritmos…además de otras muchas funcionalidades extra que ayudan a mejorar sus prestaciones como son: Simulink (plataforma de simulación multidominio), GUIDE (editor de interfaces de usuario) ó la capacidad de trabajar con cajas de herramientas (tollboxes).
Merced a las anteriores características, Matlab es, en la actualidad, un software muy potente con aplicaciones en muchos ámbitos: sistemas de control, matemáticas, estadística y optimización, biología computacional, finanzas computacionales…y la que más nos interesa: procesamiento de señales y comunicaciones y visión artificial.

2.4 Visión por computador

La visión por computador o visión artificial se configura como la disciplina que integra métodos para obtener, procesar, analizar y entender las imágenes (2D) captadas del mundo real (3D), ello con el propósito de producir información no visual que pueda ser manejada por un ordenador. De esta manera, se pretende modelar el sistema visual humano permitiendo automatizar una amplia gama de tareas gracias al aporte de información que necesitan las máquinas para la toma de decisiones correctas en cada tarea asignada.
Se han ido perfeccionando las técnicas de visión por computador a lo largo de los años. Los primeros sistemas prácticos se basaban en imágenes binarias (blanco y negro) que se procesaban por bloques, ventanas o píxeles y más adelante se introdujeron sistemas de intensidad de gris.

El optimismo inicial que suscitó esta disciplina se vio truncado por el desfase con las tecnologías. No fue hasta la década de los 90 del siglo pasado cuando se desarrolla la visión por computador moderna. Por estas fechas, crecen los algoritmos de detección y extracción de características y se desarrollan técnicas de aprendizaje automático. Estos hechos, junto con el gran avance en las tecnologías que posibilitan una gran capacidad de procesamiento de la información, han dado lugar a que se puedan resolver problemas complejos dentro de distintos ámbitos: militar, de sistemas de seguridad, industria automovilística, aeronáutica, o en entornos industriales, tanto a nivel de manufacturación como de revisión de calidad, por ejemplo.

En el ámbito de la visión por computador, la detección de objetos constituye uno de los temas punteros. El problema es simple: dada una imagen, queremos ser capaces de detectar

de manera automática un objeto en ella, como una silla, un libro o un computador. Para un ser humano se trata de una tarea sencilla pero para un ordenador no lo resulta.
Para abordar la anterior cuestión, hay que pensar la manera en la que queda codificada una imagen digital. Para una máquina, una imagen solo está constituida por cajas tridimensionales rellenas de números, en donde cada píxel queda representado por la combinación de tres valores de color: rojo, verde y azul. De este modo, cuando una máquina indaga un objeto dentro de una imagen, su trabajo se orienta en buscar patrones de píxeles que representan el objeto en particular dentro de la imagen. El reconocimiento de dichos patrones se convierte en una tarea realmente ardua y compleja, ya que hay que tener en cuenta varios factores como son la perspectiva, la iluminación, la formación de oclusiones parciales o la formación de falsas sombras o reflejos que producen pérdidas de información.

Para las personas, es muy sencillo detectar objetos en las imágenes, ya que nuestro sistema visual usa mucha más información proporcionada por la propia imagen, como por ejemplo el contexto en la que fue tomada. Sin embargo, para una máquina entender estas escenas del modo en que lo realizan las personas en la actualidad, es algo totalmente imposible y ello porque todavía no están dotadas de esta capacidad de interpretación unida al género humano.
A pesar de lo expuesto, existe un gran optimismo en cuanto al futuro de la visión por computador. A medida que se trabaja en ese campo, se implementan mejoras en los algoritmos de detección y merced al avance en las tecnologías relacionadas con el procesamiento de información, en los próximos años, se asentarán cada vez más sistemas de visión artificial en nuestro entorno.

2.5 Tratamiento digital de imágenes

El tratamiento digital de señales representa una técnica para manipular matemáticamente una señal de información ya sea para modificarla o mejorarla.

Su objetivo es optimizar el aspecto de las imágenes para destacar o hacer menos evidentes ciertas características que nos puedan interesar. En contraste con la visión por computador, la meta de esta disciplina no se dirige en obtener información de las imágenes, sino en generar una nueva imagen a partir de la original, cuyo resultado será el más adecuado para una aplicación en concreto.

Con las técnicas de tratamiento de imágenes se pueden destacar aspectos de las imágenes captadas para mejorar propiedades que el sistema visual humano posee, por ejemplo: mejora del contraste, el brillo, la calidad. Asimismo, se puede retocar una imagen para favorecer las operaciones que se pueden realizar sobre ellas. Estas operaciones pueden realizarse sobre el dominio de la frecuencia, con la Transformada de Fourier, o en el dominio del espacio, aplicando las operaciones directamente sobre los píxeles.
Este tipo de técnicas tienen relación directa con la visión por computador, ya que resaltando ciertos aspectos de la imagen, se pueden después aplicar algoritmos de visión artificial para extraer información más concreta de la imagen.

3 Descripción del sistema

La realización de este TFG propone la puesta en marcha de una app para dispositivos móviles Android, por lo que en este capítulo se ofrecerá más información sobre dicha app: desde la elección del sistema operativo hasta el planteamiento de su desarrollo pasando por los objetivos de ésta, dando una visión general (los detalles técnicos se explicarán en posteriores capítulos) del modo en que se ha afrontado el desarrollo de la aplicación.

3.1 Elección del software

Una de las primeras dudas que surgen cuando se quiere desarrollar una aplicación para dispositivos móviles se centra en la elección del sistema operativo para los que estará disponible. Esta cuestión adquiere especial importancia ya que no solamente a nivel técnico, sino también a nivel de diseño, nuestra aplicación estará relacionada con el sistema operativo en el que vaya a funcionar.
En el mercado actual de los dispositivos inteligentes, básicamente, podemos encontrarnos "smartphones" con uno de estos dos sistemas operativos: Android o IOS, creados por las grandes empresas tecnológicas Google y Apple respectivamente.

Ambos sistemas operativos ofrecen muy buenas prestaciones para el usuario. Sin embargo, existen grandes diferencias entre ellos lo que implica que, en función de nuestros intereses, nos convenga más uno u otro.

Para la creación de este proyecto, se decidió desarrollar la aplicación en el sistema operativo Android por los motivos que se exponen a continuación:

Una de las pretensiones que busca este proyecto, se enfoca en conseguir que la aplicación llegue al máximo número de usuarios posibles. Android e IOS son los que rigen el mercado actual en este sentido, pero Android le ha "ganado la partida" a IOS, tal y como muestran los datos del mercado de años recientes.

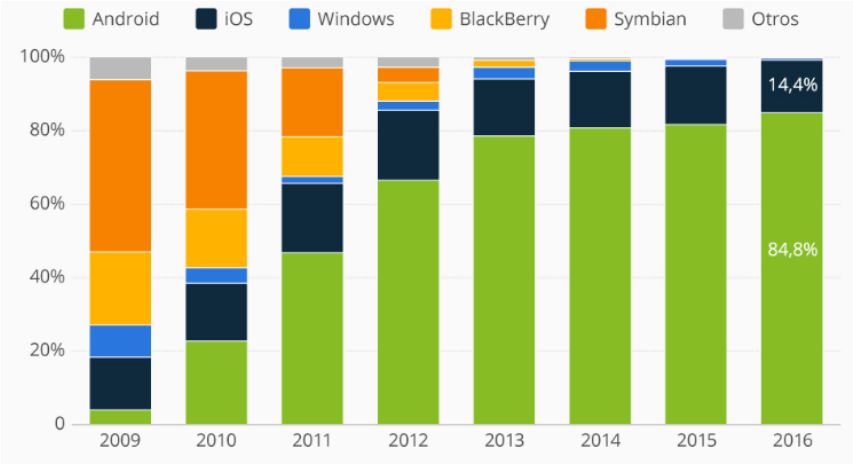

Figura 3-1: Evolución de la distribución de los sistemas operativos en el mercado de "Smartphones" (Fuente: Business Insider).

Por la anterior razón, Android se perfila como una muy buena opción para los intereses del proyecto. Otro de los motivos que, por otro lado, muestra una de las grandes diferencias con IOS, reside en que Android es una plataforma libre y de código abierto, en contraposición con el modelo de desarrollo de software propietario que caracteriza a IOS.

Si queremos realizar pruebas del funcionamiento de nuestra aplicación, con IOS, necesitamos una licencia de desarrollador que alcanza un precio de 90$ y además solo podemos hacer comprobaciones en un dispositivo. Por otra parte, Android ofrece una mayor facilidad a la hora de realizar pruebas con las aplicaciones, ya que pueden realizarse desde cualquier dispositivo sin disponer de una licencia de desarrollador (aunque si se desea publicar la aplicación en el mercado, sí habrá que comprarla a un precio de 25$). Por estos motivos, se escogió Android como plataforma de desarrollo para el proyecto.
Una vez escogido el sistema operativo en donde queremos desarrollar nuestra aplicación, el siguiente punto a tener en cuenta se ciñe al entorno de desarrollo donde vamos a estructurar el código.

Existen dos IDE (Integrated Development Enviroment ó entorno de desarrollo integrado) principales: el primero es el SDK (Software Development Kit ó kit de desarrollo software) de Android para Eclipse (parecido a Netbeans) y el segundo es el Android Studio.

El SDK de Android lleva más tiempo entre los desarrolladores y hasta hace relativamente poco tiempo era la principal opción para el desarrollo en Android ya que recibía gran soporte por parte de Google y de la comunidad Android. A mediados de 2013, Google inició el proyecto Android Studio para convertirlo en la herramienta principal de desarrollo teniendo, actualmente, el apoyo necesario por parte de la empresa tecnológica. Ya que podemos hallar en Internet gran cantidad de material didáctico con respecto a Android Studio y que también se trata de un programa con ciertos años de maduración, se eligió esta plataforma de desarrollo para nuestra aplicación.

3.2 Objetivos de la aplicación

Tal y como se ha expuesto en el capítulo 1 de esta memoria, el objetivo en el que queda envuelto este TFG está comprendido en un proyecto más amplio cuyo propósito es la creación de una app para dispositivos móviles que, en términos generales, sea capaz de detectar el nivel de llenado de un recipiente. A su vez dicho proyecto se divide en dos partes, una de las cuales, a la que este TFG atañe, queda relacionada con la captura de una imagen en las mejores condiciones posibles del objeto en cuestión.

Siguiendo con lo explicado en los objetivos referidos en el capítulo 1, lo que el desarrollo de este trabajo pretende es el de recopilar una serie de información y procedimientos que sirvan como base para impulsar el crecimiento y desarrollo de la aplicación móvil orientada a un usuario final. Es por esto que el objetivo de la aplicación de este TFG es la de verificar la viabilidad de la misma y para ello, además de proporcionar información sobre el entorno de desarrollo de aplicaciones Android, de las herramientas disponibles para este contexto y del procedimiento seguido para la obtención de un modelo del objeto, lo que se pretende con la app en sí es la de dar un ejemplo básico que ratifique el futuro del desarrollo de la aplicación de cara a alcanzar los objetivos finales.

3.3 Planteamiento de la aplicación

3.3.1 Idea del desarrollo

Debido a la poca experiencia tanto del tutor como del alumno en el tema del desarrollo de aplicaciones Android, el planteamiento principal que se quería llevar a cabo desde el comienzo era la de realizar algo sencillo, que pudiera ser factible de lograr en el año académico y que no implicara mucha complejidad tanto en el diseño como en la programación de la app. Por este motivo, se pensó en el siguiente diseño de interfaz de usuario como punto a alcanzar:

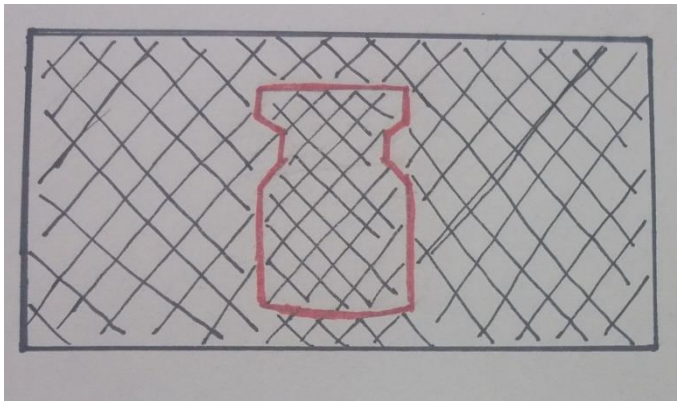

Figura 3-3-1: Idea conceptual de la interfaz de usuario de la app.

Básicamente lo que tenemos en la imagen superior es la representación de la pantalla del dispositivo móvil en posición horizontal captando imágenes de manera continua del exterior a través de la cámara del móvil y, en superposición a la pantalla de manera fija, tenemos la forma del objeto a reconocer en color rojo. Debido al contexto en que se sitúa este TFG, el objeto a detectar se decidió que fuera un bote transparente lleno de "cosas".

Figura 3-3-2: Bote transparente lleno de "cosas".

La silueta en rojo del bote en la pantalla pretende guiar al usuario hacia la detección del objeto: el usuario va recorriendo la escena con el dispositivo móvil y en el momento en que superpone la silueta en rojo de la pantalla con la silueta del objeto en la realidad, la aplicación reconoce el objeto y toma una imagen de manera automática para guardarla dentro del dispositivo móvil.

3.3.2 Estructura de la aplicación

La estructura de este trabajo se ha dividido en tres bloques principales: el primero se corresponde con la interfaz de usuario; el segundo con la captura de la foto y su almacenamiento en la memoria del dispositivo móvil y el tercero trata de la parte lógica de detección del recipiente transparente, en donde se ha implementado un algoritmo básico de detección de objetos basado en el conteo de píxeles de la imagen captada por la cámara en tiempo real.

Para la interfaz de usuario y para el tercer bloque, además, se ha desarrollado un proceso de creación del molde del recipiente siguiendo unos cuantos pasos que se describirán en el Anexo B de esta memoria. Dicho proceso es genérico y puede seguirse para la elaboración de diferentes moldes que se usen para la detección de otro tipo de recipientes.

3.3.3 Etapas del desarrollo

Para cumplir con el objetivo que la aplicación pretende resolver se acordó una estrategia de desarrollo para regular los tiempos en que se desarrollaría la app a lo largo del curso académico:

• Para el primer cuatrimestre del curso, debido a la poca experiencia del alumno en el desarrollo de aplicaciones para Android, el trabajo consistió en adquirir una base que permitiera familiarizarse con el entorno de programación Android Studio así como descubrir herramientas y funciones propias del lenguaje a la vez que se ponía en marcha alguna aplicación a modo de prueba.

• En el segundo cuatrimestre, una vez completado el período de "formación", el propósito era trabajar ya en la aplicación en sí. Para ello, fue necesario precisar más en los aspectos técnicos que iban a conformar la app.
Concretamente, la primera etapa que se llevó a cabo fue realizar una primera versión de la interfaz de usuario además de ser capaz de tomar una foto con la cámara y guardarla dentro del dispositivo. Una vez conseguido esto, el siguiente paso fue completar la aplicación mediante la incorporación del algoritmo de detección y terminar la interfaz de usuario, de tal modo que cuando se reconociera el objeto en concreto se tomara la foto y se guardara en el dispositivo. Para este segundo paso, se tuvo que trabajar con la librería de OpenCV para el tratamiento de imágenes en tiempo real además de crear el proceso de obtención del molde de la silueta del objeto usado tanto en el algoritmo de detección como en la interfaz de usuario.
Dicho trabajo realizado queda documentado y explicado en los posteriores capítulos y anexos.

4 Implementación

En este capítulo lo que se pretende es ahondar más en los aspectos técnicos del proceso de implementación de la app que en el capítulo anterior se ha presentado. Para ello, primeramente, se expondrá un diagrama de flujo de la trayectoria del trabajo que se ha seguido para el desarrollo de la aplicación con el objeto de recopilar de un modo más inmediato el procedimiento seguido:

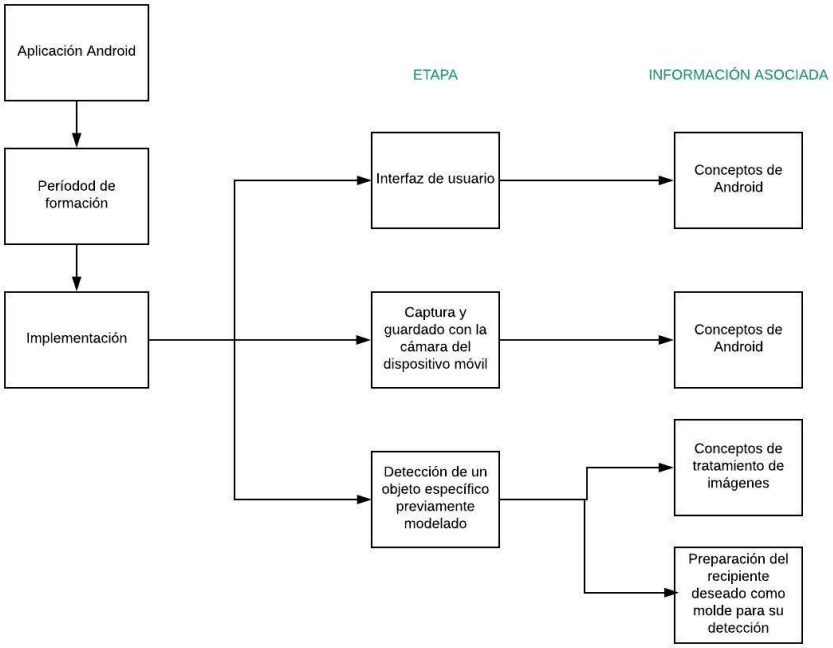

Figura 4-1: Diagrama de flujo de la implementación de la app.

Como se puede observar, la implementación ha constado de tres etapas tal y como se expuso en el capitulo anterior. Además, para cada etapa, se le ha asociado una información extra que ayuda a comprender mejor los pasos seguidos dentro de cada una. Dicha información se desarrollará en los siguientes epígrafes y anexos.

A continuación, antes de entrar en el desarrollo de la aplicación, se expondrá la documentación previa llevada a cabo antes de cada etapa organizada en dos epígrafes:

4.1 Conceptos de Android

Para comprender mejor el funcionamiento y la estructura de una aplicación Android es necesario explicar una serie de conceptos y componentes básicos presentes en cualquier aplicación basada en el lenguaje de programación Android.

4.1.1 Actividad

Señalamos la actividad como el primer concepto básico.

Una actividad representa un componente de la aplicación poseedora de una interfaz que permite al usuario interactuar con el fin de realizar una acción, ya sea hacer una foto, consultar el correo electrónico, realizar una llamada, etc.

Una actividad, visualmente, suele ocupar el total de la pantalla del dispositivo pero puede ser modular y tener unas dimensiones más reducidas ó flotar sobre otras ventanas y actividades. Normalmente cualquier aplicación está formada por numerosas actividades interconectadas entre sí y cada una, a su vez, puede iniciar otras nuevas.

La actividad inicial mostrada por una aplicación al comenzar, se califica como la actividad principal, utilizándose como punto de partida para poder lanzar otras actividades a que realicen su función programada. Cada vez que una nueva actividad empieza, la anterior es detenida aunque el sistema evita que se cierre y dejándola en un segundo plano en la pila de actividades.

Asimismo, una actividad puede dejar de desempeñar su función, apartándola definitivamente del sistema. Todas estas acciones conforman lo que denominamos como el ciclo de vida de cualquier actividad, estando cada estado interconectado entre sí a través de métodos propios de la misma clase de actividad, tal y como se ilustra en la siguiente imagen:

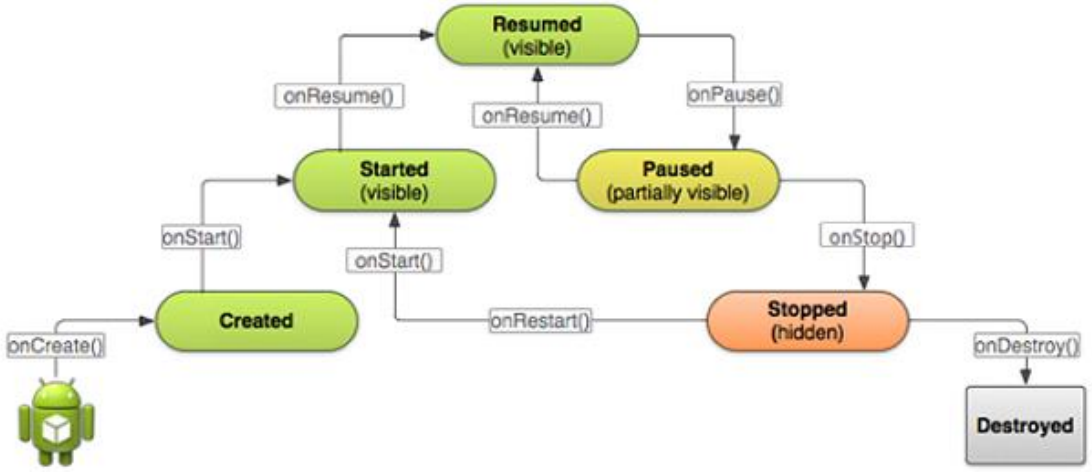

Figura 4-2: Ciclo de vida de una actividad Android.

- **Activa** o '**Resumed**':

La actividad se encuentra encima de la pila de actividades, por loque es visible y el usuario puede interactuar con ella.

- **Pausada** o '**Paused**':

La actividad es visible pero no tiene el foco, es decir, existe otra actividad que no ocupa el total de la pantalla sobre ella.

- **Parada** o '**Stopped**':

La actividad no es visible, el programador elige qué preferencias guardar cuando la aplicación entra en este estado y poder recuperarla en un momento posterior.

- **Destruida** o '**Destroyed**':

La actividad es "matada" por el sistema.
Adicionalmente, toda actividad está dividida en dos partes generadas en dos archivos: uno de tipo Java y otro de tipo XML. El primero representa la parte lógica donde se puede manipular, interactuar y colocar el código que desempeña la funcionalidad de la actividad. El segundo es empleado para el diseño de la interfaz cuyos elementos serán configurados según el código de la parte lógica.

4.1.2 Intenciones

Otro concepto básico para comprender el funcionamiento de una aplicación Android corresponde a las intenciones.
Una intención no es más que una solicitud para realizar una acción. En esencia, es un mensaje transferido entre los componentes de una aplicación. Son utilizados, por ejemplo, para cambiar de una actividad a otra.

4.1.3 Vistas

Otro concepto es el de las vistas. Una vista se define como un tipo de objeto empleado para dibujar contenido (interactivo o no) en la pantalla del dispositivo. Mientras que se puede instanciar una vista desde el código Java, la manera más sencilla es a través de un archivo de diseño XML. Se trata de una clase muy útil ya que es la base para los widgets (botones, cajas de texto…), permitiendo, además, introducir tus propios gráficos en la aplicación.

4.1.4 Manifiesto de Android

Todas las aplicaciones deben tener un archivo AndroidManifest.xml (exactamente con este mismo nombre) en su directorio raíz.

Este archivo proporciona información fundamental acerca de la aplicación al sistema Android, la cual es fundamental para poder ejecutar el código de la aplicación.

Entre la información que contiene, encontramos lo siguiente:

- Describe los componentes de la aplicación, como las actividades, los servicios, los receptores de mensajes y los proveedores de contenido que la integran.

- Determina los procesos que alojan a los componentes de la aplicación.

• Determina los permisos que debe tener la aplicación. Por ejemplo, permisos para que el usuario pueda usar la cámara del dispositivo y tomar y guardar fotos.

• Declara el nivel mínimo de Android API que requiere la aplicación.

• Enumera las bibliotecas con las que debe estar vinculada la aplicación.

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk android:minSdkVersion="15" />

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

    <application
        android:label="@string/app_name"
        android:icon="@drawable/ic_launcher">
        <activity
            android:name="MyActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Figura 4-3: Ejemplo AndroidManifest.xml.

4.2 Conceptos de tratamiento de imágenes

Tal y como se ha expuesto en el epígrafe 2.5. “Tratamiento digital de imágenes”, esta disciplina de técnicas por computación de señales de información visual busca como objetivo destacar características concretas de una imagen original con el fin de crear otro tipo de imagen que resulte más apropiada para el resultado que se pretende obtener.

Dado el propósito de éste TFG, resulta inevitable valerse de algunas técnicas y algoritmos desarrollados por dicha disciplina. Para comprender de un modo más exacto las operaciones realizadas en las imágenes que se explicarán en epígrafes posteriores, es necesario dar a conocer algunas de dichas técnicas del procesado de imágenes usadas en el desarrollo de la aplicación.

4.2.1 Imagen binarizada

Una imagen binarizada es aquella que posee un único canal, es decir, sin componentes de color. Además, cada píxel queda representado por un valor lógico, esto es, por un 1 ó un 0, existiendo únicamente en la imagen el blanco ó el negro. Este tipo de imágenes es muy útil a la hora de realizar operaciones entre ellas ya que, al tener únicamente dos posibles

valores por píxel (0 ó 1), reduce considerablemente el coste computacional de las operaciones.

Figura 4-4: Imagen original.

Figura 4-5: Imagen binarizada.

4.2.2 Detección de bordes Canny

Desarrollado en 1986 por John F.Canny, es un algoritmo de múltiples etapas para detectar bordes en imágenes. Facilita el reconocimiento de objetos o la segmentación de regiones entre otras cosas. Está considerado como uno de los mejores métodos de detección de contornos mediante el empleo de máscaras de convolución y basado en la primera derivada.

Es necesario convertir la imagen a escala de grises para aplicar este método y el resultado obtenido es una imagen binarizada con los bordes detectados.

Figura 4-6: Imagen original en gris.

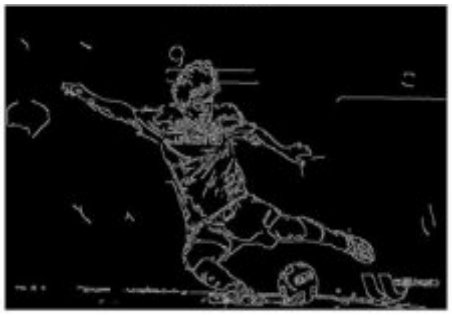

Figura 4-7: Bordes por Canny.

4.2.3 Dilatación en imágenes

La dilatación consiste en una técnica morfológica que se describe como un crecimiento de píxeles. Este operador permite incrementar las dimensiones de los bordes de cada elemento situado en la imagen, lo cual ayuda a rellenar hoyos dentro de una región o simplemente engrosar contornos. Dicha operación, debe realizarse sobre una imagen binarizada.

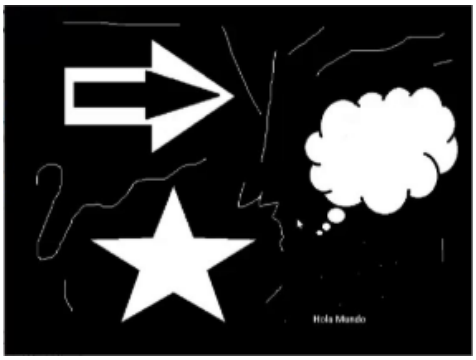

Figura 4-8: Imagen original.

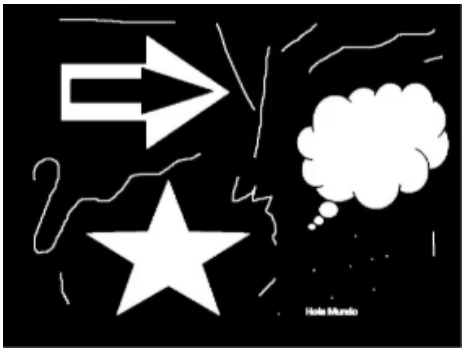

Figura 4-9: Imagen dilatada.

4.2.4 Escalado de una imagen

El escalado de una imagen se configura como una transformación de tipo afín consistente en variar las dimensiones de una imagen para aumentarla o reducirla. Si la variación a lo largo del ancho y del largo es proporcional, se conservan las proporciones de los elementos que componen la imagen, o por el contrario se distorsionan.

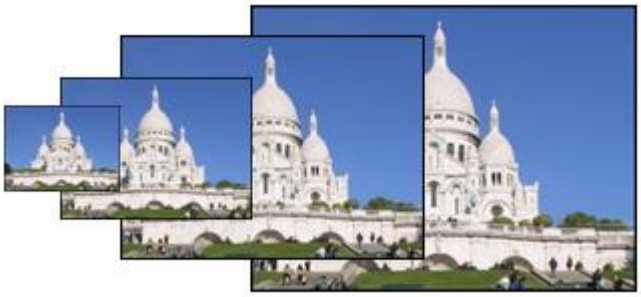

Figura 4-10: Escalado de una imagen a distintas escalas.

4.3 Desarrollo de la aplicación

4.3.1Interfaz de usuario

La interfaz de usuario aunque haya sido mostrada como una etapa que pudiera ser independiente de las otras dos, lo cierto es que en el proceso de desarrollo empezó programándose en la etapa de captura y guardado de una imagen y terminó completándose en la última etapa de detección del objeto. Para poder explicarla, se empezará recordando la idea que se quería llevar a cabo expuesta en el capítulo anterior:

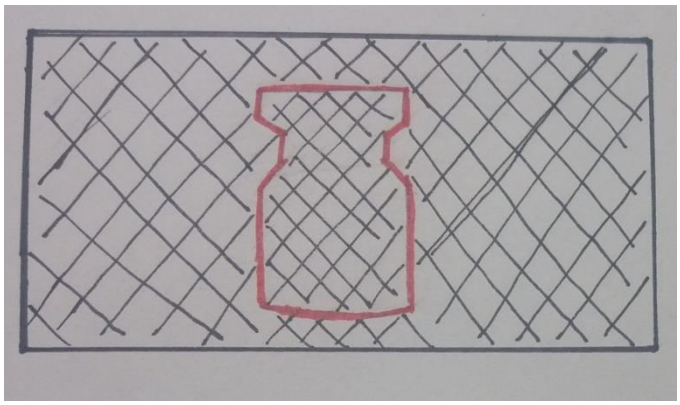

Figura 4-3-1-1: Idea conceptual de la interfaz de usuario de la app.

La idea era sencilla: mostrar en la pantalla del dispositivo las imágenes del exterior que captaba la cámara mientras el usuario recorría la escena guiado por la silueta del recipiente superpuesta a la pantalla, hasta hacer coincidir la silueta de la pantalla con la de la escena en la realidad y reconocer así el objeto.

Figura 4-3-1-2: Objeto concreto a reconocer: bote transparente lleno de "cosas".

Para llevar a cabo esta idea, la primera tarea fue ser capaz de mostrar en la pantalla lo mismo que veía la cámara del móvil. Para esto, el propio lenguaje Android posee varias soluciones pero lo que se quería era algo más concreto: que las imágenes procedentes de la cámara cubrieran toda la pantalla y además poder "tener a mano" dichas imágenes para poder ser analizadas posteriormente.

La librería OpenCV proporciona la mejor solución ante estos requisitos. Para poder operar con ella es necesario instalarla tanto en el entorno de desarrollo Android Studio, como en el propio dispositivo móvil. Dicha instalación se detalla en el *Anexo A* de este TFG. Una vez preparada la librería de OpenCV en ambas plataformas, se está preparado para usar sus clases y métodos.

Como ya se mencionó en el epígrafe 4.1.Conceptos de Android, una actividad está dividida en dos partes: la parte XML, referente al diseño de la interfaz de dicha actividad y la parte del archivo de tipo Java que se ocupa de manejar la lógica que se desea implementar.

Haciendo uso de las librerías de OpenCV, utilizamos una de sus clases llamada *JavaCameraView* a modo de puente entre ésta y la cámara de Java. Dicha clase es usada para establecer una vista (llamada también layout) que permite mostrar al usuario lo que capta la cámara del dispositivo activada y a la vez poder manejar cada frame que la cámara recoge de manera interna. Dicho layout lo introducimos en el archivo XML.

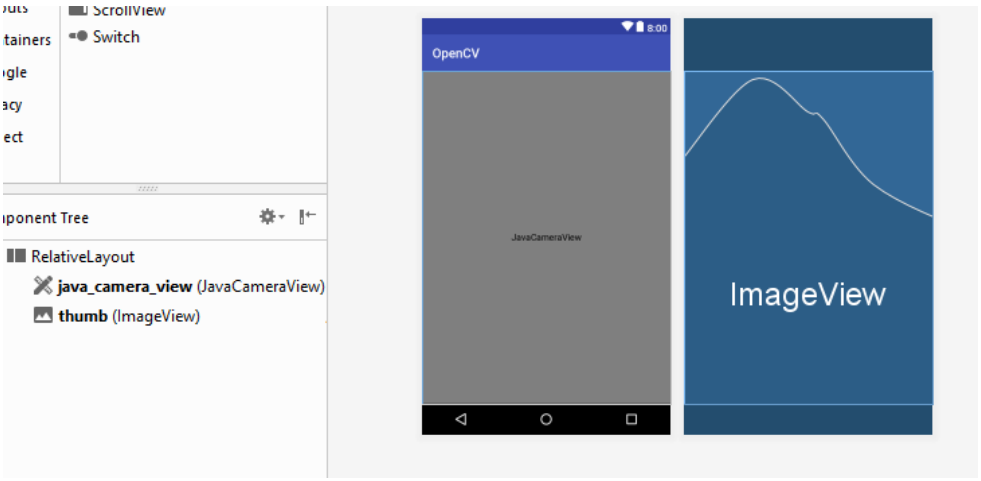

Figura 4-3-1-3: JavaCameraView en el archivo XML.

A la vez que en la pantalla del dispositivo vemos la imagen en tiempo real proporcionada por la clase *JavaCameraView*, también tenemos superpuesta otra vista del tipo *ImageView*, que contiene la silueta del objeto a detectar y que sirve a modo de guía para el usuario tal y como ya se ha comentado anteriormente. El *Anexo B* que esta memoria detalla proporciona un paso a paso para conseguir la silueta del objeto específico con las mismas dimensiones que usa nuestra aplicación en la pantalla del dispositivo. El resultado de esta operación es el siguiente:

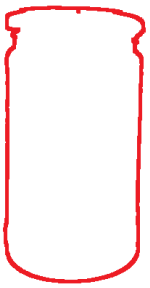

Figura 4-3-1-4: Silueta del recipiente usada como guía.

Como ya se ha dicho, se introduce dentro del *ImageView* la silueta del recipiente (el código se entrega al tutor de este TFG) y el resultado visual final es el que sigue:

Figura 4-3-1-5: Interfaz de usuario, resultado final.

Como se puede ver, en la pantalla del dispositivo se tiene la imagen captada en tiempo real por la cámara con la silueta del objeto específico a modo de guía superpuesta a ésta. Además, cada frame captado puede ser analizado internamente en el archivo de tipo Java gracias al método *OnCameraFrame*. Método el cuál se usará para introducir en él el algoritmo de detección que se explicará más adelante.

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

Figura 4-3-1-6: Método *OnCameraFrame*.

4.3.2 Captura de imagen con la cámara del dispositivo móvil

La función de este apartado consiste en lo siguiente: una vez detectado el objeto deseado, se toma una foto de manera automática y se guarda dentro del dispositivo. Para ello, primeramente, hemos de proporcionar una serie de permisos a la aplicación (dentro del manifiesto de Android) para que pueda ser usada la funcionalidad de la cámara.

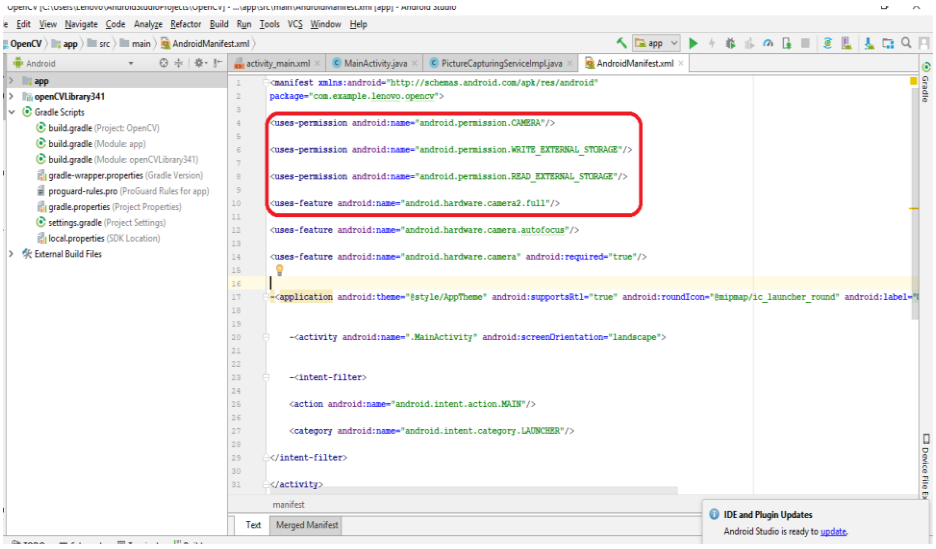

Figura 4-3-2-1: Permisos específicos para usar la funcionalidad de la cámara.

Dichos permisos consisten en poder utilizar la propia cámara, ser capaz de "escribir" en la memoria del teléfono (guardar las fotos) y poder leer de ella. Adicionalmente, se ha añadido una instrucción extra del tipo <*uses-feature*> para que en un futuro, cuando se quiera descargar la app a través de Google Play, ésta cumpla con los requisitos de las funciones de software y hardware.

La funcionalidad de esta parte de la aplicación ha sido programada dentro de una llamada "PictureCapturingServiceImpl.java". En términos generales, lo que se realiza en ella, es manejar una sesión de captura de imagen. Dicha sesión se encarga de manipular a través de las clases y métodos aportados por las propias librerías de la cámara, los distintos estados que conforman la toma de una imagen y su almacenamiento en el dispositivo.

```
import android.hardware.camera2.CameraAccessException;
import android.hardware.camera2.CameraCaptureSession;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraDevice;
import android.hardware.camera2.CameraManager;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;
import android.hardware.camera2.TotalCaptureResult;
import android.hardware.camera2.params.StreamConfigurationMap;
```

Figura 4-3-2-2: Librerías para el uso de la cámara del dispositivo.

La acción más relevante dentro del código de esta clase radica en la creación de un objeto derivado de la clase abstracta "StateCallback". Dicho objeto se encarga de realizar la labor antes mencionada, manejando los métodos que componen los estados de una sesión de captura con la cámara en función del punto en que se encuentre dicha sesión de captura.

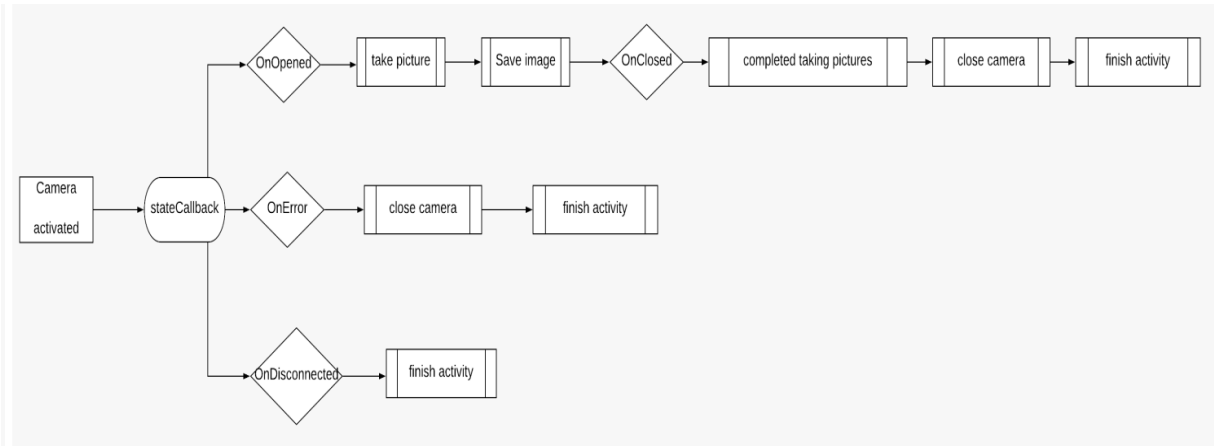

Figura 4-3-2-3: Diagrama de flujo de la sesión de captura de imagen.

La cámara es activada. El objeto de la clase StateCallback recibe actualizaciones del estado de la cámara. Los estados que se manejan son:

• **OnOpened:**

La cámara se ha abierto correctamente y está en funcionamiento para capturar una imagen. En el proceso de captura se prepara un recipiente para almacenar los bytes que componen la imagen obtenida y se guarda dentro de la memoria con un nombre especificado.

• **OnError:**

La cámara no se ha abierto de manera correcta y se procede a cerrarla y a cesar la actividad.

• **OnDisconnected:**

La cámara se ha desconectado y por lo tanto, se finaliza la actividad.

• **OnClosed:**

Una vez consumada la acción de la toma y el posterior guardado de la foto, se procede a la desactivación de la cámara y al cese de la actividad.
Por último, toda esta actividad, en conjunto, que compone la captura y el guardado de la foto es lanzada desde el main de la aplicación a través de un mensaje de solicitud de acción, es decir, a través de una intención.

```
startActivity(new Intent(MainActivity.this, PictureCapturingServiceImpl.class));
```

Figura 4-3-2-4: Actividad lanzada desde el main a través de un mensaje "intent".

4.3.3 Detección de un objeto específico previamente modelado

Tal y como ya se ha explicado con anterioridad, la idea para esta parte de la aplicación es usar directamente la silueta del recipiente sobrepuesta a la vista de la cámara para el reconocimiento del objeto específico.

Figura 4-3-3-1: Vista de la silueta del recipiente (rojo) superpuesta a la imagen que capta la cámara del objeto.

De este modo, el usuario tendrá en todo momento una referencia con la información en cuanto a tamaño y ángulo del objeto que se quiere reconocer. Para detectar el recipiente, simplemente el usuario tiene que superponer la silueta (en color rojo) encima del objeto, haciendo que ambos contornos encajen, en la mayor medida de lo posible, para que el algoritmo de detección, que se explicará en este epígrafe, actúe con mayor eficacia.
Para ser capaz de explicar el código del algoritmo de detección, en primer lugar, se va a exponer la teoría que apoya su implementación.

Primero, se quiere disponer de una imagen binaria de la silueta del objeto para ser usada a modo de máscara. Para obtenerla, se ha seguido el proceso descrito en el *Apéndice B* que detalla esta memoria. Proceso que, por otro lado, puede tomarse como referencia para obtener distintas siluetas de otros recipientes. El modo en que se crea la máscara determinará la manera en cuanto al ángulo y a la distancia que el usuario deberá tomar para situarse ante el objeto con el fin de que éste pueda ser reconocido.

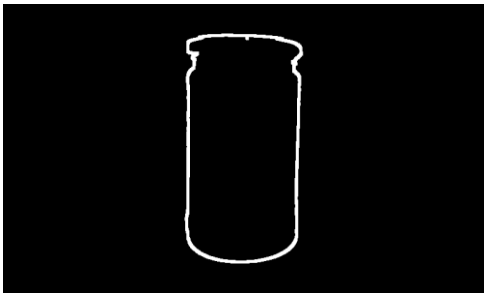

Figura 4-3-3-2: Imagen binaria de la silueta del recipiente.

El objetivo es filtrar cada frame obtenido con la cámara en tiempo real a través de la silueta del objeto (máscara) con la operación lógica "and" y contar el número de píxeles blancos de la imagen resultante. Para realizar esta operación, previamente, hay que efectuar un procesado de la imagen en tiempo real con el detector de bordes de Canny. De esta manera, se consigue una imagen con los contornos de los objetos que permanecen dentro de la escena y además queda también binarizada.

El propósito de la operación lógica "and" reside en superponer exactamente las dos siluetas del objeto (la de la máscara y la del tiempo real), siendo el resultado óptimo una imagen con la misma silueta del objeto como único elemento.

Nos encontramos con dos posibles escenarios:

a) que el recipiente esté situado dentro de la escena
b) que no lo esté.

• En el caso de que estuviera presente el recipiente en la escena:

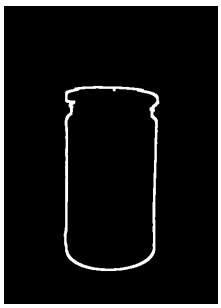

Figura 4-3-3-3: Silueta del recipiente (máscara).

Figura 4-3-3-4: Silueta del recipiente en tiempo real.

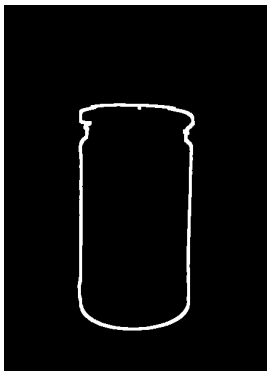

Figura 4-3-3-5: Resultado ideal después del filtrado entre ambas siluetas.

• En el caso de que se estuviera enfocando otra cosa que no fuera el recipiente:

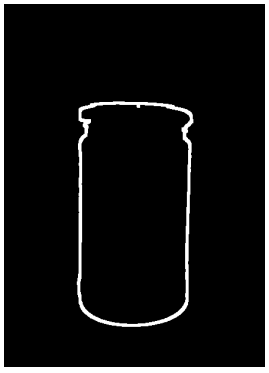

Figura 4-3-3-6: Silueta del recipiente.

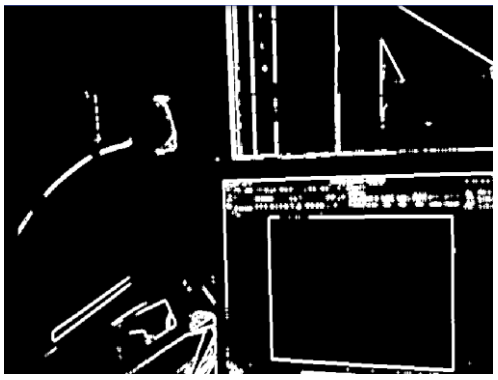

Figura 4-3-3-7: Siluetas de otros objetos distintos al recipiente en tiempo real.

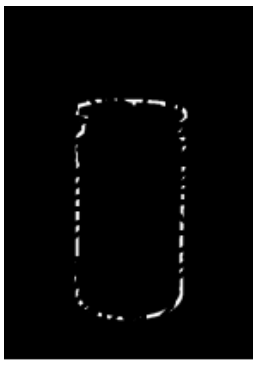

Figura 4-3-3-8: Resultado después de la operación "and" entre ambas imágenes.

Teóricamente, una vez enfocado el objeto en cuestión, el conteo de los píxeles blancos de la imagen resultante del filtrado tiene que ser igual al número de píxeles blancos de la imagen que contiene el contorno del objeto (máscara).

En la práctica, se confía en que las dos cifras resultantes sean parecidas o al menos que la cantidad de píxeles blancos en la imagen filtrada alcance un valor muy superior cuando el usuario se centra en el objeto, extremo que no se daría cuando se enfoca otra cosa que no sea el recipiente que se desea detectar. Para la toma de la foto, simplemente se fija un

umbral y cuando la cantidad de píxeles blancos en la imagen filtrada supera dicho umbral, se lanza la actividad de captura de foto explicada en el anterior apartado.

Las imágenes mostradas anteriormente son recursos ilustrativos que defienden la estrategia seguida para la implementación del algoritmo básico de detección. Por otra parte, las conclusiones a las que se han llegado en el ejercicio práctico arrojarán otro tipo de resultados los cuales habrá que tomarse en consideración para elaborar una serie de factores y apuntes que ayuden a la optimización de todo el proceso de reconocimiento de este tipo de recipientes.
Se ahondará más en dichos resultados prácticos y conclusiones en el siguiente capítulo de este TFG.

Una vez explicado el algoritmo de detección empleado, se desarrollará el proceso llevado a cabo para su implementación dentro de la app.

Para finalizar este capítulo, la última cuestión que se aborda, nos introduce directamente en la parte interna del código de la aplicación.

En cuanto al archivo de tipo Java, es en esta parte donde se implementa la funcionalidad del algoritmo previamente descrito. Para poder manejar cada frame que arroja la cámara del dispositivo, tenemos que programar dentro del método igualmente proporcionado por OpenCV: *onCameraFrame(inputFrame)*. Dicho método recibe como argumento el frame captado a través del *JavaCameraView*, lo cual permite realizar operaciones de manera directa con las imágenes captadas en tiempo real. Tales operaciones son las ya comentadas a lo largo de este apartado.

```
mRgba = inputFrame.rgba();
Imgproc.cvtColor(mRgba, imgGrey, Imgproc.COLOR_RGB2GRAY);
Imgproc.Canny(imgGrey, imgCanny, threshold1: 200, threshold2: 200);
Imgproc.dilate(imgCanny, imgCanny, Imgproc.getStructuringElement(Imgproc.MORPH_DILATE,
        new org.opencv.core.Size( width: 5, height: 5)));
```

Figura 4-3-3-9: Procesamiento de la imagen captada en tiempo real.

El primer bloque de código, arriba mostrado, corresponde al procesamiento de la imagen con el fin de que la app, a nivel interno, "observe" las siluetas de los objetos que quedan dentro de la pantalla del dispositivo móvil. Haciendo uso de los métodos proporcionados por las librerías de OpenCV, en primer lugar, convertimos la imagen que nos llega de 3 canales (imagen a color) a una imagen con un único canal (imagen en escala de grises).
A continuación, aplicamos el algoritmo de Canny para detección de contornos y por último, se realiza una pequeña dilatación a los bordes localizados con el fin de que resulten algo más visibles para el usuario.

```
Core.bitwise_and(imgCanny, mask480x800, imgResultado);
contadorNonZeroPixels = Core.countNonZero(imgResultado);
if (contadorNonZeroPixels > 2200){
    startActivity(new Intent( packageContext: MainActivity.this, PictureCapturingServiceImpl.class));
```

Figura 4-3-3-10: Algoritmo básico de reconocimiento de objetos.

El segundo y último bloque de código llevado a cabo corresponde con la implementación del algoritmo de detección. De nuevo, gracias a las librerías proporcionadas por OpenCV, esta tarea se convierte en algo relativamente sencillo: para cada frame captado en tiempo real, se filtra junto a la imagen "máscara" con la operación lógica "and". A continuación contabilizamos sus píxeles blancos resultantes. Si dicho número es mayor que cierto umbral, lanzamos la actividad de captura de foto y obtenemos de esta manera la imagen del objeto elegida. Como puede intuirse, la elección de dicho umbral será determinante para el funcionamiento del algoritmo, por lo que será necesario una serie de pruebas para entender y determinar el valor de dicho umbral.

5 Evaluación: pruebas, resultados, limitaciones y valoración

En el capítulo anterior se describió el proceso de funcionamiento de la aplicación bajo un modelo simple de detección de objetos ideal. Desde un punto de vista teórico se puede intuir cómo será su comportamiento bajo unas circunstancias ideales que permiten mostrar con total claridad la silueta del recipiente transparente que se quiere reconocer, sin embargo, las circunstancias reales en donde se use la app harán que se tenga en cuenta cierto tipo de consideraciones a la hora de entender y, por lo tanto, mejorar el algoritmo de detección, que al fin y al cabo es la pieza central de la aplicación.

Un primer estudio que puede ayudar a comprender el funcionamiento del algoritmo pasa por establecer un valor óptimo para el umbral de detección que represente al mayor número de escenarios posibles en el que pueda ubicarse el objeto deseado, que en este caso, se trata de un recipiente transparente.

Sin lugar a duda, un recipiente transparente no presenta las mismas condiciones que un objeto opaco. En un objeto opaco, la luz no atraviesa su estructura, sino que es reflejada totalmente. Por ello, su detección de contornos por Canny resulta ser óptima. Por otra parte, en un objeto transparente, la luz sí atraviesa su estructura lo que provoca que no toda la luz incidente sea reflejada hacia el exterior del objeto. Este aspecto conduce a zonas donde la detección de contornos por parte del algoritmo de Canny es incapaz de visualizar el recipiente y, por lo tanto, el algoritmo de detección de objetos pensado para la aplicación no sea tan eficiente como cabría de esperar.

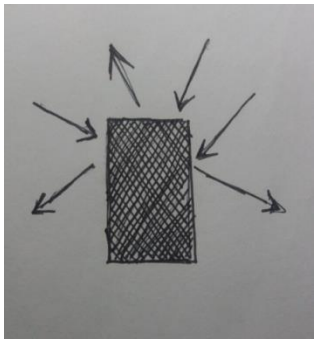

Figura 5-1: Interacción de la luz en un objeto opaco.

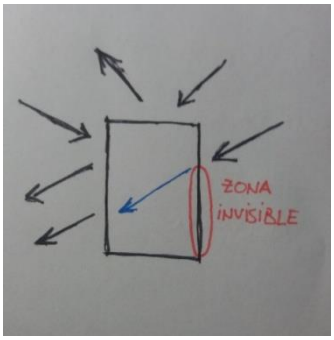

Figura 5-2: Interacción de la luz en un objeto transparente.

Por lo anteriormente aludido, en este apartado se evaluarán, a través de las pruebas realizadas, las condiciones externas a la aplicación para establecer el mejor umbral en cada caso en la detección del recipiente transparente elegido y de este modo hacer un primer análisis de cómo las condiciones externas como lo son iluminación, ángulo de posicionamiento frente al objeto… condicionan su detección.

5.1 Pruebas

Para la realización de las pruebas, se pensó en elaborar dos "sesiones de captura" que pudieran representar los dos escenarios más comunes en donde un usuario pudiera usar la app: en el exterior, con luz natural y abundantes elementos de fondo; y en el interior, con luz artificial y pocos ó ningún elemento de fondo.

Cada sesión de captura consiste en usar la aplicación por parte del programador (sin haber establecido aún ningún umbral de decisión y por lo tanto con el algoritmo de detección de objetos desactivado) durante unos pocos segundos (el tiempo exacto no se midió en el momento) en donde habría instantes que se hiciese coincidir la imagen guía con la silueta del recipiente y otros momentos en donde la cámara estuviera enfocando otras cosas excepto el recipiente.

Figura 5-3: Entorno exterior.

Figura 5-4: Entorno interior.

El objetivo de estas sesiones de captura es recopilar la información arrojada a través del monitoreo del número de píxeles blancos que la imagen resultante de la operación lógica "and" entre la máscara y la imagen en tiempo real.

Figura 5-5: Detección ideal del recipiente en imagen en tiempo real

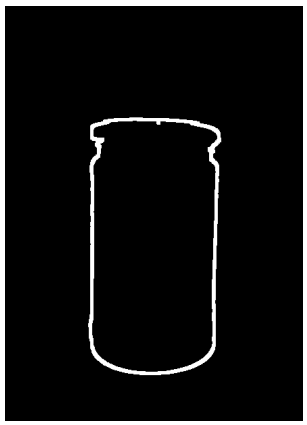

Figura 5-6: Máscara

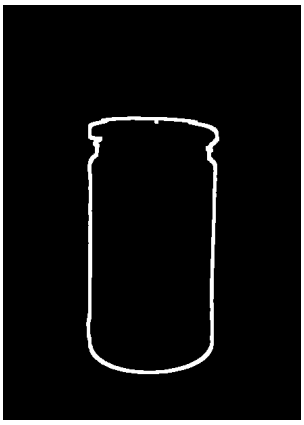

Figura 5-7: Resultado ideal del filtrado, que coincide con la imagen de la máscara

Idealmente el resultado esperado queda ilustrado en las anteriores imágenes, pero en la realidad, dependiendo de los hándicaps como pueden ser la iluminación o el ángulo de posicionamiento del dispositivo frente al objeto, que dificultan en cierta medida la correcta detección del objeto en concreto, los resultados visuales después del filtrado entre la máscara y la imagen en tiempo real son las imágenes expuestas a continuación.

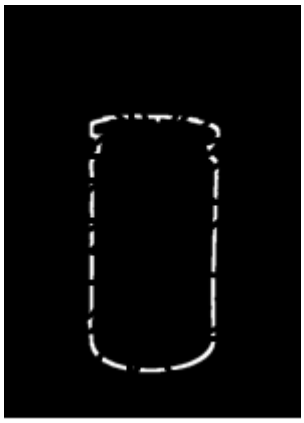

Figura 5-8: Posible resultado 1.

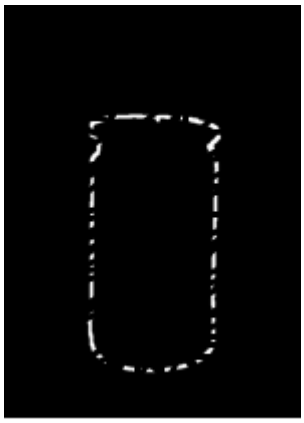

Figura 5-9: Posible resultado 2.

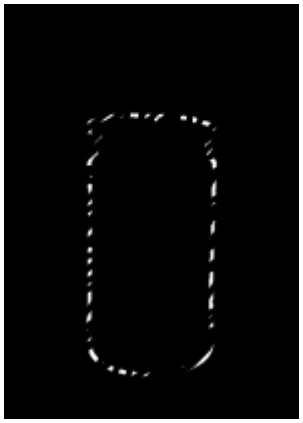

Figura 5-10: Posible resultado 3.

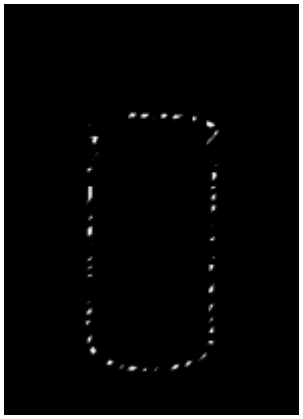

Figura 5-11: Posible resultado 4.

Traduciendo las anteriores operaciones a un ámbito ponderable, se estima que si la máscara tiene 6292 píxeles blancos (dato real), la imagen resultante puede tener números como

4000, 3000 ó 2000 píxeles blancos, en el momento en que el usuario superponga las dos siluetas del recipiente (la silueta guía y la silueta del recipiente en tiempo real). De este modo, conociendo de antemano los datos que un usuario puede encontrar al utilizar la app, puede adaptarse el umbral de decisión de una manera precisa. Determinar con exactitud esas últimas cifras citadas, nos dará información valiosa para analizar y comprender la influencia de las condiciones que afectan al funcionamiento del algoritmo de detección de objetos desarrollado para la app.

5.2 Resultados

Aunque la duración de cada sesión de captura haya sido del orden de unos pocos segundos, el número de píxeles blancos quedará determinado en cada frame, de tal modo que la relación establecida será el número de píxeles blancos por frame.

A continuación, se exponen los datos obtenidos del resultado tras la operación del filtrado en cada sesión de captura:

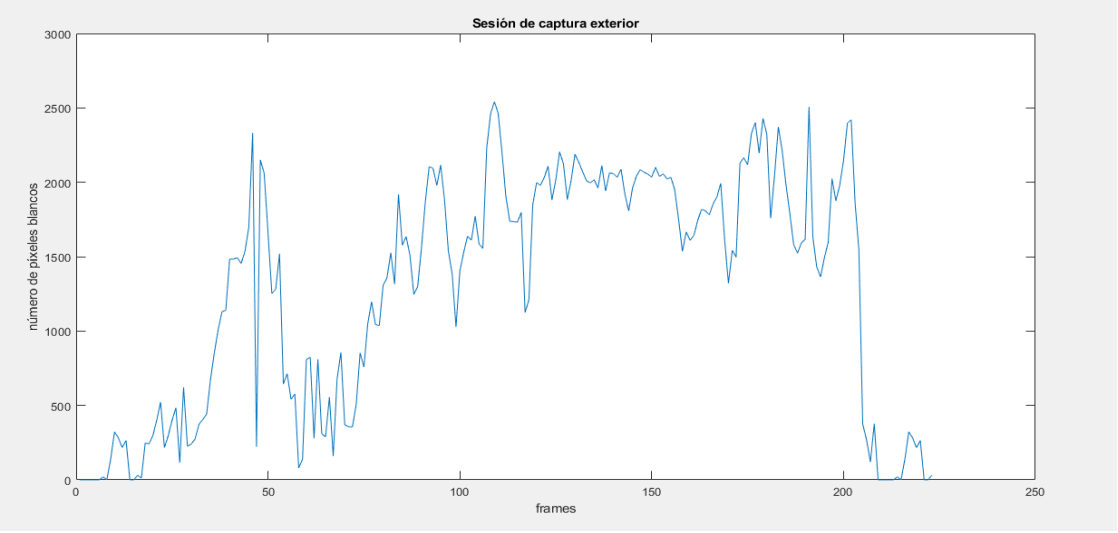

Figura 5-12: Gráfica de los datos de la sesión de prueba realizada en ambiente exterior.

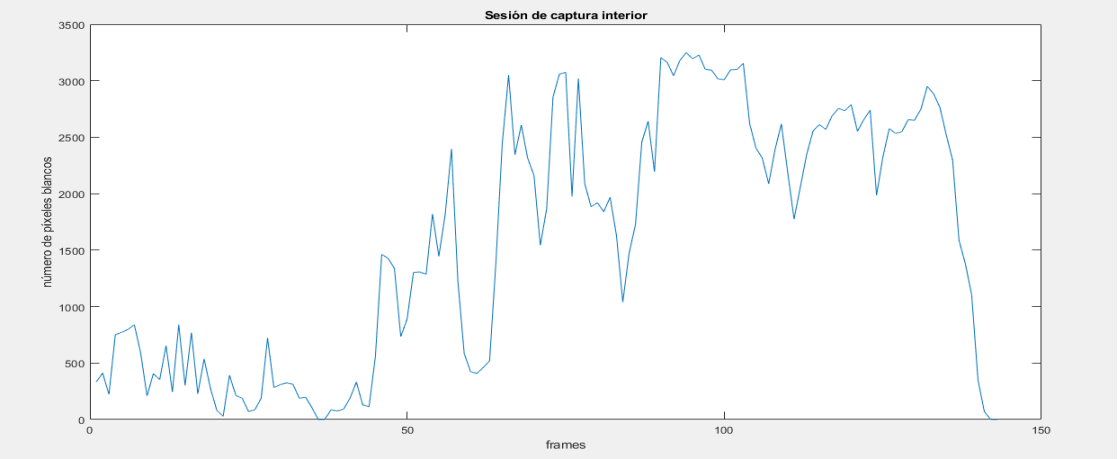

Figura 5-13: Gráfica de los datos de la sesión de prueba realizada en ambiente interior.

Como se puede observar en ambas gráficas, se aprecia la existencia de un comportamiento que entra dentro de los resultados esperados. Se puede ver que en las dos gráficas aparecen máximos relativos que corresponden con los momentos en que la coincidencia de siluetas del recipiente ha sido máxima (la de la imagen guía con la del tiempo real). Sin embargo, hay que destacar que el número de píxeles blancos resultantes quedan lejos de los 6292 que corresponden con el modelo ideal.

Por otro lado, se aprecia que, de manera individual, cada sesión de captura presenta un número de píxeles blancos distintos: mientras que en la sesión de captura exterior se alcanza máximos de entorno a 2300-2500, en la sesión de captura interior nos encontramos con máximos de entorno a 3000.

5.2 Limitaciones

Analizando de manera genérica los resultados obtenidos, se puede intuir que el valor de dichas cifras es debido al diezmado que las condiciones en contextos reales producen sobre la correcta identificación de la silueta del recipiente transparente.

Justificando esta premisa, las siguientes imágenes muestran de qué manera visual el algoritmo de detección de Canny reconoce los contornos del recipiente transparente en las dos escenas:

Figura 5-14: Ejemplo escena interior 1.

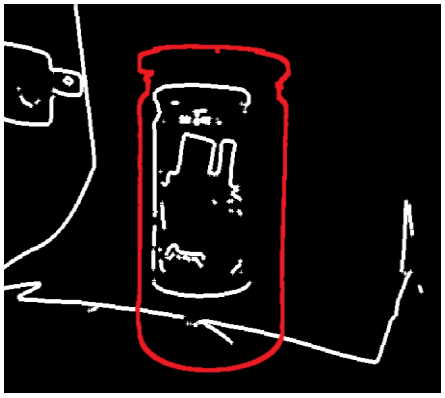

Figura 5-15: Ejemplo escena interior 2.

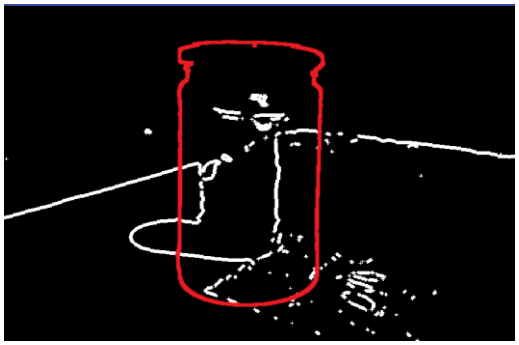

Figura 5-16: Ejemplo escena exterior 1.

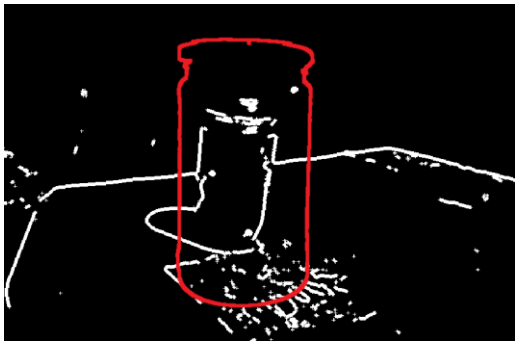

Figura 5-17: Ejemplo escena exterior 2.

En base a las imágenes anteriores, se puede apreciar que la imagen en un entorno cerrado presenta más información de la silueta del recipiente transparente en comparación a la que se obtendría en un entorno abierto.

Antes de plantear conclusiones que proporcionen una respuesta inmediata a la cuestión que se planteaba al principio de este capítulo respecto a la determinación de un umbral de decisión genérico, es importante identificar de qué manera los elementos externos pueden entorpecer la labor del algoritmo de detección de objetos transparentes ideado para este trabajo.

• En primer lugar, el elemento considerado más importante lo constituye la iluminación. En efecto, sin una correcta intensidad de iluminación sería imposible no solo no poder detectar un recipiente transparente, sino cualquier otro objeto por parte del algoritmo Canny.

• En segundo lugar, la posición del foco que da luz sobre el objeto, adquiere especial importancia en la detección de éste. Por ello, hay que señalar que si se ilumina desde una posición elevada, los haces de luz, en mayor medida, iluminarán únicamente la parte superior del objeto y no toda su estructura. El hecho de que se ilumine el objeto con la mayor homogeneidad e intensidad posible resulta ser de suma importancia para evitar que se formen "zonas invisibles" en la estructura transparente del recipiente tal y como se ilustra en la Figura 5.2 al principio de este epígrafe.

• En tercer lugar, otro elemento importante para reconocer de la mejor forma posible la silueta del recipiente transparente es el ángulo en que el usuario se posiciona frente al objeto. Las reflexiones y refracciones de las ondas que inciden en el recipiente, tal y como se ha comentado, provocan zonas en la estructura del mismo incapaces de ser detectadas por el reconocedor de contornos de Canny. Esto conlleva que el usuario tenga que estar probando distintas posiciones para encontrar el mejor ángulo de visión que minimice el efecto producido por las reflexiones y refracciones de los haces de luz al entrar en contacto con el recipiente transparente.

• Por último, otro factor externo también condicionante para el correcto funcionamiento de la app lo componen los elementos de fondo. En el caso de detectarse un número abundante de siluetas en el fondo de la escena, puede darse la posibilidad de que, sin que esté el objeto en cuestión presente, se active la toma de la foto provocando un falso positivo.

5.3 Valoración

Volviendo a la cuestión planteada en el principio de este capítulo respecto al establecimiento de un umbral general que represente a todos o al mayor número de escenarios posibles, la respuesta a dicha pregunta es que, de momento, no existe un único valor de decisión capaz de realizar esta tarea. Cada escenario es único en cuanto a cuestiones de entorno: iluminación, elementos que acompañan al objeto en la escena ó ángulo de posicionamiento frente al recipiente son elementos que dificultan el correcto funcionamiento del algoritmo de detección, y ello por las características específicas que conlleva un objeto transparente. Sin embargo, sí que existe un umbral de decisión en cada escenario capaz de comprobar la veracidad de las pretensiones que se quieren llevar a cabo en este TFG. En efecto, fijándose en los datos obtenidos en el epígrafe anterior, se puede

establecer un umbral de decisión basado en el los picos máximos vistos en las gráficas de las figuras 5.12 y 5.13 mostradas. Así por ejemplo, para un ambiente interno, si los datos arrojan niveles máximos de píxeles blancos de entorno a 2300-2500, se puede establecer un umbral de decisión de 2400 para el cual se detecte el objeto.

6 Conclusiones y trabajo futuro

Recordando el objetivo principal que se quería alcanzar con la realización de este TFG, desde el inicio del mismo lo que se ha pretendido ha sido la creación de un trabajo dedicado a la documentación de una serie de etapas y procedimientos que permitieran poner en marcha una aplicación Android orientada básicamente al reconocimiento de objetos de estructura conocida. Debido al contexto en que se sitúa este TFG, el objeto en cuestión se ha tratado de un recipiente transparente lleno de "cosas", y el esfuerzo dedicado por el que atraviesa propone el desarrollo de una aplicación para dispositivos móviles Android, no a nivel de usuario, sino únicamente a un nivel funcional, en donde los pasos y la documentación detallada en estas páginas pudieran servir como base para la verificación y la posterior mejora de la app.

Al comienzo de esta memoria se han precisado las motivaciones y los objetivos del TFG de una manera más extensa, se ha explicado el contexto tecnológico en donde se sitúa y a continuación se ha procedido con la explicación de la app. Se ha detallado la función que pretende desempeñar y además se han proporcionado los pasos seguidos en su desarrollo con los anexos como material adicional. Por último, se ha dedicado un capítulo al tema de su evaluación: se han explicado las pruebas realizadas además de esclarecer los resultados obtenidos, concediendo una serie de informaciones que resultan de suma utilidad para la comprensión del trabajo realizado.

Como conclusión a este TFG, se pude decir que sí se ha cumplido con los propósitos requeridos. Con la elaboración de este trabajo se ha logrado asentar una serie de bases en cuanto a desarrollo y procedimientos que permiten mirar con optimismo hacia la mejora del proyecto donde queda envuelto el TFG. No obstante, aún queda mucho trabajo de investigación y de pruebas que realizar con el fin de mejorar las prestaciones y la eficiencia de la aplicación. Tomando este trabajo como soporte, existe mucha capacidad de mejora en las distintas etapas que componen su desarrollo, motivo por el cual, el siguiente epígrafe tratará de exponer hacia donde deben dirigirse los esfuerzos en futuros trabajos.

6.1 Trabajo futuro

• En primer lugar, en cuanto a las prestaciones de la app: las pruebas efectuadas y los resultados obtenidos se han realizado con un único modelo de recipiente. Una de las pretensiones para este TFG fue la posibilidad de hacerlo con varios modelos de recipiente, pero debido al tiempo limitado disponible para su realización, únicamente se ha podido elaborar el trabajo con un único recipiente.

• Otra de las ideas iniciales era disponer de una guía de color para la silueta del recipiente, de tal modo que dependiendo de la proximidad del objeto que se quiere reconocer, ésta cambia de color proporcionando información extra para el guiado hacia el usuario tal y como se muestra a continuación.

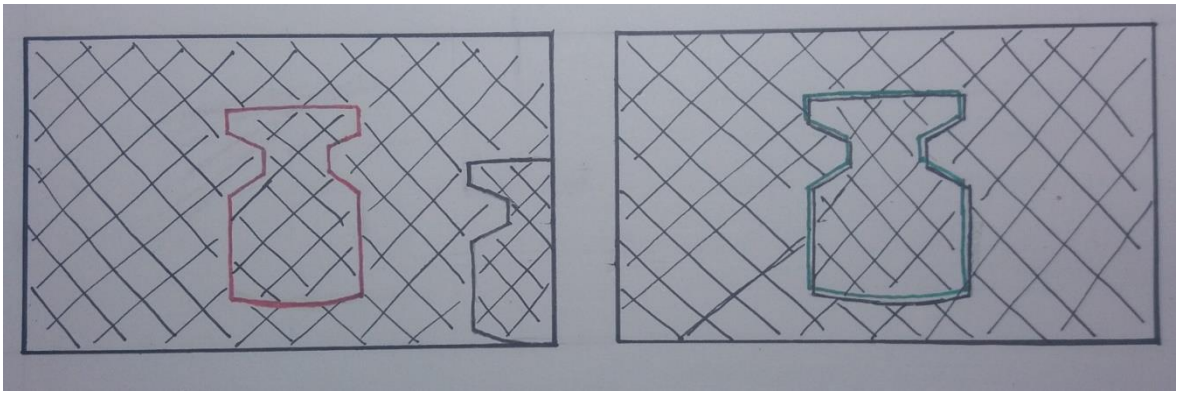

Figura 6-1-2: Idea conceptual interfaz de usuario 2

• Otra de las tareas que se debe llevar a cabo para la mejora de la app es el tema de la compatibilidad entre distintos dispositivos. En este trabajo, la app desarrollada se ha realizado en un único dispositivo móvil con unas dimensiones de ancho y alto determinadas. Esto provoca que el algoritmo de detección únicamente funcione con las imágenes insertadas para esta ocasión, por lo que si se quiere realizar la app para otros dispositivos, con otras dimensiones de pantalla, resulta necesario adaptar la aplicación para otras medidas de altura y anchura.

• Y en último lugar el algoritmo de detección de recipientes transparentes. Lejos queda de ser un modelo altamente eficaz. Existe mucha teoría y técnicas relacionadas con éste ámbito capaz de extraer un mayor rendimiento al método básico propuesto en este TFG.

Referencias

[1] https://histinf.blogs.upv.es/2012/12/14/android/
[2] https://www.xatakandroid.com/sistema-operativo/historia-y-evolucion-de-android-como-un-sistema-operativo-para-camaras-digitales-acabo-conquistando-los-moviles
[3] https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android
[4] http://labibliadelprogramador.blogspot.com/2012/09/estructura-de-android.html
[5] https://developer.android.com/guide/components/activities?hl=es-419
[6] https://www.desarrolloweb.com/articulos/android-que-es-una-activity-o-actividad.html
[7] https://desarrolloweb.com/articulos/ciclo-vida-actividad-aplicacion.html
[8] http://blog.auriboxtraining.com/curso-de-desarrollo-de-aplicaciones-moviles-con-android/intencion/
[9] http://www.proyectosimio.com/es/programacion-android-layouts-vistas/
[10] https://developer.android.com/guide/topics/ui/overview?hl=es-419
[11] https://code.tutsplus.com/es/tutorials/android-from-scratch-understanding-views-and-view-groups--cms-
[12] http://www.tuprogramacion.com/glosario/que-es-el-android-manifest
[13] https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419
[14] https://es.scribd.com/doc/72111058/Historia-Matlab
[15] https://es.mathworks.com/company/newsletters/articles/the-origins-of-matlab.html
[16] https://es.wikipedia.org/wiki/MATLAB
[17] https://opencv.org/about.html
[18] http://mapir.isa.uma.es/varevalo/drafts/arevalo2004lva1.pdf
[19] http://www.cuatrorios.org/index.php?option=com_content&view=article&id=169:opencv-librer%C3%ADa-de-visi%C3%B3n-por-computador&catid=39:blogsfeeds
[20] https://www.universidadviu.es/conceptos-basicos-procesamiento-una-senal-digital/
[21] http://www.monografias.com/trabajos95/procesamiento-digital-de-senales/procesamiento-digital-de-senales.shtml
[22] https://es.wikipedia.org/wiki/Procesamiento_digital_de_se%C3%B1ales
[23] https://blog.infaimon.com/vision-computador-soluciones-permite/
[24] http://dmery.sitios.ing.uc.cl/Prints/Books/2004-ApuntesVision.pdf
[25] https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial
[26] http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/mendieta_d_d/capitulo1.pdf

Glosario

API: Application Programming Interface

APK: Android Application Package

IDE: Integrated Development Environment

EPS: Escuela Politécnica Superior

UAM: Universidad Autónoma de Madrid

SDK: Software Development Kit

Anexos

A Instalación de la librería de OpenCV en Android Studio y dispositivo móvil

INSTALACIÓN LIBRERÍA OPENCV EN EL ENTORNO DE DESARROLLO:

1. Descargar la librería de OpenCV a través de su página web: https://opencv.org/

   En el momento de realizar este paso, la página se mostraba del siguiente modo:

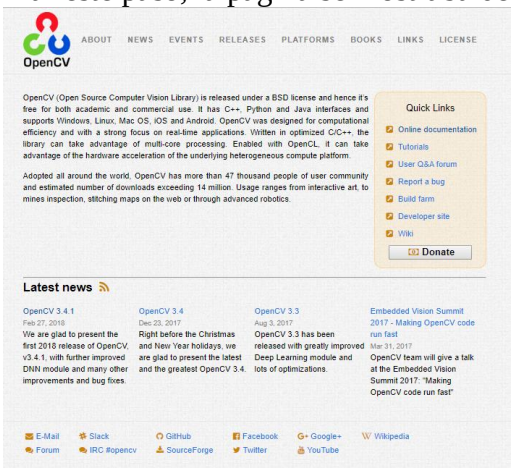

Figura A-1

   En ella podemos ver múltiples versiones de la librería OpenCV.

   Lo que nos interesa es el SDK (Software Development Kit).Descargamos la última versión (en este caso OpenCV 3.4.1):

Figura A-2

SDK: conjunto de herramientas de desarrollo de software que permite crear una aplicación informática para un sistema concreto. En el caso de Android, al basarse su lenguaje de programación en java, precisamente nos interesa el sdk de java para trabajar con la librería OpenCV en el entorno de desarrollo Android Studio.
Una vez descargado el kit, lo extraemos y lo guardamos en un sitio accesible.
Como se puede observar, tenemos el sdk de java y además para C++ (native).

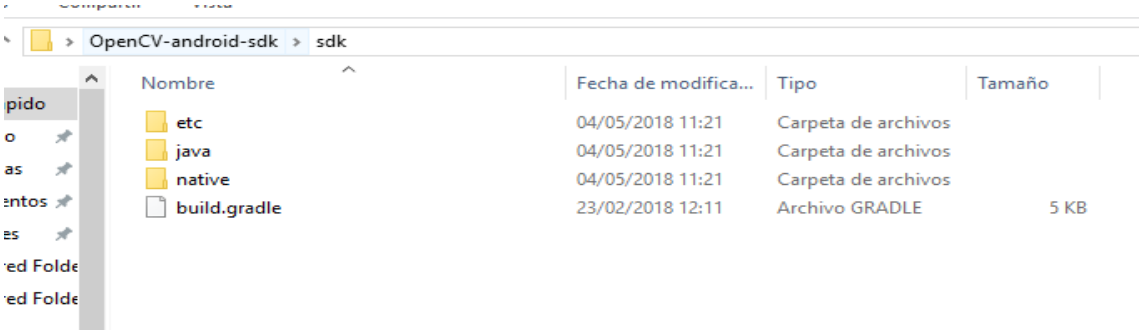

Figura A-3

Una vez completado este primer paso, vamos a Android Studio para integrar la librería.

2. Instalación de la librería de OpenCV en el entorno de desarrollo Android Studio:

Primeramente creamos un nuevo proyecto y nos aseguramos de poner un SDK mínimo adecuado para llegar a los máximos dispositivos posibles en función del

dispositivo donde vayas a realizar las pruebas de tu aplicación. En mi caso usé un API 23 ya que en el momento de realizar este trabajo lo consideré la más apropiada.

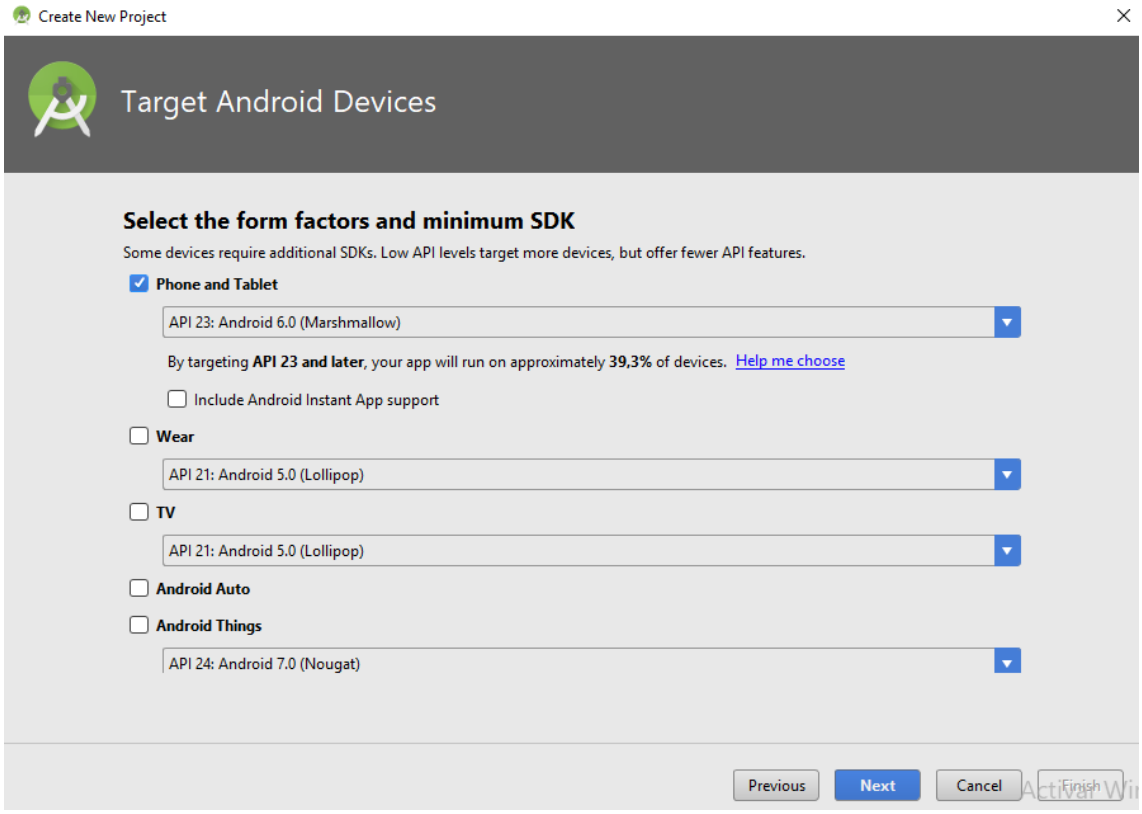

Figura A-4

¡POSIBLE ERROR!: Si nada más comenzar el proyecto salta en la línea de mensajes el siguiente error mostrado en la figura:

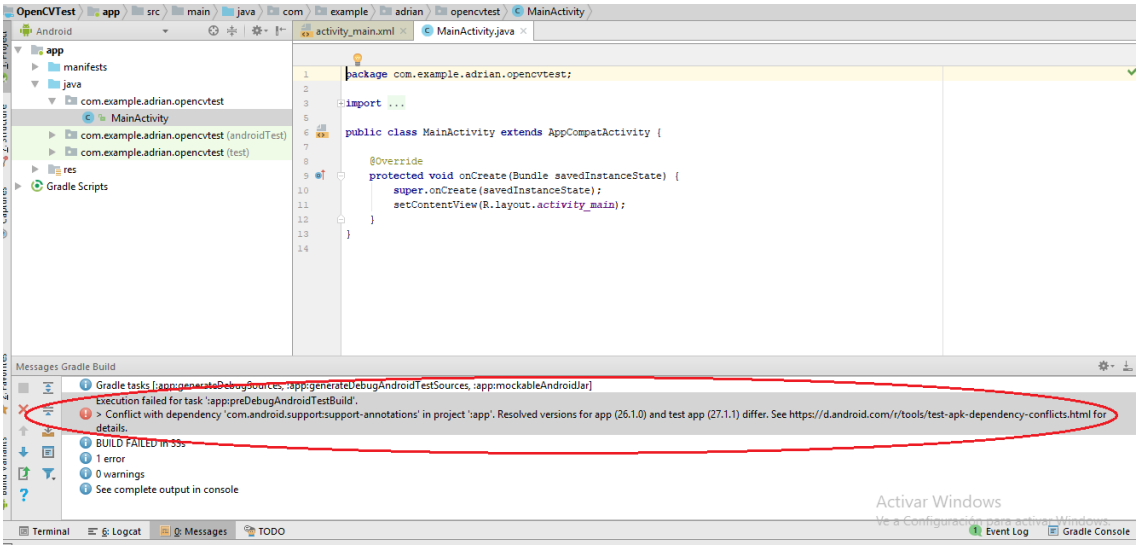

Figura A-5

-Significa que hay un problema de compatibilidad de versiones usadas en el gradle entre la “app” y el “test app”. Una solución para arreglar esto es la siguiente:
-Vaya primero al gradle script del módulo de la aplicación:

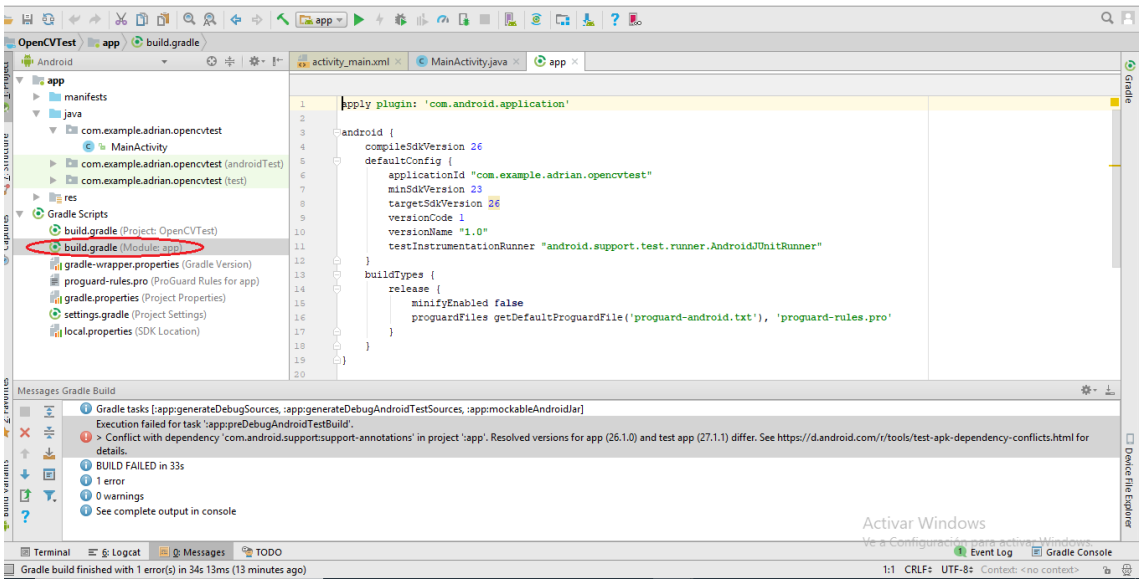

Figura A-6

-Y a continuación, en el script, en la parte de "dependencies", tiene 2 opciones:

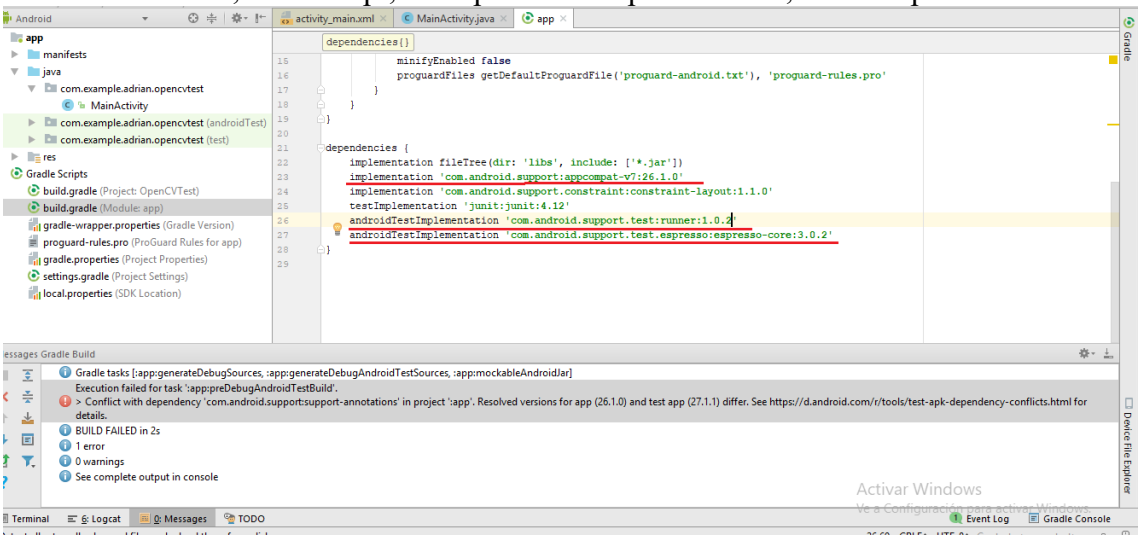

Figura A-7

O bien cambia la versión de la "app" de la 26.1.0 (primera línea destacada en la imagen) a la que se muestra en el error: la 27.1.1. Ó cambia la versión del "test app" a una más antigua (las dos últimas líneas destacadas de la imagen anterior). Yo elegí la segunda opción tal como lo muestro en la siguiente imagen:

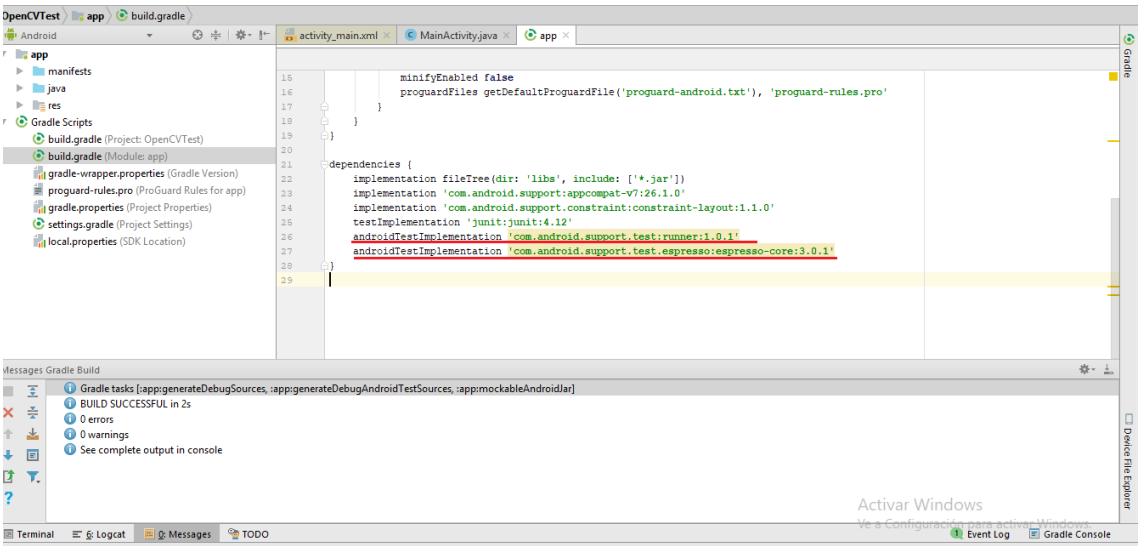

Figura A-8

Después de sincronizar el proyecto para reconocer los cambios, el error estará solventado.

A continuación, una vez creado el nuevo proyecto sin errores,tenemos que cargar la librería en Android Studio. Para eso, nos vamos a FILE→NEW→IMPORT MODULE, y en la casilla de SOURCE DIRECTORY importamos el sdk de java descargado previamente.

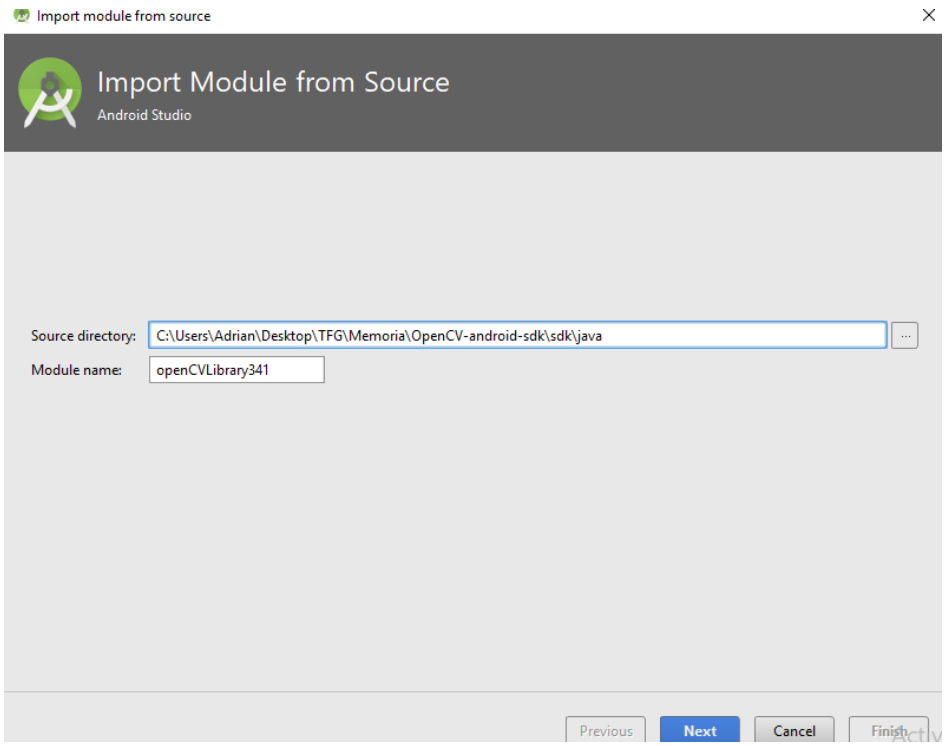

Figura A-9

En la siguiente ventana que nos aparecerá, pulsamos “finish” dejando los parámetros por defecto.

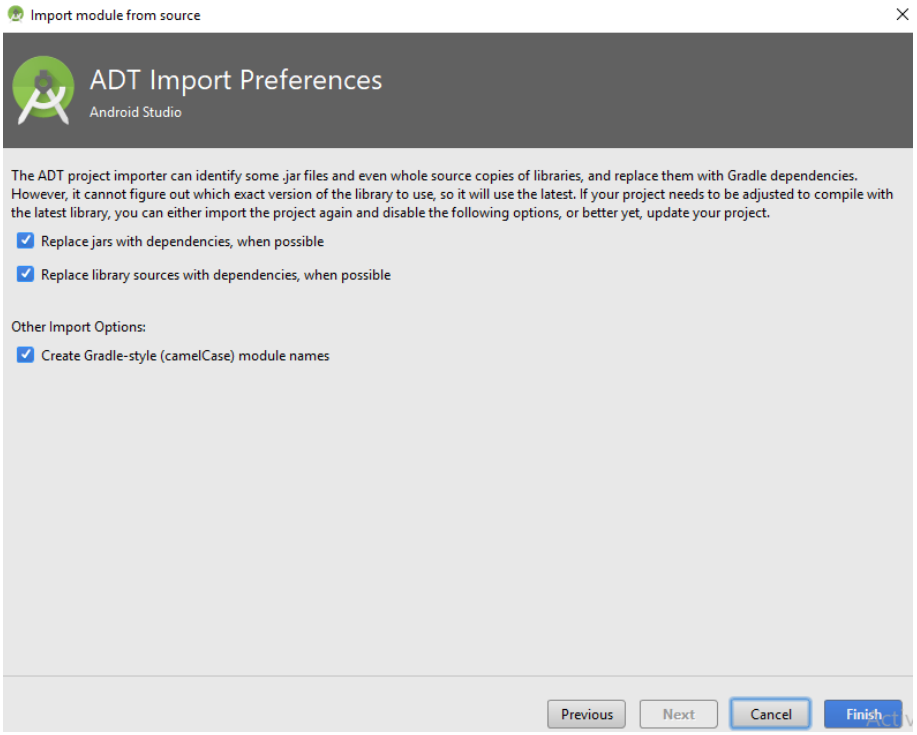

Figura A-10

Una vez cargada la librería de OpenCV dentro del entorno de desarrollo, el siguiente paso es sincronizarla con nuestro proyecto. Para ello tenemos que abrir dos gradle scripts: el script de la nueva librería que tiene que aparecer automáticamente luego del paso anterior, y el script delmódulo de la "app":

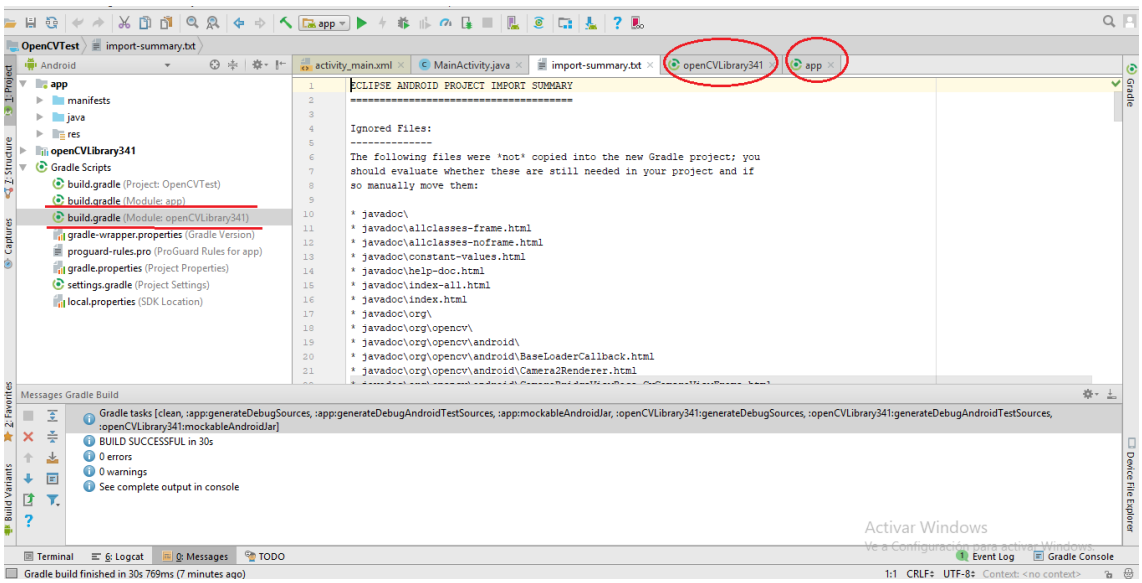

Figura A-11

Una vez abierto los dos scripts, tenemos que poner en común los siguientes parámetros: "compileSdkVersion", "minSdkVersion" y "targetSdkVersion". Para ello tomamos como referencia los valores de nuestra "app" y cambiamos los de la nueva librería adaptándolos a nuestra aplicación:

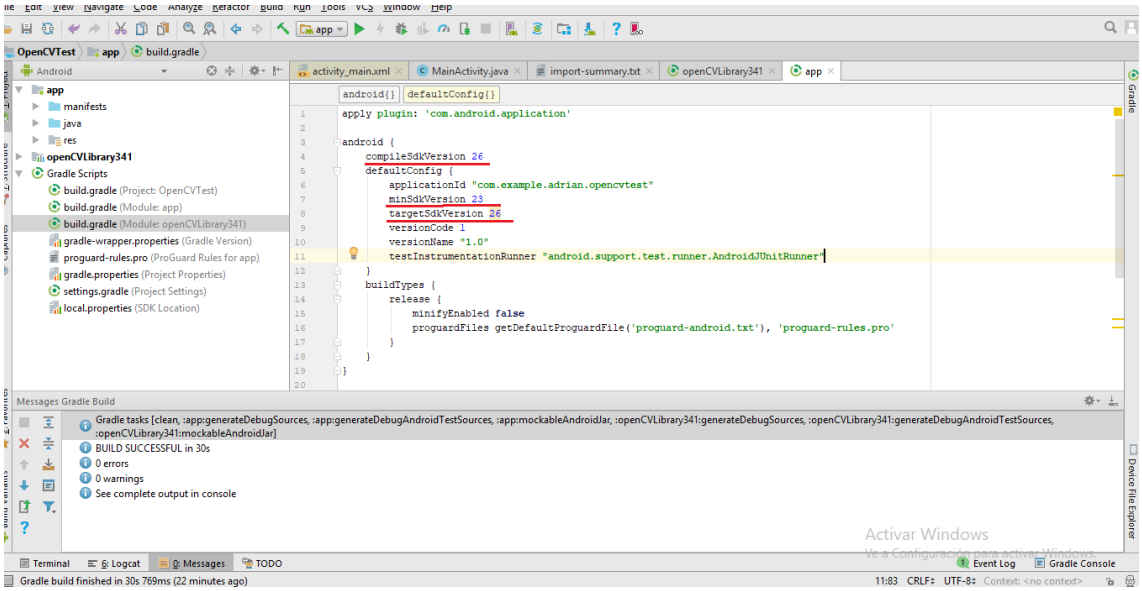

Figura A-12: Gradle Script de nuestra app

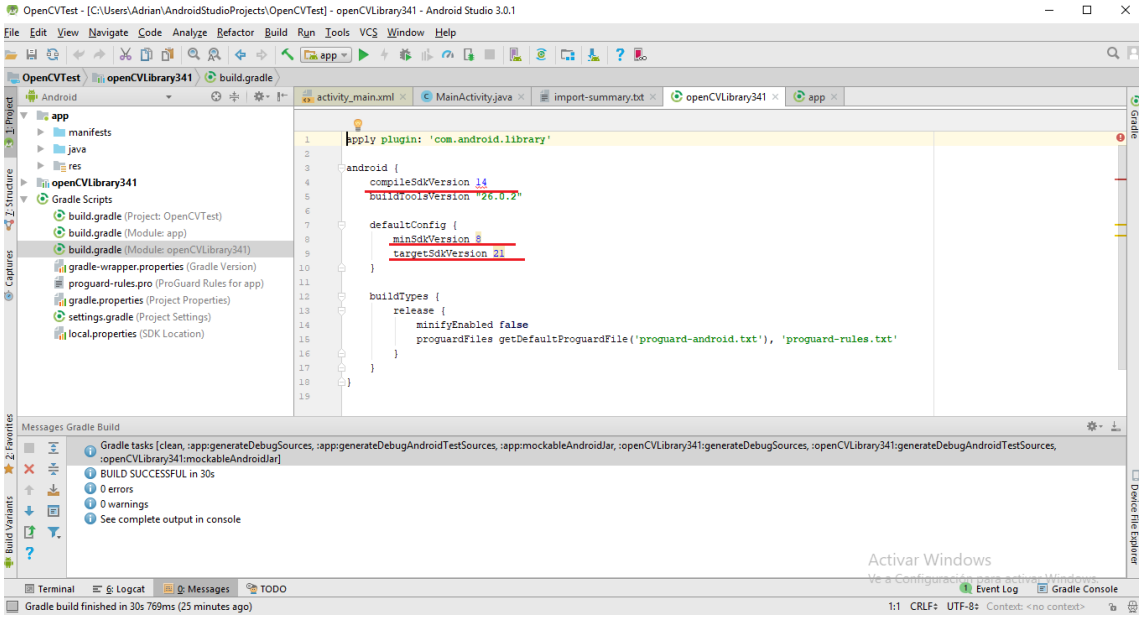

Figura A-13:Gradle Script de la librería de OpenCV sin cambiar

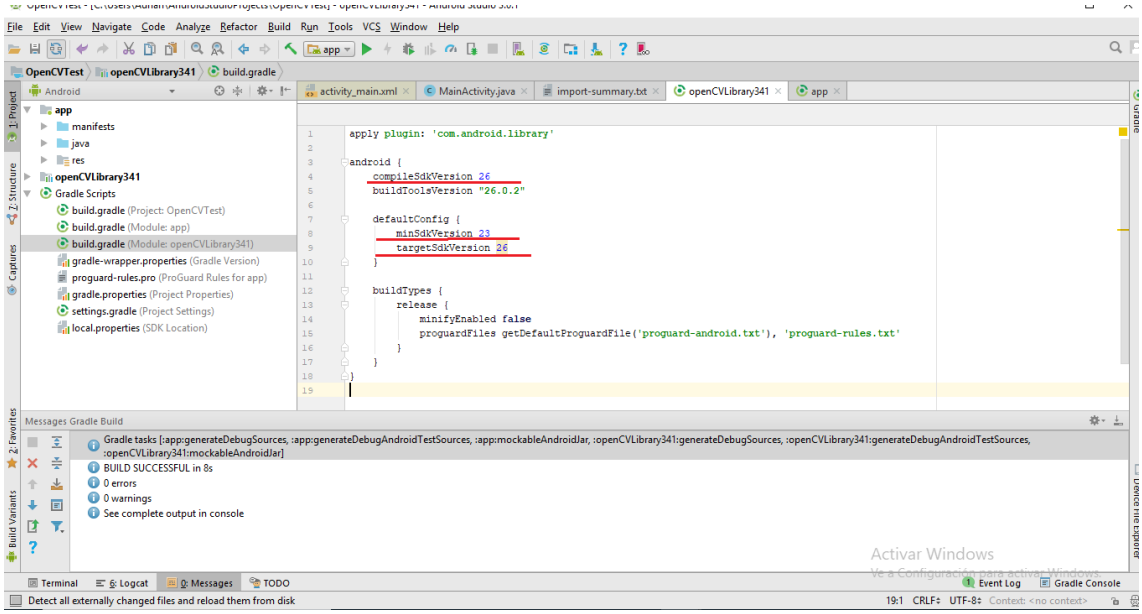

Figura A-14: Gradle Script de la librería de OpenCV adaptado al de nuestra "app"

Posteriormente tenemos que crear la dependencia del proyecto con la librería OpenCV. Nos dirigimos a FILE→PROJECT STRUCTURE y nos saldrá la siguiente ventana:

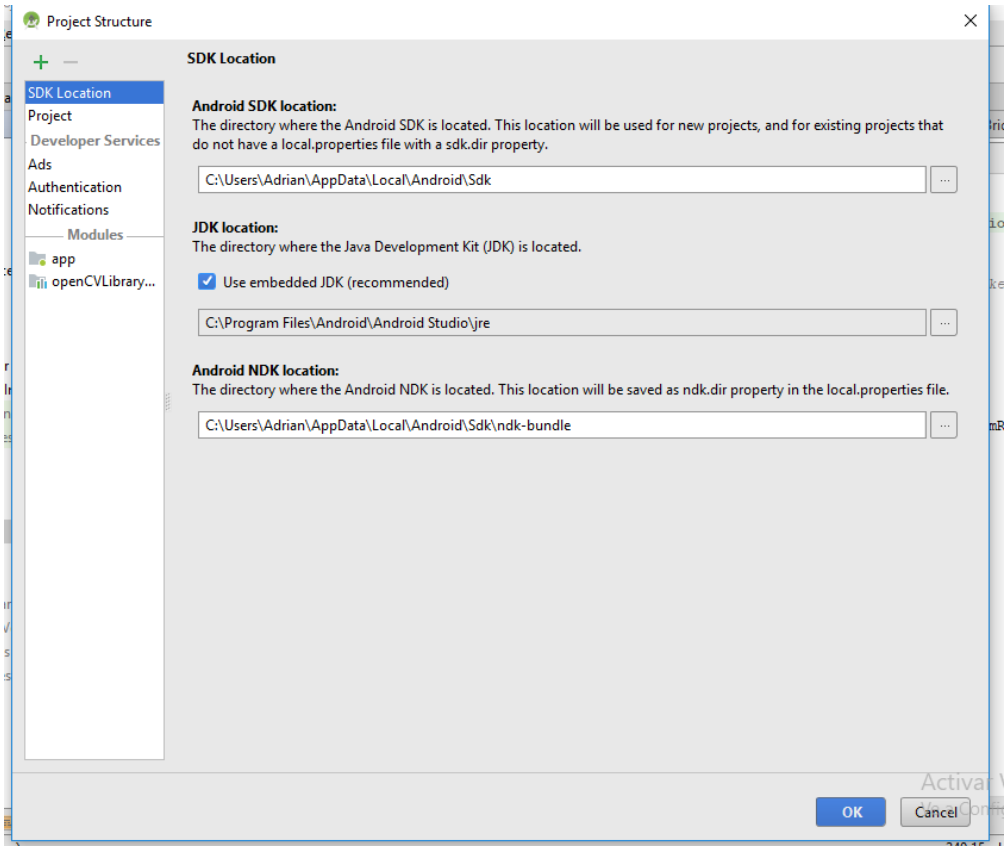

Figura A-15

En la columna de la izquierda, vamos a “app” y a continuación a la pestaña “Dependencies”:

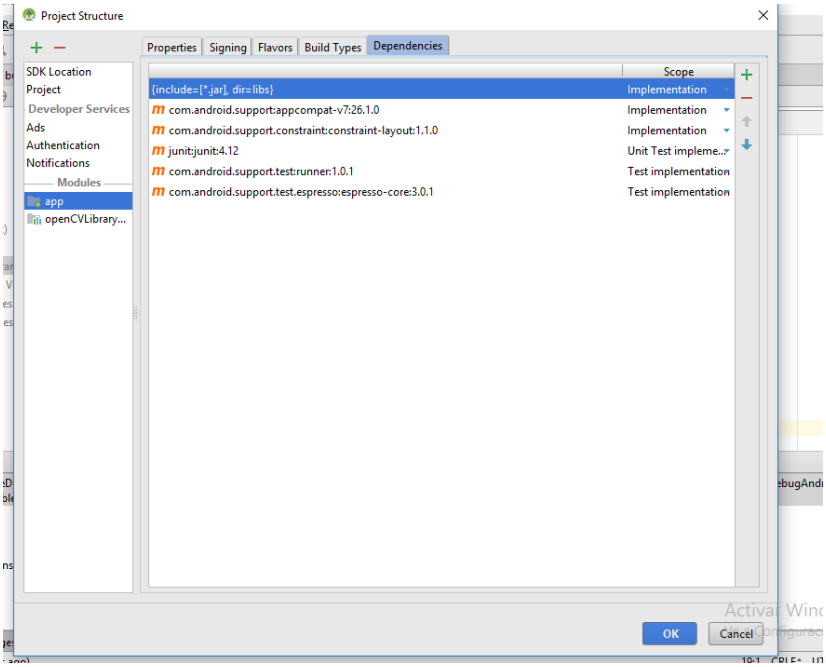

Figura A-16

Pulsamos en “add” (dentro del círculo rojo) y seleccionamos la opción 3, “Module dependency”:

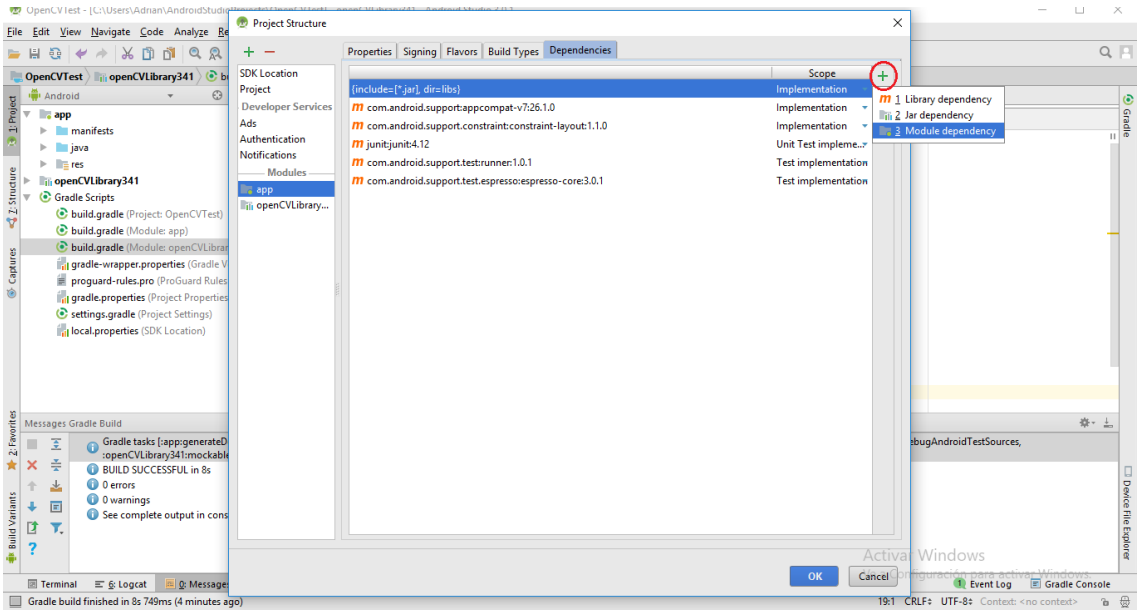

Figura A-17

Se nos tiene que mostrar nuestra última versión de OpenCV tal como aparece en la imagen:

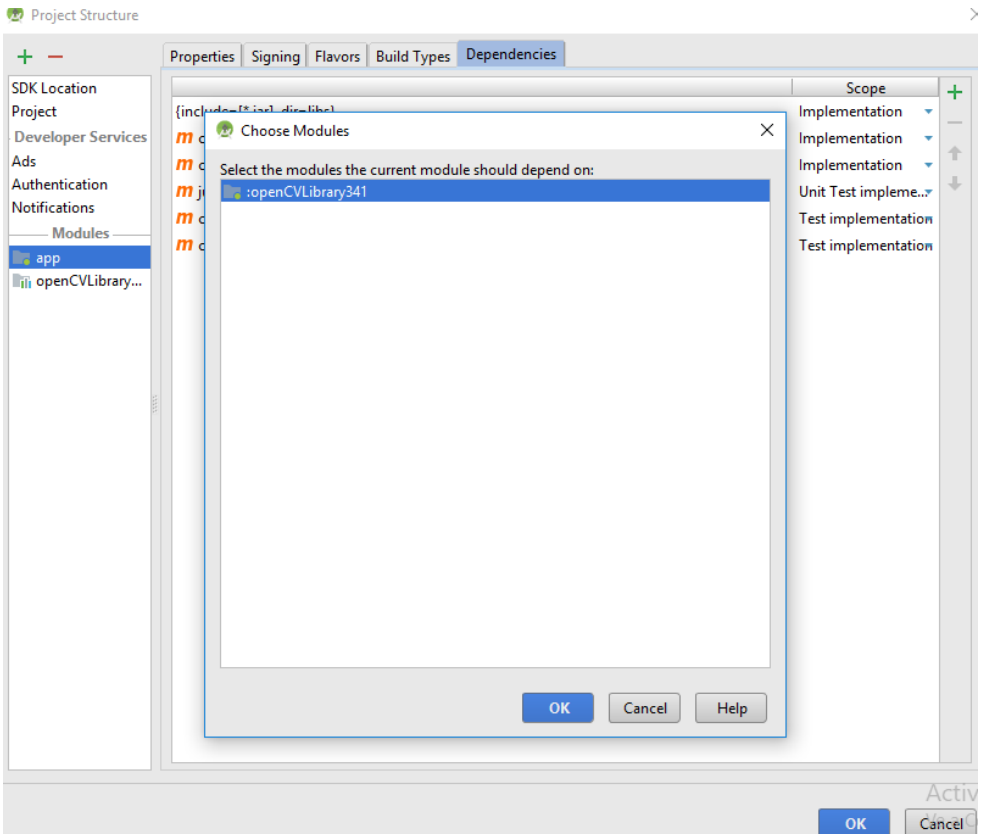

Figura A-18

Y volvemos a donde estábamos pulsando “ok” en las ventanas.

Una vez introducido OpenCV en el entorno de desarrollo y haberlo sincronizado haciendo una dependencia con nuestro proyecto, el siguiente paso es “atar” las librerías para trabajar con ellas en nuestra aplicación.
Para completar este último paso, en la pestaña ”Project”, de nuevo en modo de visualización “Android”, pulsamos con el botón derecho del mouse en la carpeta “app” y seguimos con NEW→FOLDER→JNI FOLDER, como se muestra en la imagen:

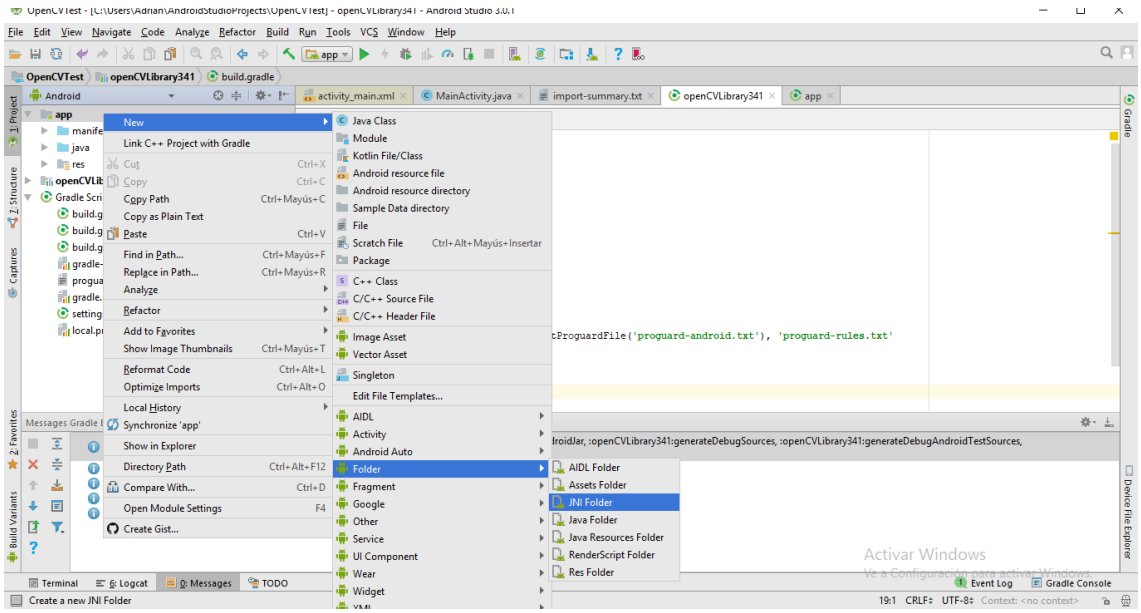

Figura A-19

A continuación, se nos mostrará una ventana de configuración en donde habrá que marcar el box que nos permite cambiar la localización de la carpeta ("Change Folder Location") y posteriormente indicamos el nombre de la subcarpeta donde se almacenarán las librerías con el nombre específico "jniLibs" tal como se muestra en la siguiente imagen:

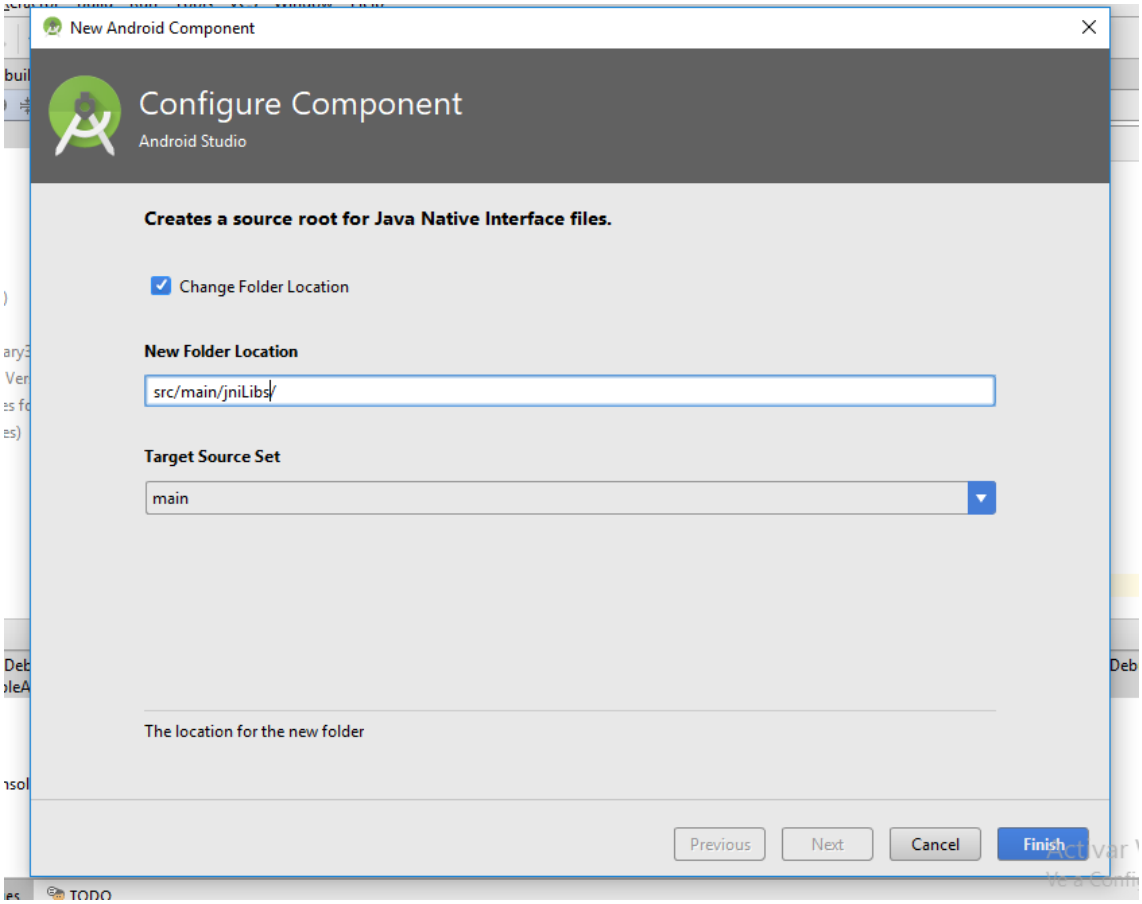

Figura A-20

¡POSIBLE ERROR!: En teoría se ha tenido que crear una subcarpeta con ese nombre que puedes observar junto con las demás subcarpetas (“manifests”, “java” y “res”) dentro de “app”. Pero si en lugar de eso vemos una subcarpeta con el nombre “cpp” (imagen 1), significa que es posible que ya hayas creado la carpeta JNI en algún proyecto anterior.

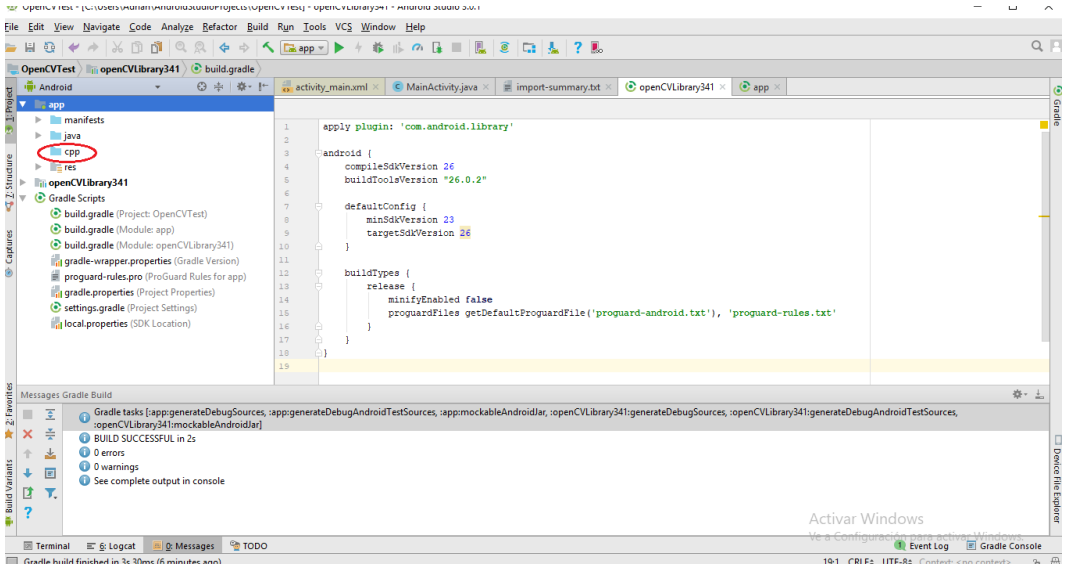

Figura A-21

Para arreglar esto nos vamos de nuevo al gradle script de la “app” y localizamos la siguiente línea:

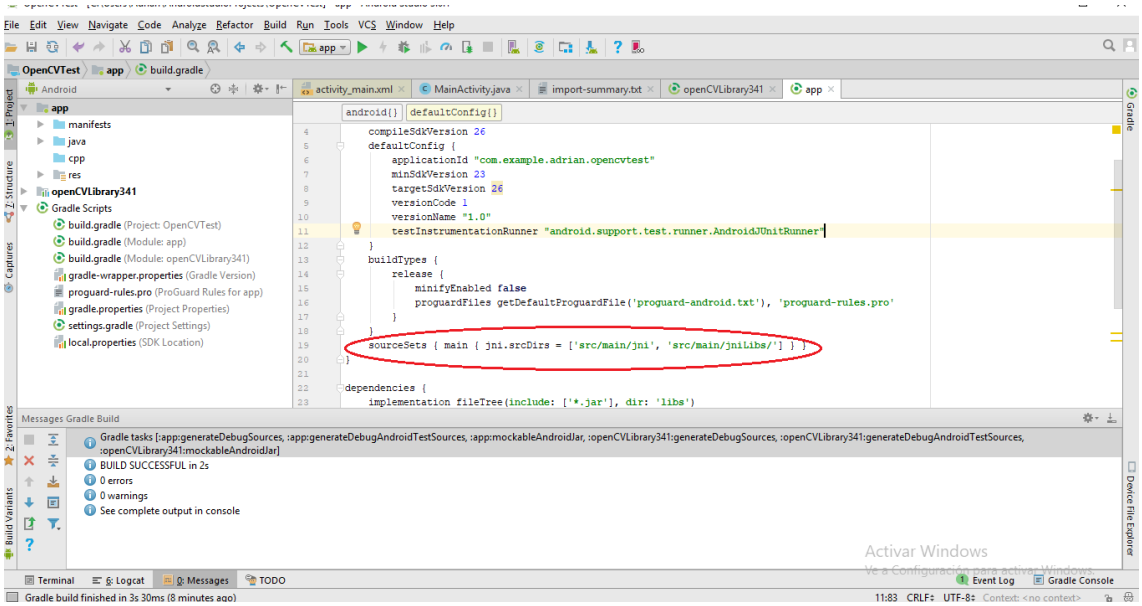

Figura A-22

Simplemente la borramos y luego de sincronizar el proyecto para que te reconozca los cambios, automáticamente tienes tu subcarpeta “jniLibs” creada correctamente:

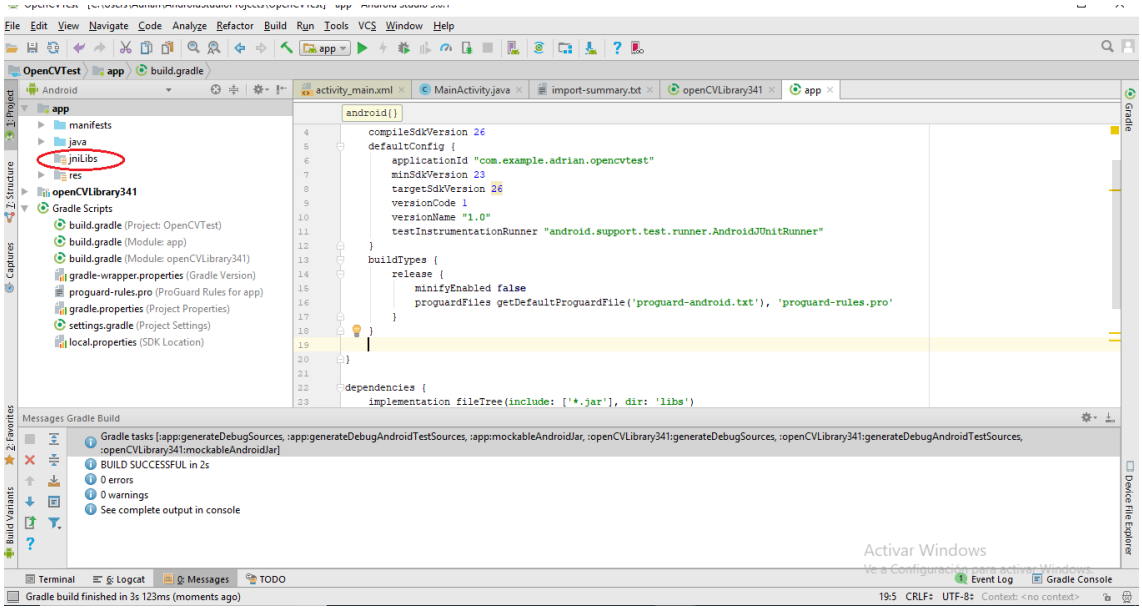

Figura A-23

A continuación tenemos que copiar las librerías descargadas anteriormente en el paso 1 y pegarlas en esta nueva subcarpeta.
Para copiar las librerías accedemos a ellas a través de la carpeta descargada siguiendo la ruta análoga paracada usuario, que se muestra en la siguiente imagen:

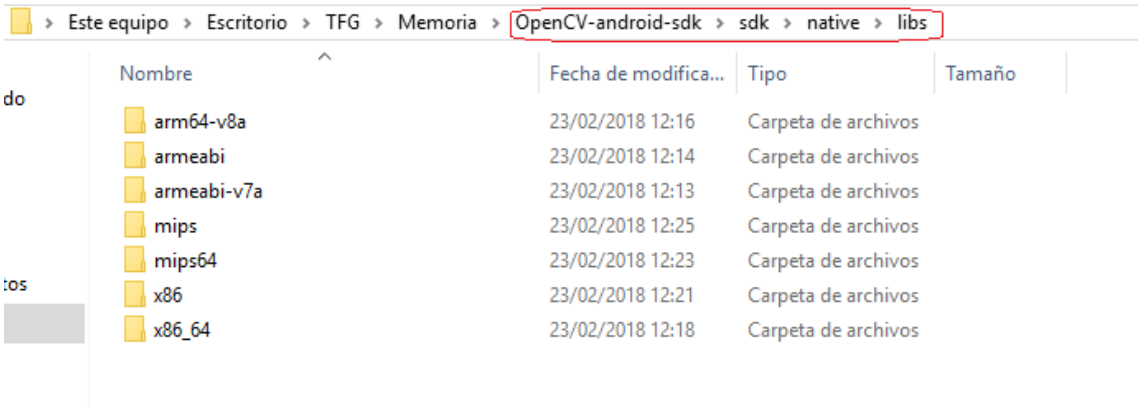

Figura A-24

Seleccionamos todas las librerías para copiarlas y volvemos a Android Studio para pegarlas. Dentro de Android Studio, siguiendo la misma configuración de vista de carpetas que dejamos (pestaña “Project” y modo de visualización “Android”), pulsamos en la carpeta destino con el botón derecho para proceder al pegado:

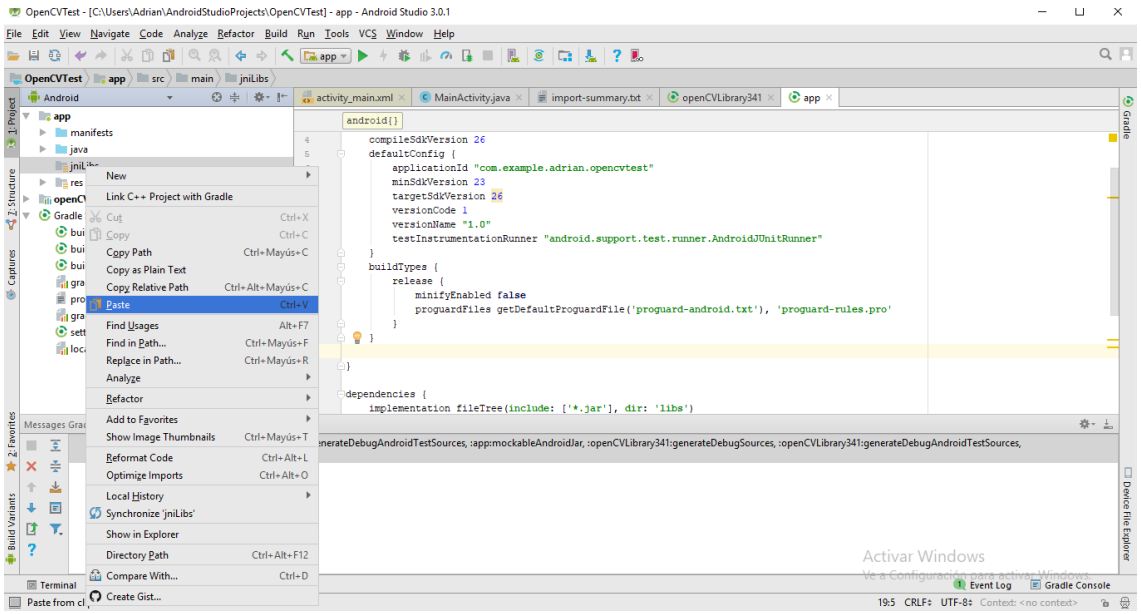

Figura A-25

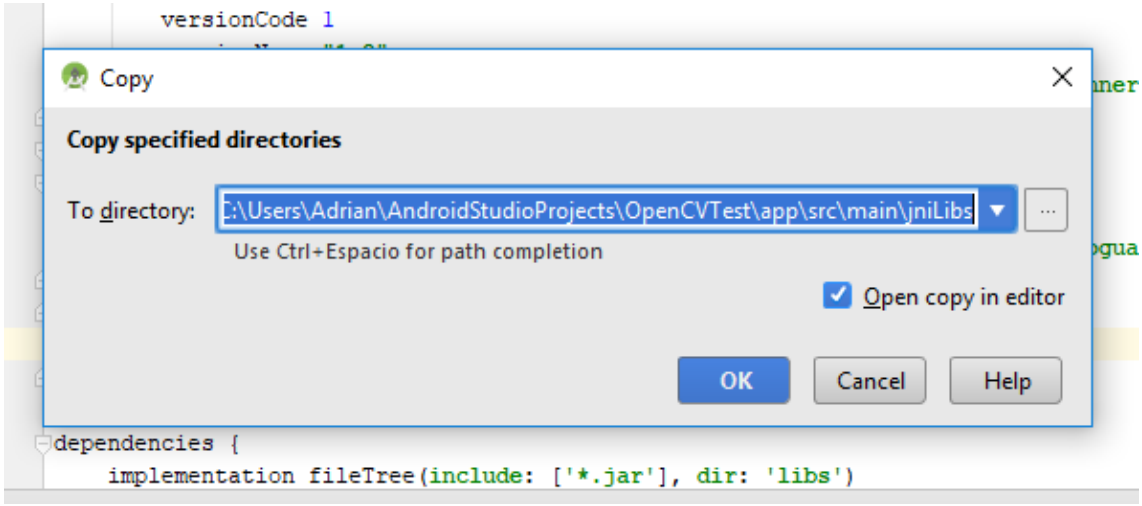

Figura A-26

Una vez guardadas las librerías se puede comprobar que efectivamente las tenemos copiadas, visualizándolas en la pestaña “Project” y vista “Android” dentro de la carpeta “app”:

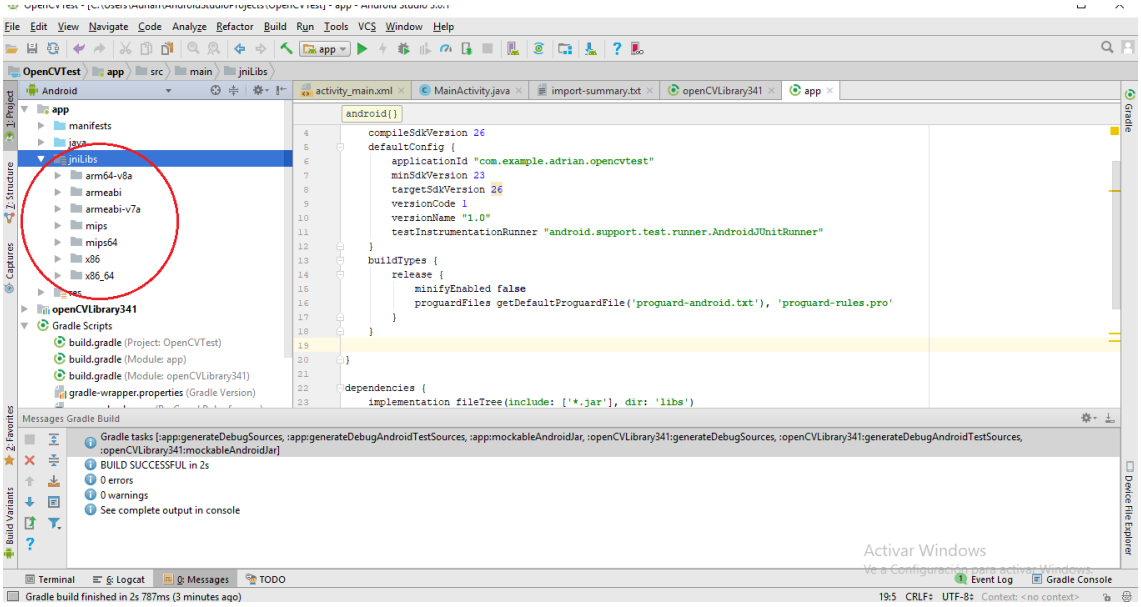

Figura A-27

Y en teoría ya estarías listo para trabajar con las librerías de OpenCV para Android. Si se desea hacer una última comprobación, se puede ejecutar la aplicación comprobando que realmente está cargado OpenCV del siguiente modo:

-Meta las siguientes líneas de código en la actividad principal que te permiten, una vez lanzada la aplicación, comprobar que efectivamente están cargadas y listas para su uso las librerías de la última versión de OpenCV:

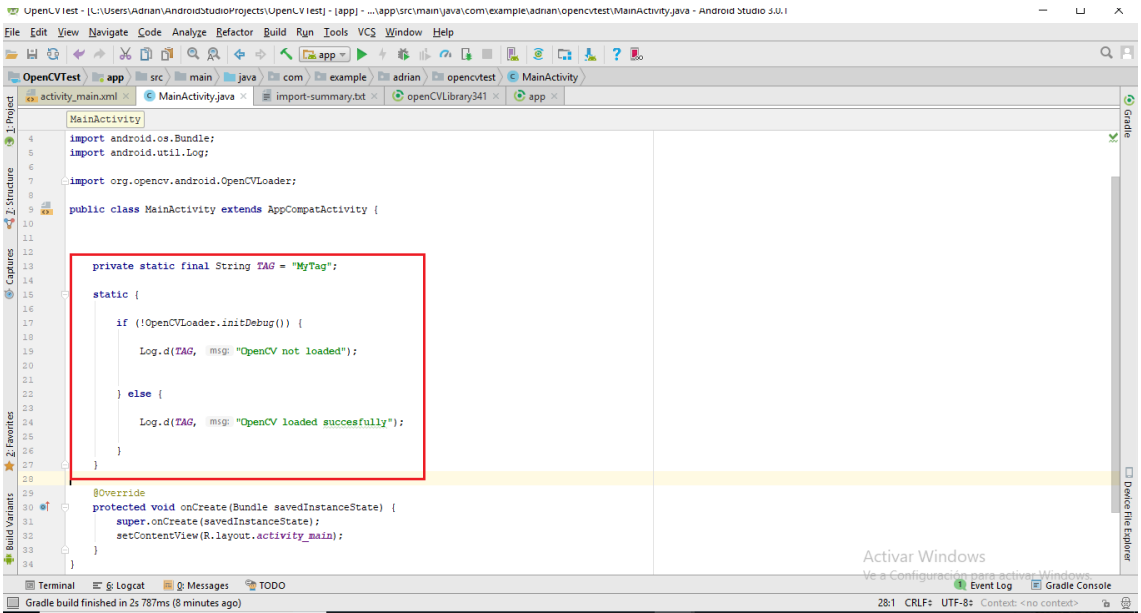

Figura A-28

-Una vez compilada y ejecutada la aplicación, en el Logcat, podrá verse el mensaje de confirmación:

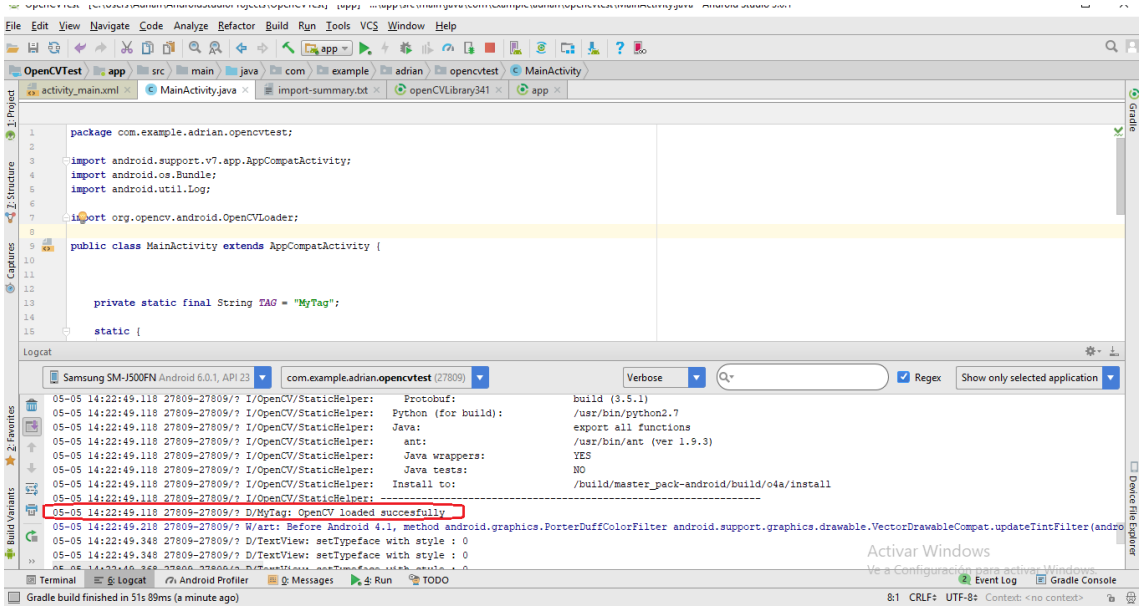

Figura A-29

INSTALACIÓN LIBRERÍA OPENCV EN EL DISPOSITIVO MÓVIL:

Con los pasos descritos en el punto anterior, conseguimos integrar las librerías de OpenCV en el entorno de desarrollo de Android para usar así sus métodos y clases que nos facilitan el manejo de imágenes en tiempo real. Una vez hecho esto, para probar nuestra aplicación que usa dichas librerías en el móvil, es necesario la instalación de OpenCV en el dispositivo.

Para su instalación en un dispositivo Android, siga los siguientes pasos:

1. Acceda a Play Store y busque “opencv”.

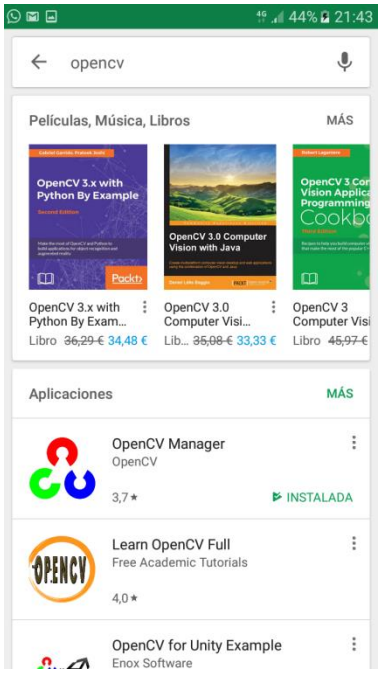

Figura A-30

2. La aplicación que nos interesa se llama “OpenCV Manager”. Simplemente descárguela y automáticamente se instalará en su dispositivo lista para ser usada por su aplicación.

B Preparación del recipiente deseado como molde para su detección

En el siguiente epígrafe, se detallarán los pasos seguidos en la obtención de la silueta de un recipiente transparente para su uso tanto como imagen guía como máscara para el algoritmo de detección de objetos pensado para la app.

-En primer lugar hay que tomar una foto del recipiente con el mayor contraste posible. Para ello y como primer paso, hay que "preparar" el recipiente. Se vertió el líquido de una lata de Coca-Cola dentro del recipiente vacío hasta llenarlo del todo y después, con un fondo blanco, se tomó una foto con la cámara del móvil en la posición en la que se deseaba que el objeto fuera reconocido. Para potenciar más el contraste buscado, se usó el postprocesado de imágenes que el mismo dispositivo móvil tiene habilitado.

Figura B-1: Bote1TomaA

-En segundo lugar, para no obtener imágenes no deseadas en pasos posteriores, se retocó la foto con el programa Paint (integrado prácticamente en cualquier ordenador personal) y así volver a potenciar más aún la diferencia entre el recipiente y el fondo en ciertas partes de la imagen.

Figura B-2: Bote1TomaA_ArregladoConPaint

-En este tercer paso, se extrajo la silueta del recipiente propiamente dicha. Es un paso que a su vez incluye 3 etapas programadas en Matlab que se detallan a continuación:

-Binarizado de la imagen entera.
-Uso del algoritmo de detección de contornos Canny.
-Pequeña dilatación de contornos detectados.

El resultado obtenido es mostrado a continuación:

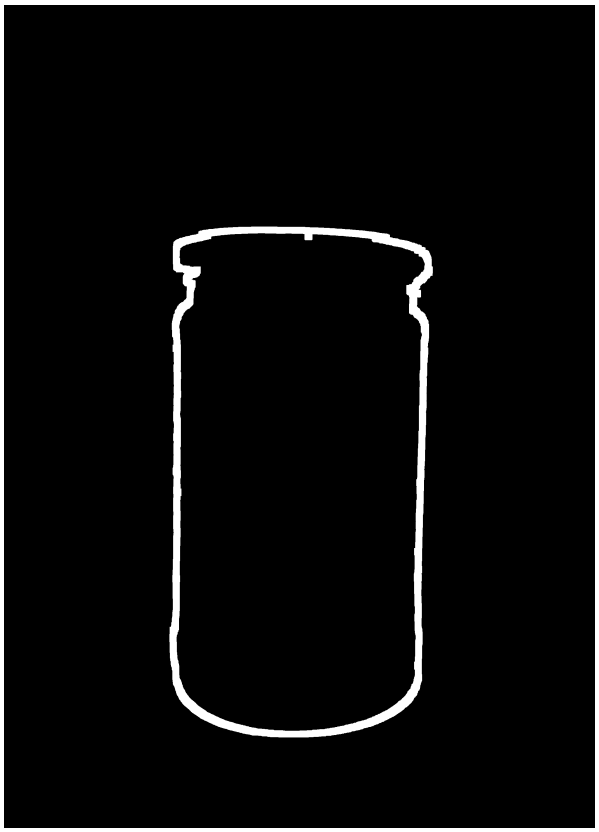

Figura B-3: Bote1TomaA_ArregladoConPaint_Binarizado+Canny+Dilatado25

-En el cuarto paso, se procedió a escalar la silueta de la imagen. El verdadero motivo de esto se encuentra en el siguiente paso, pero básicamente se debe a que en todos estos momentos se ha tratado con imágenes de unas dimensiones superiores a las dimensiones de las imágenes que nuestro dispositivo móvil maneja internamente en la aplicación. Para crear una imagen adecuada para su manejo dentro de la aplicación, es decir, que sea compatible con las imágenes captadas en tiempo real, es necesario crear otra nueva con las mismas dimensiones. Proceso por el cual se empieza escalando la silueta del recipiente del paso anterior.

Para ello se ha vuelto a usar Matlab reduciendo un factor 0.2 la silueta del recipiente obtenida en el anterior paso:

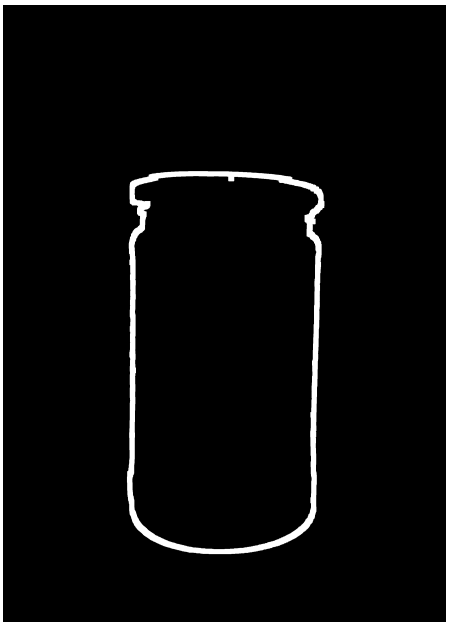

Figura B-4: Bote1TomaA_ArregladoConPaint_Binarizado+Canny+Dilatado25_Escalado020

Para que se aprecie la operación de escalado:

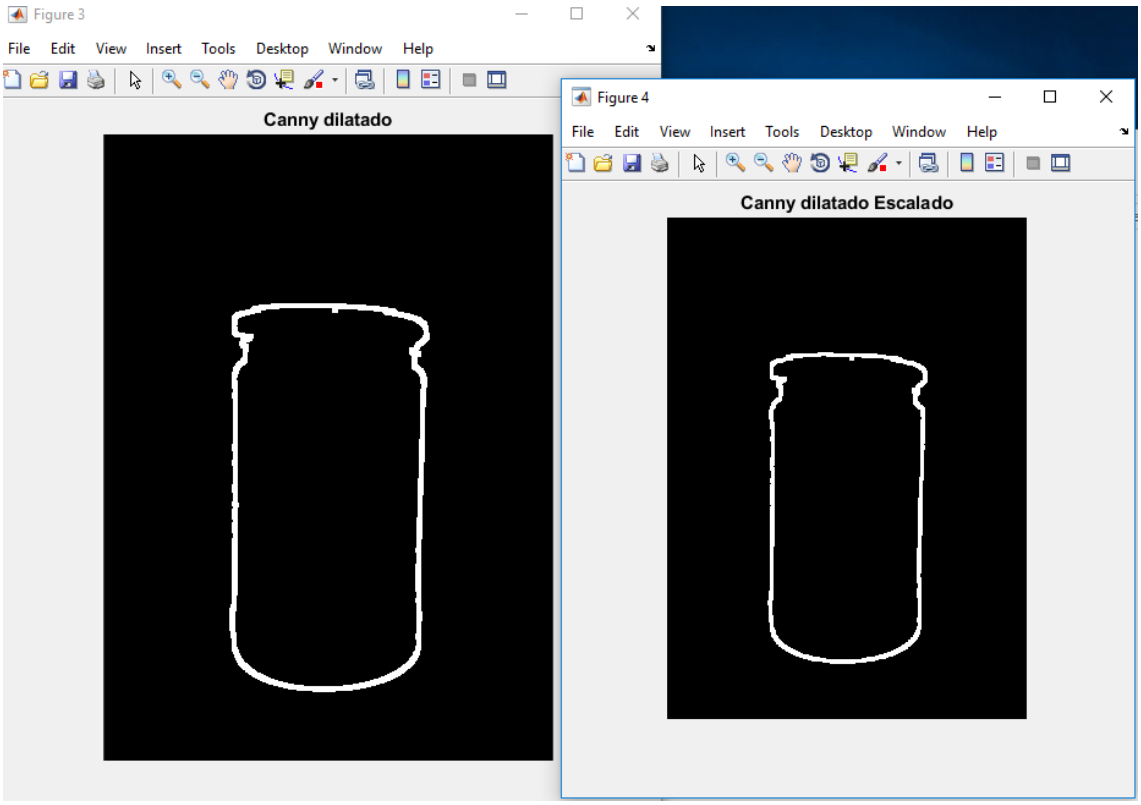

Figura B-5: Comparación de imágenes escaladas

-Para el quinto paso, se introdujo la silueta del recipiente del paso anterior en una nueva imagen con las dimensiones requeridas para la pantalla del dispositivo.

En nuestro caso, en todo el proceso con el tratamiento de imágenes se operó con aquellas de dimensiones de 800píxeles de ancho por 480 píxeles de alto tal como se muestra a continuación:

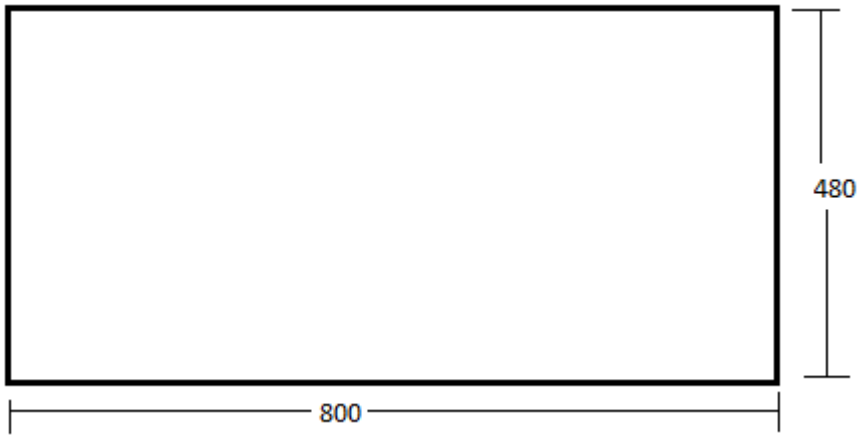

Figura B-6: Dimensiones de las imágenes que se usan en la app

Para crear imágenes nuevas con las dimensiones requeridas se utilizó un editor de imágenes online llamado "pixlr". Dicho editor es de acceso gratuito y su uso es muy intuitivo. Se puede acceder a él a través de la dirección: https://pixlr.com/editor/ .

Una vez accedido, se nos mostrará una pantalla de inicio como la siguiente:

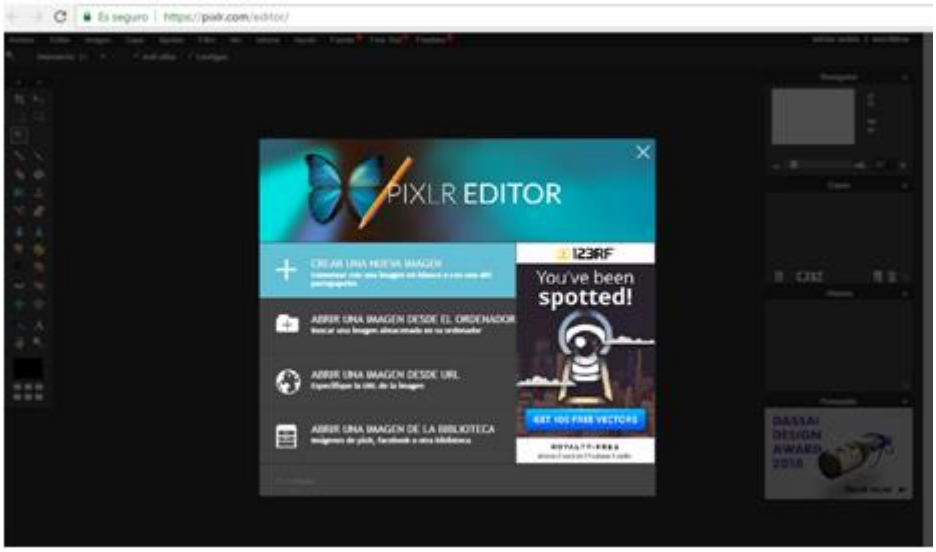

Figura B-7: Pantalla de inicio de "pixlr"

En ella creamos una nueva imagen con las dimensiones deseadas tal y como se muestra:

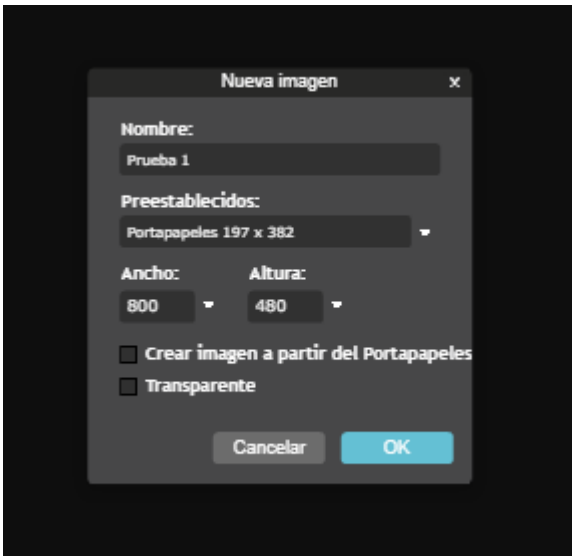

Figura B-8: Creación de una nueva imagen máscara con las dimensiones deseadas

En este paso tenemos dos opciones: podemos o bien crear la imagen guía ó bien podemos crear la imagen usada en la app como máscara. Ambas tienen las mismas dimensiones pero para la imagen guía se marca el box "Transparente" para que no haya nada de fondo y solamente se tenga lo que es la silueta en la imagen.

Se continuó como si estuviésemos creando la imagen máscara y al finalizar esta parte del proceso se mostrará de qué modo se crea la imagen guía.

Para crear la imagen máscara, se necesitó un fondo de color negro. Para ello, se hizo uso de la función "bote de pintura" que el mismo editor proporciona.

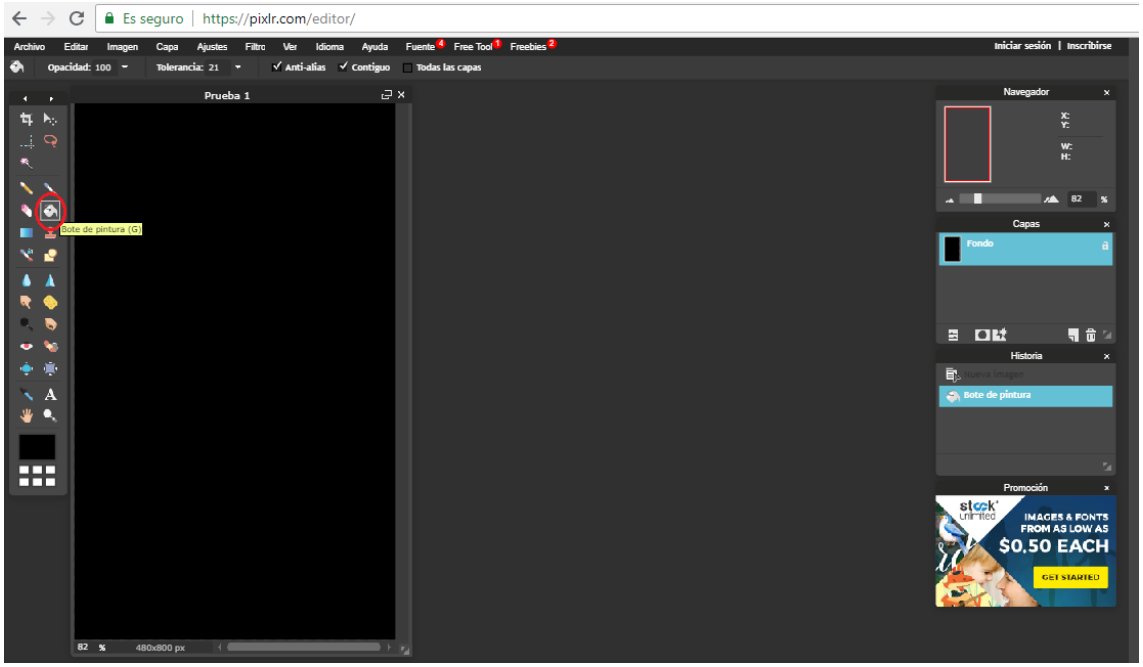

Figura B-9: Relleno del fondo de la imagen con negro

A continuación, hay que copiar la silueta de nuestro recipiente en esta nueva imagen. Para ello,enpixlr, sin cerrar esta imagen nos vamos a *Archivo→Abrir Imagen*, y seleccionamos la imagen con nuestra silueta del recipiente.

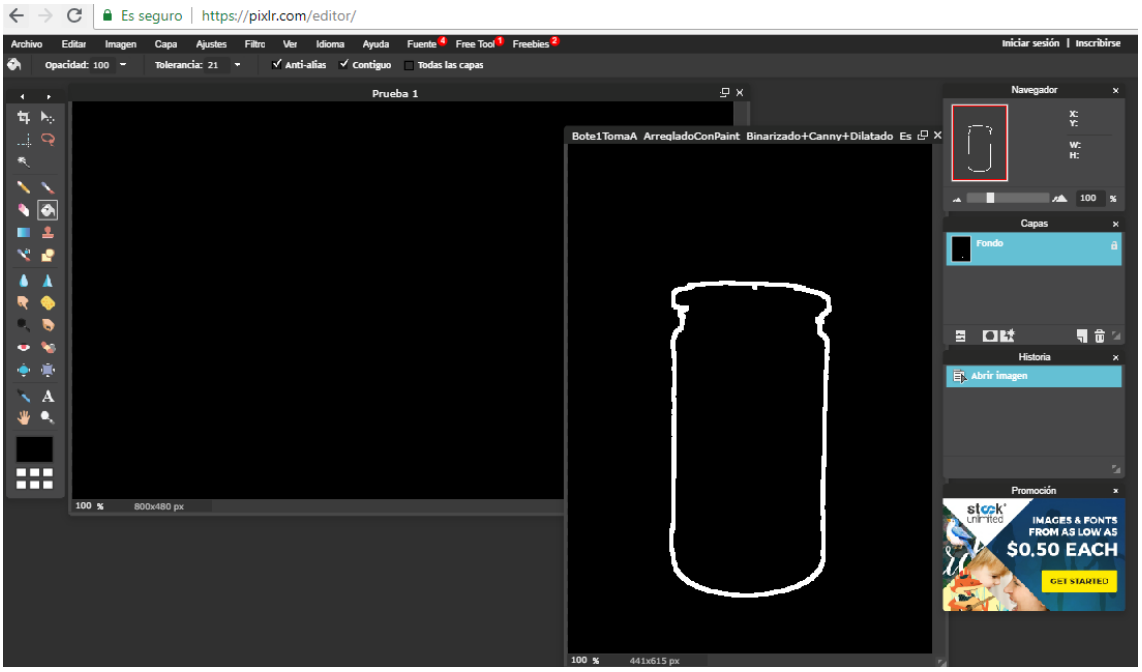

Figura B-10: A la izquierda la imagen en negro con las dimensiones requeridas y a la derecha la última imagen del recipiente creada con Matlab

En este momento, para seleccionar únicamente la totalidad de la silueta blanca de la imagen de la derecha usamos la función "varita" que el editor facilita.

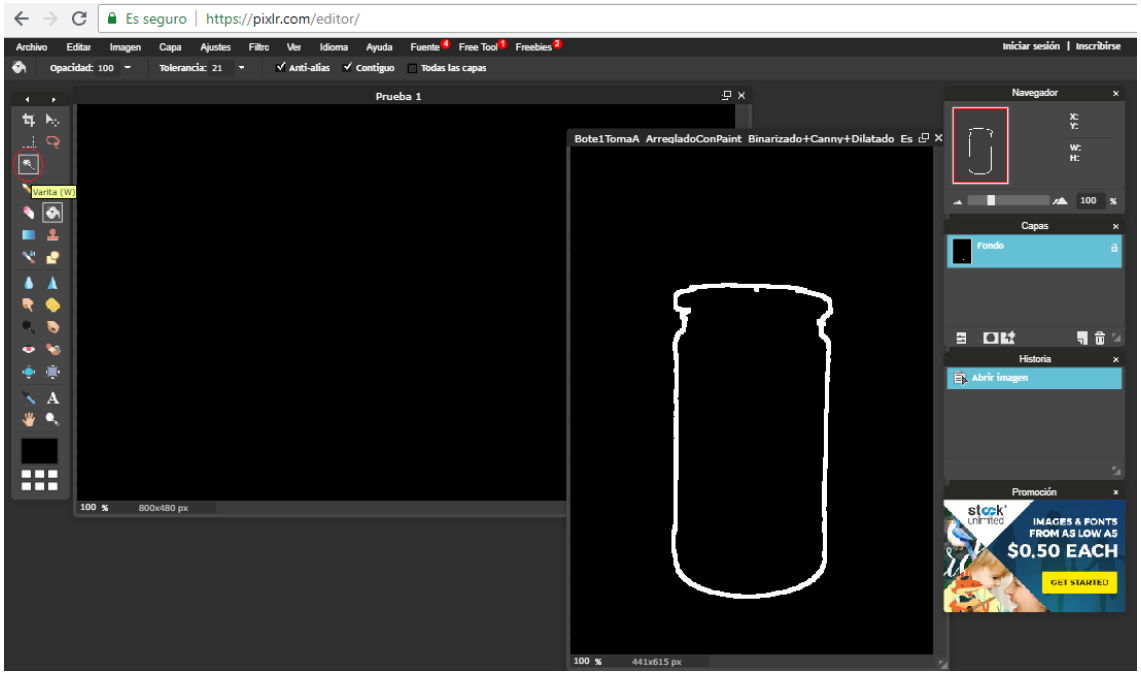

Figura B-11: Función "varita" que el editor de imágenes dispone

A continuación, en la imagen de la derecha simplemente pinchamos en cualquier punto de la región blanca de la silueta y automáticamente se selecciona:

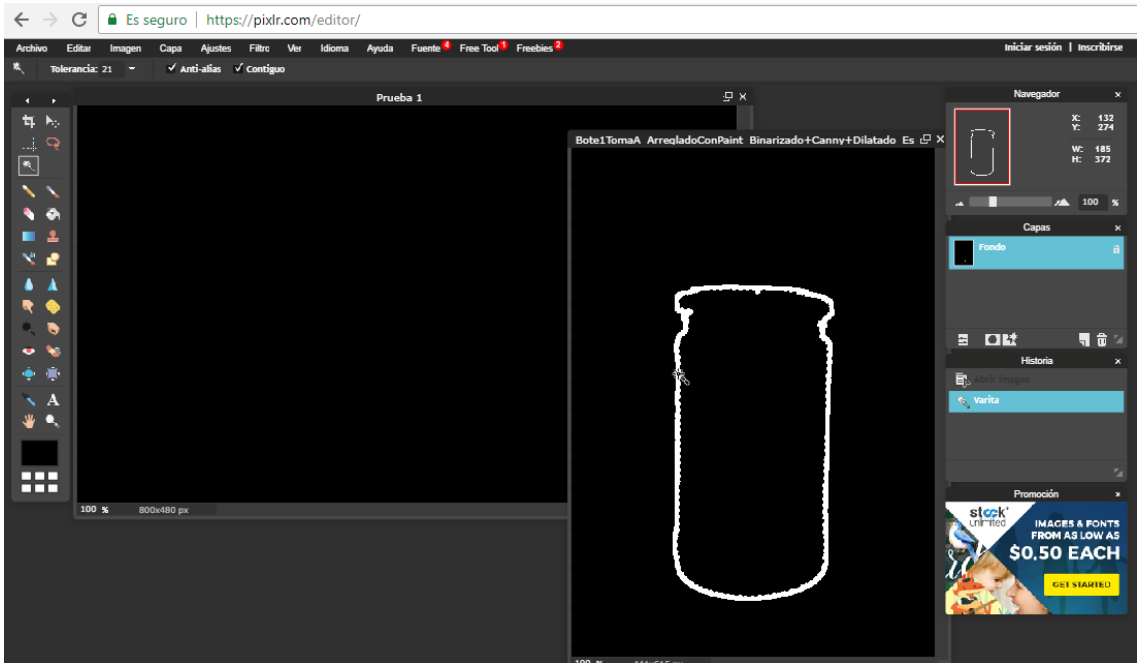

Figura B-12: Región blanca seleccionada con la función "varita"

Ahora simplemente copiamos y pegamos en la imagen de la izquierda:

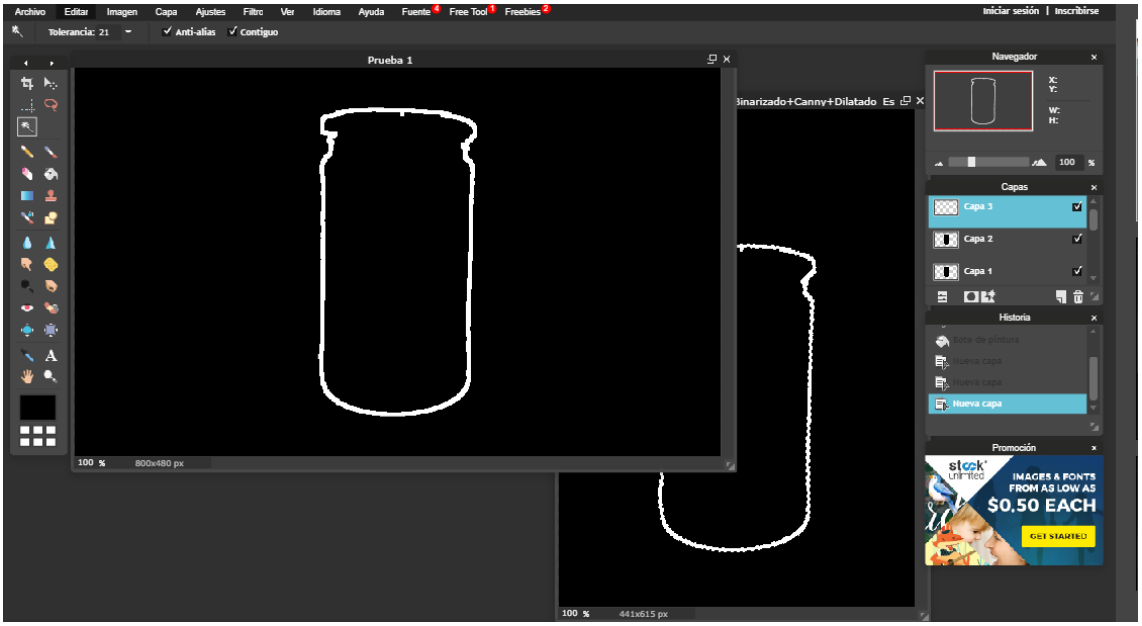

Figura B-13: Silueta del recipiente en la imagen máscara con las dimensiones deseadas

Guardamos la imagen en formato PNG (importante que sea en dicho formato) y ya se tendrá la imagen para ser usada como máscara.

Para la creación de la imagen guía volvemos a empezar con otra imagen nueva con las dimensiones deseadas pero esta vez con el box "Transparente" activado.

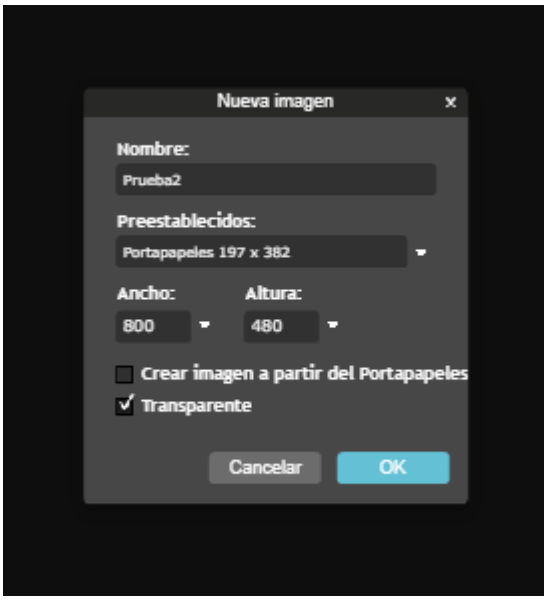

Figura B-14: Creación de una nueva imagen guía con las dimensiones deseadas

A continuación volvemos a realizar un proceso parecido al anterior con la imagen máscara: Cargamos la imagen con la silueta del recipiente:

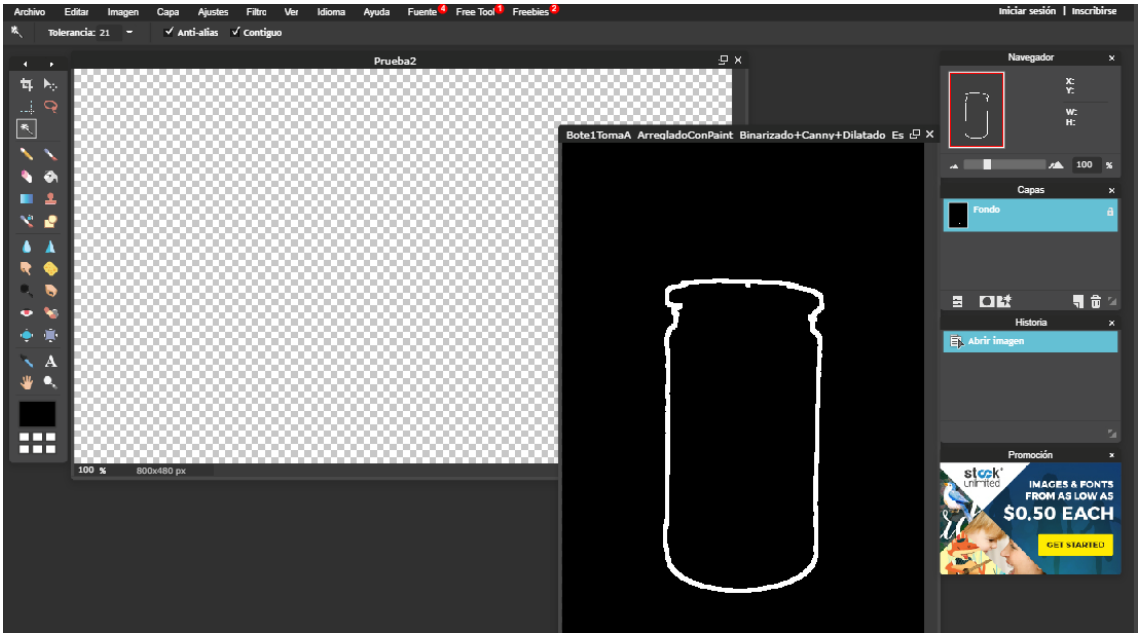

Figura B-15: A la izquierda la imagen transparente con las dimensiones deseadas y a la derecha de nuevo la imagen con la silueta del recipiente

Y esta vez, sin rellenar el fondo copiamos y pegamos la silueta en la imagen de la izquierda:

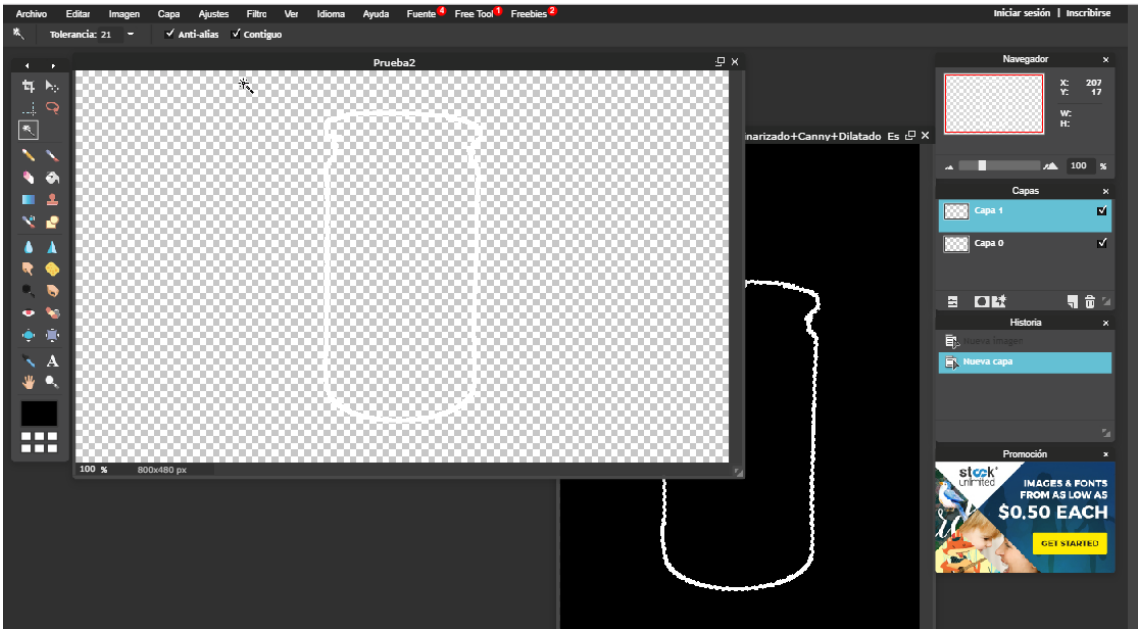

Figura B-16: Silueta del recipiente en la imagen guía con las dimensiones deseadas

Antes de guardar, con el fin de que la imagen pueda desempeñar una función de guía, es necesario rellenar la silueta de un color llamativo. Con la misma función de "relleno de color" pintamos toda la silueta de un color deseado y de esta manera creamos la imagen guía. Por último, guardamos de nuevo en formato PNG y de esta manera tenemos las dos imágenes, máscara y guía, con las dimensiones deseadas.

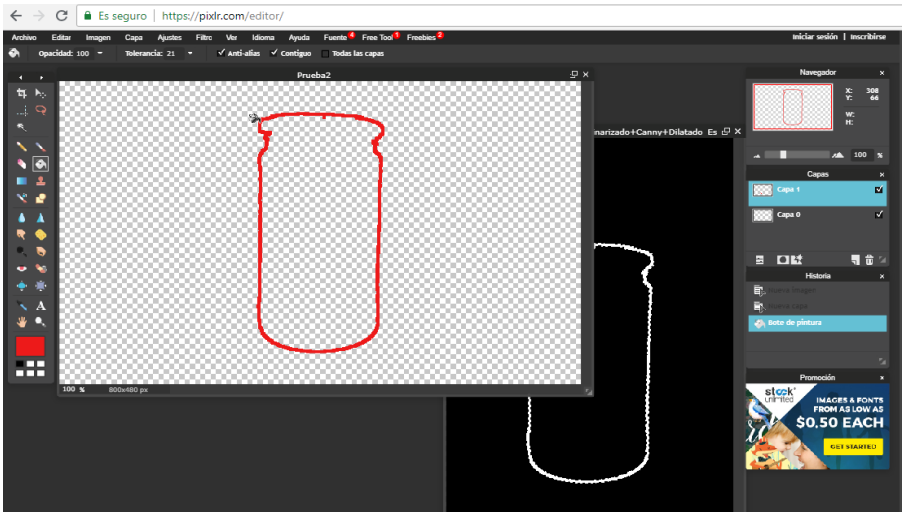

Figura B-17: Silueta del recipiente en la imagen guía con las dimensiones deseadas