Reconocimiento de una plataforma para aterrizaje de UAV mediante procesamiento de imágenes

Emilio Álvarez Piñeiro

Madrid, septiembre de 2016

Trabajo Fin de Grado en Ingeniería del Software

Departamento de Ingeniería del Software e Inteligencia Artificial

Facultad de Informática

Universidad Complutense

Curso 2015-2016

Director: Gonzalo Pajares Martinsanz

Autorización de difusión y utilización

El alumno firmante, matriculado en el Grado de Ingeniería del Software impartido por la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo de Fin de Grado, realizado durante el curso académico 2015-2016 y bajo la dirección de Gonzalo Pajares Martinsanz en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet, y garantizar su preservación y acceso a largo plazo.

Emilio Álvarez Piñeiro

Madrid, septiembre de 2016

Índice

Tabla de figuras VII
Lista de abreviaturas IX
Resumen X
Abstract XII
CAPÍTULO 1: Introducción 1
1.1 Objetivos 3
1.2 Estructura de la memoria 4
CAPÍTULO 2. Técnicas de tratamiento de imágenes digitales 5
- Modelo de color HSI 5
- Imagen binaria. 6
- Umbralización automática 6
- Scale-invariant feature transform (SIFT) 7
- Oriented FAST and Rotated BRIEF (ORB) 7
- Método basado en características de Haar 7
- Conjunto extendido de características Haar 9
CAPÍTULO 3. Análisis y diseño 11
3.1 Requisitos 11
3.1.1 Aplicación de escritorio 11
3.1.2 Aplicación web 12
3.1.3 Sistema de detección 13
3.2 Diseño 13
3.3 Casos de uso 13
3.3.1 Para la aplicación de escritorio 14
3.3.2 Para la aplicación web 15
3.4 Diagramas de actividad 17
3.4.1 Tratamiento de la imagen 17
3.4.2 Aplicación de escritorio 17
3.4.3 Aplicación web 18
CAPÍTULO 4. Implementación 21
4.1 Tecnologías para el tratamiento de imágenes 21
4.1.1 OpenCV 21

4.1.2 Python 21
4.2 Aplicación de escritorio 22
4.2.1 Qt 22
4.2.2 PyQt – Python bindings para Qt 23
4.3 Servidor web 23
4.3.1 NodeJS 23
4.3.2 Express 24
4.3.3 Passport & Github Auth API 24
4.3.4 Nodemailer 24
CAPÍTULO 5. Detección de la plataforma 25
5.1 Metodologías de análisis de imágenes - OpenCV 25
5.2 Alternativa seleccionada 25
5.3 Resultados obtenidos 27
CAPÍTULO 6. Conclusiones y trabajo futuro 31
6.1 Conclusions and future works 32
Referencias 33
Anexos 36

Tabla de figuras

Figura 1. Plataforma de aterrizaje.

Figura 2. (a) Aproximación del UAV a la plataforma; (b) UAV en vuelo con la cámara de visión en la parte inferior.

Figura 3. (a) Imagen en escala de grises; (b) Imagen binaria.

Figura 4. Máscaras de Haar para extraer las características utilizadas [11].

Figura 5. Cálculo de regiones [11].

Figura 6. Comprobación de ventanas.

Figura 7. Características Rainer y col. [12].

Figura 8. Diagrama de actividad tratamiento de imagen.

Figura 9. Diagrama de actividad de aplicación de escritorio.

Figura 10. Diagrama de actividad de aplicación web.

Figura 11. (a) Imagen utilizada para realizar la identificación; (b) Coincidencia en imagen con rotación.

Figura 12. (a) Imagen utilizada para realizar la búsqueda, tamaño 200x200 píxeles; (b) Coincidencia en imagen, tamaño 576x324 píxeles; (c) No coincidencia en misma imagen, tamaño 1152x638 píxeles.

Figura 13. (a) Imagen original en RGB; (b) Imagen en escala de intensidad.

Figura 14. Detección que incluye una región de falso positivo.

Figura 15. (a) Máscara generada; (b) Resultado de la aplicación de la máscara.

Figura 16. Imagen binaria utilizando el umbral variable de Otsu.

Figura 17. Contornos obtenidos utilizando el umbral de la Figura 15.

Figura 18. Coincidencia encontrada al final del proceso.

Figura 19. Comprobación IP forwarding activo.

Figura 20. Activar IP Forwarding.

Figura 21. Reglas de firewall.

Figura 22. (a) Menú de gestión de aplicaciones; (b) Opción para registrar nuevas aplicaciones.

Figura 23. Formulario de creación de aplicaciones.

Figura 24. Parámetros GitHub OAuth.

Figura 25. Página de autenticación.

Figura 26. (a) Autenticación de GitHub; (b) Autorización de aplicación.

Figura 27. Página principal.

Figura 28. Confirmación de transmisión al servidor.

Figura 29. Correo electrónico con archivo adjunto.

Figura 30. Ventana principal.

Figura 31. Imagen cargada en la aplicación.

Figura 32. Resultado del tratamiento de imagen.

Lista de abreviaturas

UAV: Vehículo aéreo no tripulado (Unmanned Aerial Vehicle).

USV: Vehículo submarino no tripulado (Unmanned Surface Vehicle).

AUVSI: Association for Unmanned Vehicle Systems International.

SALACOM: Sistema Autónomo para la Localización y Actuación ante Contaminantes en el Mar.

SSO: Single Sign On.

UML: Lenguaje unificado de modelado (Unified Modeling Language).

RGB: Modelo de color Red-Green-Blue.

HSV: Modelo de color Hue-Saturation-Value.

API: Application Programming Interface.

TBB: Threading Building Blocks.

IDE: Integrated Development Environment.

NPM: NodeJS Package Manager.

SMTP: Secure Message Transport Protocol.

ROI: Región de interés (Region Of Interest).

SSL: Secure Socket Layer.

Resumen

En el año 2016 se vendieron en EE.UU más de un millón de Unmanned Aerial Vehicles (UAVs, Vehículos aéreos no tripulados), casi el doble que el año anterior, país del que se dispone de información. Para el año 2020 se estima que este mercado alcance los 5.600 millones de dólares en todo el mundo, creciendo a un ritmo del 30% anual. Este crecimiento demuestra que existe un mercado en expansión con muchas y diversas oportunidades de investigación.

El rango de aplicaciones en los que se utiliza este tipo de vehículos es innumerable. Desde finales del s.XX, los UAVs han estado presentes en multitud de aplicaciones, principalmente en misiones de reconocimiento. Su principal ventaja radica en que pueden ser utilizados en situaciones de alto riesgo sin suponer una amenaza para ningún tripulante. En los últimos años, la fabricación de vehículos asequibles económicamente ha permitido que su uso se extienda a otros sectores. A día de hoy uno de los campos en los que ha adquirido gran relevancia es en agricultura, contribuyendo a la automatización y monitorización de cultivos, pero también se ha extendido su uso a diferentes sistemas, tales como seguridad, cartografía o monitorización, entre otros [1].

Es en esta situación en la que se propone el proyecto SALACOM [2], que explora la posibilidad de utilizar esta tecnología en sistemas de repuesta rápida para la detección y contención de vertidos contaminantes en entornos acuáticos con el apoyo de vehículos autónomos marinos de superficie (USV, Unmanned Surface Vehicles). En el mencionado proyecto se pretende utilizar sistemas UAVs para detectar y analizar las zonas de vertido y proveer la información respecto a la localización y las técnicas de contención adecuadas a los sistemas USV. Una vez se haya realizado el análisis de la situación del vertido, los USV trabajarían conjuntamente con los UAVs para desplegar las barreras de protección seleccionadas en la zona afectada.

Para esto, los UAVs o drones, términos similares en lo que respecta a este proyecto y que a lo largo de esta memoria se usarán indistintamente, deben ser capaces de despegar desde los USV y volver a aterrizar sobre ellos una vez realizada su labor. El proyecto que se describe en la presente memoria se centra en la fase de aterrizaje y, más concretamente, en la detección de la plataforma seleccionada como plantilla mediante técnicas de tratamiento de imágenes. Esto serviría como sistema de apoyo para guiar el dron hacia la plataforma para que pueda realizar el descenso correctamente y finalizar así su misión o bien para realizar operaciones de recarga de la batería. El dron está equipado con la correspondiente cámara de visión a bordo, con la que obtiene las imágenes, las procesa e identifica la plataforma para dirigirse hacia ella, si bien, dado que el sistema de procesamiento de imágenes no se encuentra totalmente operativo, este trabajo se centra en el desarrollo de una aplicación software independiente del sistema de visión a bordo del dron, basada en el

desarrollo de técnicas de reconocimiento de la plataforma. La plataforma a utilizar proviene de una patente [3], consistente en una figura geométrica con formas características, de muy difícil aparición en entornos de exterior. La figura pintada en negro se halla impresa sobre un panel de fondo blanco de 1m × 1m de superficie.

En este trabajo se han explorado diversas opciones disponibles para realizar la identificación de las regiones de interés. El principal objetivo es realizar la selección de una tecnología que pueda cumplir potencialmente con los criterios necesarios para llevar a cabo la tarea y seleccionar los métodos de detección adecuados para realizar la identificación de la figura contenida en la plataforma. Se ha pretendido utilizar tecnologías de fácil uso, amplío soporte y, cuando ha sido posible, de código libre. Todo ello integrado en una aplicación informática, que es la que se presenta en el presente trabajo.

Palabras clave: Aprendizaje automático, Dron, Detección de objetos, OpenCV, Python, Percepción computacional, SALACOM, Tratamiento de imagen, Visión por computador.

Abstract

In year 2016 more than 1 million UAVs were sold only in the United States, twice as many as the previous year, unique information available. By 2020, this market will have reached annual sales of 5.6 billion $ across the globe, growing a 30% each year. This growth goes to show this is an expanding market with a great amount of investigation opportunities.

The range of applications in which these vehicles are used is unimaginable. Since the last years of 20th century, UAVs have been extensively used in different applications. Their strongest point of advantage, they can be used in high risk situations without being a threat to any crew members. In the last years, low-price devices production has stimulated their use in other sectors. Nowadays they're mostly used in agriculture, especially in crop automation, but also on public security systems, cartography or monitoring [1].

This is the starting point for SALACOM Project [2], which explores the possibility to use this technology in fast response systems for detection and containment of polluting discharges at seas and lakes using Unmanned Surface Vehicles (USV). In this project, UAVs are used to detect and analyze the leaked areas and provide their location to USV systems and containment measures to be deployed. Once the analysis in done, USVs would cooperate to set up the right protection barriers in the affected areas.

For this purpose, drones or UAVs, must be able to take off from surface systems and land again on them once their task is finished. This project focuses on the landing phase, specifically, on the landing platform detection using image processing techniques. It would be used as a support system in the approximation phase, helping to perform the landing and conclude the mission. The drone is equipped with an onboard camera which obtains the images, processes them and identifies the platform to approximate. Albeit the system is not fully developed, this project focuses on building a software application independent from the onboard vision system and based in the implementation of platform automated recognition techniques. Platform design comes from a patent [3], based on geometric figure with characteristic patterns, difficult to reproduce on the exterior. The figure, painted in black, is printed on a white panel of 1m x 1m surface.

Several available methods to detect the regions of interest have been explored in this project. The main objective being choosing a set of technologies capable of meeting the necessary criteria to complete the task and select adequate detections methods to perform the detection of the figure inside the platform. Technologies with ease of use, wide support and free software (whenever possible) have been used and all have been integrated in the software application exposed in this project.

Key words: Automated learning, Computer-based vision, Computational perception, Drone, Image processing, Object detection, OpenCV, Python, SALACOM.

CAPÍTULO 1: Introducción

Los UAVs (Unmanned Aerial Vehicles, vehículos aéreos no tripulados) o drones, términos que en este trabajo se utilizarán indistintamente, están adquiriendo un gran auge en distintas aplicaciones [1]. El trabajo que se propone se sitúa en el marco del proyecto SALACOM [2], financiado por el Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, en el que participa el director del presente trabajo como investigador.

En el proyecto SALACOM se planea desarrollar un sistema de detección de vertidos contaminantes, análisis del vertido y reacción mediante el uso de vehículos no tripulados, incluyendo drones y vehículos autónomos marinos de superficie (Unmanned Surface Vehicles, USVs). Este proyecto se divide en dos subsistemas:

- Por una parte, el subsistema encargado de la localización y coordinación de los sistemas de contención en superficie mediante los USV. Es el encargado de desplazar estos vehículos hasta la zona del vertido y de controlar el despliegue de los recursos necesarios para detener el avance del mismo.
- Por otra parte, el subsistema encargado de controlar los vehículos aéreos que, en vuelo, deben localizar y delimitar la zona afectada por el vertido, realizar el análisis del contaminante desde el aire y transmitir esa información al centro de control en tierra. Estos vehículos están equipados con dispositivos de captura de imágenes y transmisión de datos. Una vez realizada la labor de detección, los drones deberán realizar el aterrizaje sobre plataformas colocadas sobre los USV utilizando sistemas de guiado tal como la localización por GPS y el análisis de imágenes.

Dentro del segundo subsistema, una de los puntos clave es el aterrizaje de los drones sobre una plataforma a bordo de los USV. En condiciones reales, en el momento de la aproximación, tanto el dron como el USV se encontrarán en un entorno acuático, de suerte que durante la fase de aterrizaje el UAV se sitúa muy cerca de la superficie del agua. Es una maniobra muy delicada por la complejidad, las condiciones cambiantes de la superficie de contacto y posibilidad de que el dron caiga al agua, quedando inutilizado. Por ello ha de realizarse con todas las garantías de éxito posible. Es aquí donde entra en juego el uso del análisis de imágenes para realizar la identificación de la plataforma, pudiendo ser utilizado como guía en el proceso de descenso y aproximación.

El proyecto de fin de grado que se describe en esta memoria tiene como objetivo el desarrollo de una aplicación informática, cuyo fundamento es el tratamiento de imágenes con el fin de reconocer e identificar la figura en la plataforma y por consiguiente la propia plataforma.

No es la versión definitiva a bordo del dron sino una versión de escritorio cuya finalidad es determinar la efectividad de los métodos de procesamiento y reconocimiento de imágenes para su posterior adaptación al sistema de procesamiento a bordo, en base a la eficiencia de los métodos utilizados.

Por tanto, en la parte inicial del proyecto que se presenta, se procede a la selección de una librería de procesamiento de imágenes que se pueda utilizar para realizar la detección de la región de interés. Para llevar a cabo esta elección se tiene en cuenta que el lenguaje de programación en la que se base la librería sea versátil, cuente con una amplia comunidad de desarrolladores, sea intuitivo y potente. También se tiene en cuenta que la librería esté orientada a aplicaciones en tiempo real, pues en la versión definitiva a bordo del dron será necesario realizar la detección en el menor tiempo posible. En la **Figura 1** se muestra la plataforma conteniendo la figura que es necesario reconocer. Dicha plataforma se encuentra patentada [3]. Como puede observarse en dicha figura, aparecen una serie de regiones características, con formas geométricas específicas y posiciones relativas entre ellas, que son las que es necesario reconocer mediante las técnicas apropiadas de tratamiento de imágenes.

***Figura 1**. Plataforma de aterrizaje.*

En la **Figura 2** se muestran dos ejemplos de un dron aproximándose a la plataforma durante el proceso de aterrizaje y en vuelo con la cámara a bordo en su parte inferior.

(a) (b)

***Figura 2**. (a) Aproximación del UAV a la plataforma; (b) UAV en vuelo con la cámara de visión en la parte inferior.*

En la segunda fase del proyecto, se procede a la implementación y prueba de varios algoritmos de identificación que permite la librería elegida. Desde un algoritmo básico basado en propiedades sencillas, pasando por varios métodos de emparejamiento de características, hasta algoritmos de aprendizaje automático.

La tercera fase consiste en el desarrollo de una aplicación de escritorio y una aplicación web con el fin de realizar la detección de una plataforma a través de los algoritmos implementados, que devuelve marcadas las regiones detectadas y que deben corresponderse con las de la plataforma a detectar.

Ambas aplicaciones reciben como entrada una imagen en la que se pretende identificar el objeto en cuestión y ejecuta los diferentes métodos de identificación implementados, marcando las regiones identificadas como coincidencias. Las imágenes marcadas son después mostradas por pantalla, en el caso de la aplicación de escritorio, o transmitidas por correo, en el caso de la aplicación web.

1.1 Objetivos

Una vez identificado el objetivo global del proyecto, tal y como se ha indicado previamente, el objetivo específico o técnico del proyecto es implementar, mediante el uso de una librería especializada de tratamiento de imágenes, un algoritmo que permita identificar una región de interés (plataforma) en una imagen. Una vez identificada la misma, se debe señalizar y mostrar el resultado de la ejecución al usuario. Todo esto utilizando tecnologías de software libre, orientadas a tiempo real y con opción de poder aplicar técnicas de paralelización automática en el futuro para reducir los tiempos de procesamiento de las imágenes.

Para facilitar el acceso a esta aplicación se desarrolla una versión de escritorio que funcione sin necesidad de conexión a Internet y una aplicación web que permita un fácil despliegue para las situaciones en las que los equipos personales no dispongan del hardware o software necesario.

El proyecto se estructura según tres grandes funcionalidades:

- Selección de una tecnología que permita implementar varios algoritmos de detección automática.
- Diseño y desarrollo de la aplicación de escritorio que permita ejecutarla en un sistema local sin necesidad de consumir recursos en red. Para realizar la ejecución en local se necesitará que las librerías estén instaladas en nativo. Es un factor a tener en cuenta que las librerías utilizadas sean multiplataforma, que permitan su instalación y ejecución en distintos entornos (Windows, Unix).
- Cómo objetivo adicional, se incluye el diseño y desarrollo de un servidor web que permita que un usuario pueda ejecutar la aplicación desde cualquier plataforma, incluyendo dispositivos móviles, sin tener que realizar la instalación de complejas y

extensas librerías. El usuario ha de poder subir una imagen al servidor, de forma que éste realice el procesamiento computacionalmente costoso y que posteriormente le envíe el resultado.

1.2 Estructura de la memoria

La presente memoria está distribuida en seis capítulos en los que se exponen las distintas etapas del proyecto.

En el primer capítulo se expone la motivación que lleva al planteamiento del proyecto y los objetivos iniciales que se pretenden alcanzar.

En el segundo capítulo se realiza un análisis de la situación actual sobre el campo referente al tratamiento de imágenes y se detallan las diversas técnicas de procesamiento analizadas durante el desarrollo del proyecto.

En el tercer capítulo se desarrolla la fase de diseño de la aplicación utilizando principios y metodologías propias de la ingeniería del software, como son la captura de requisitos, la especificación de los casos de uso y la obtención de diagramas UML.

En el cuarto capítulo se listan las tecnologías utilizadas en la implementación del proyecto, ofreciendo los argumentos en los que se ha basado su selección. En este capítulo se incluye además el proceso de detección implementado.

En el quinto capítulo se presentan los resultados obtenidos con el método propuesto.

En el sexto capítulo se exponen las conclusiones finales del proyecto y las futuras líneas de desarrollo que se consideran aplicables.

Se incluye un Anexo que contiene los manuales/guías de uso o instalación y que, pese a no guardar relación con el proceso de desarrollo de la aplicación, son necesarios para el manejo de la aplicación en su totalidad.

CAPÍTULO 2. Técnicas de tratamiento de imágenes digitales

En este capítulo se realiza una descripción de conceptos y métodos de tratamiento y reconocimiento de imágenes relacionados con el desarrollo de este proyecto.

El tratamiento de imágenes constituye el conjunto de técnicas utilizadas para almacenar, procesar, transmitir, reconocer e interpretar imágenes mediante el uso de computadoras. Estas técnicas se utilizan hoy en día en múltiples aplicaciones. Por citar sólo algunas, se puede mencionar sistemas de vigilancia, bioseguridad, detección de movimiento, entornos agrícolas, medicina, reconocimiento de patrones, o sistemas de supervisión en producciones automatizadas y por supuesto, la que se plantea en el presente trabajo.

Se puede decir que el tratamiento de imágenes es la fase previa a otra de mayor nivel conocida como visión artificial [4, 5, 6]. En efecto, dentro del tratamiento de imágenes se engloban las técnicas encaminadas a realizar las transformaciones necesarias para mejorar la calidad de la imagen, así como aquellas otras que extraen información relevante, tales como propiedades y características subyacentes en la imagen. Las propiedades así obtenidas convenientemente codificadas se proporcionan a los métodos de análisis que procesan esta información para obtener un conocimiento al más alto nivel.

Desde el punto de vista correspondiente a la aplicación cuyo desarrollo se describe en este trabajo, las técnicas de tratamiento de imágenes son aquellas que se aplican para resaltar la plataforma del resto de la imagen, separar el fondo blanco de las regiones negras, extraer propiedades de las regiones que definen las figuras geométricas de la plataforma, tales como centroides, ejes y orientación de las elipses, número de Euler, etc. Esta información se transfiere a los métodos subsiguientes, basados en técnicas de reconocimiento para determinar que las propiedades obtenidas en su conjunto constituyen la figura global, este último proceso correspondería a lo que se conoce como visión artificial, tal y como se ha indicado previamente.

A continuación se describen los métodos aplicados e integrados en la plataforma que se propone, contemplados desde las perspectivas previamente mencionadas.

- **Modelo de color HSI**

HSI es variante representación en forma de coordenadas cilíndricas del modelo de color RGB utilizado en computación [7].

Para pasar de un valor en el modelo RGB al HSI basta con aplicar las siguientes operaciones:

$$I = (R + G + B)/3$$

$$S = 1 - m/I \quad si\ I > 0, \quad donde\ m\ es\ el\ mínimo\ valor\ entre\ R, G\ y\ B$$
$$S = 0 \qquad si\ I = 0.$$

$$H = cos - 1[\ (R - ½G - ½B)/\sqrt{R^{\,2} + G^{\,2} + B^{\,2} - RG - RB - GB}\ ] \qquad si\ G \geq B$$
$$H = 360 - cos - 1[\ (R - ½G - ½B)/\sqrt{R^{\,2} + G^{\,2} + B^{\,2} - RG - RB - GB}\ ] \quad si\ B > G$$

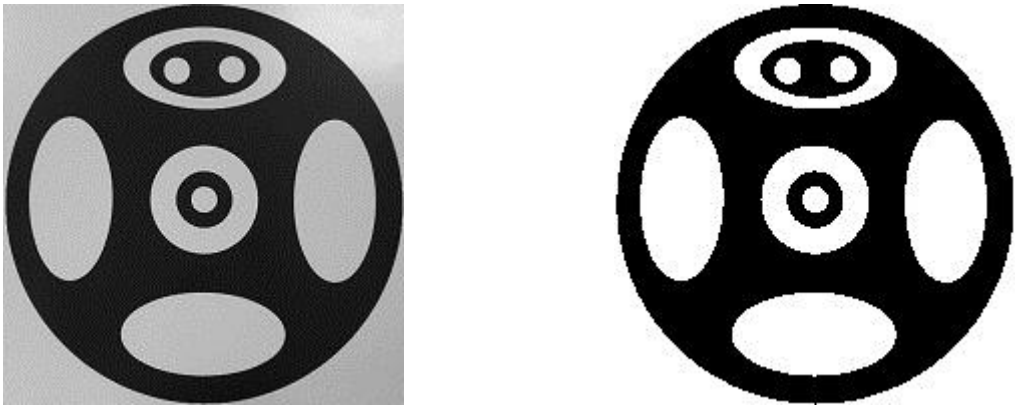
Este modelo es ampliamente utilizado en aplicaciones de visión artificial para aplicar técnicas de análisis de imágenes. Muchos de los algoritmos de tratamiento de imágenes están diseñados específicamente para utilizarse sobre imágenes en escala de grises, que se corresponde con el componente de intensidad en este modelo. En este proyecto se utiliza una variación de la componente de intensidad conocida como *luma* (Y'), que se calcula de la siguiente manera [8]:

$$Y' = 0.299\,R + 0.587\,G + 0.114\,B$$

- **Imagen binaria.**

Una imagen binaria es una representación digital de una imagen en la que cada pixel sólo puede tomar dos valores, comúnmente se representan como imágenes en blanco y negro,

Al proceso de transformar una imagen en escala de grises en una imagen binaria se conoce como binarización siendo una de las operaciones clave en el campo de procesamiento de imágenes. En el caso que nos ocupa, se utiliza para discriminar entre la figura de la plataforma sobre su propio fondo, **Figura 3**.



(a) (b)

Figura 3. *(a) Imagen en escala de grises; (b) Imagen binaria.*

- **Umbralización automática**

Para binarizar una imagen basta con escoger un valor dado dentro del rango de niveles de intensidad de la imagen, conocido como umbral, e igualar a 0 o a 1 todos los píxeles cuyo valor sea menor o mayor que el umbral dado, respectivamente. A este método se le conoce como método de umbral directo. No obstante, de cara al procesamiento automático de las imágenes se utiliza el conocido como método de umbral de Otsu [4,5] que, a través de una

serie de cálculos estadísticos utilizando la varianza de valores de los píxeles de la imagen según su histograma de intensidad, selecciona un valor óptimo del umbral que se aplicará en la binarización.

- **Scale-invariant feature transform (SIFT)**

Este algoritmo, propuesto por Lowe [9], especifica un método de extracción de características o *features* invariantes en escala, siendo ampliamente utilizado en aplicaciones de reconocimiento automático mediante el emparejamiento entre dos imágenes o vistas de un objeto.

- **Oriented FAST and Rotated BRIEF (ORB)**

ORB es un algoritmo de obtención de características, propuesto por Rublee y col. [10], cuyo propósito es ofrecer una alternativa eficiente a SIFT. Este algoritmo utiliza características basadas en FAST y BRIEF para extraer los puntos clave de una imagen de forma mucho más rápida, hasta dos órdenes de complejidad por debajo de SIFT, siendo además invariante en rotación.

- **Método basado en características de Haar**

Diseñado por Viola y col.. [11], este algoritmo está diseñado para realizar la detección automática de objetos utilizando técnicas de aprendizaje automático. Aunque la propuesta inicial del algoritmo está orientada a reconocimiento facial en imágenes, se puede extender a cualquier tipo de objeto. La principal característica de este algoritmo es su bajo coste computacional en comparación con el resto de sistemas de detección facial anteriores.

Aplica las denominadas máscaras de Haar para calcular las características de la imagen, utilizando diferentes tamaños de ventana y desplazándolas a lo largo de toda la imagen. Cada característica genera un valor numérico, la suma de los píxeles de las regiones blancas se resta a la suma de los píxeles de las regiones negras. La **Figura 4** muestra el conjunto de máscaras de Haar que se han definido con tal propósito.

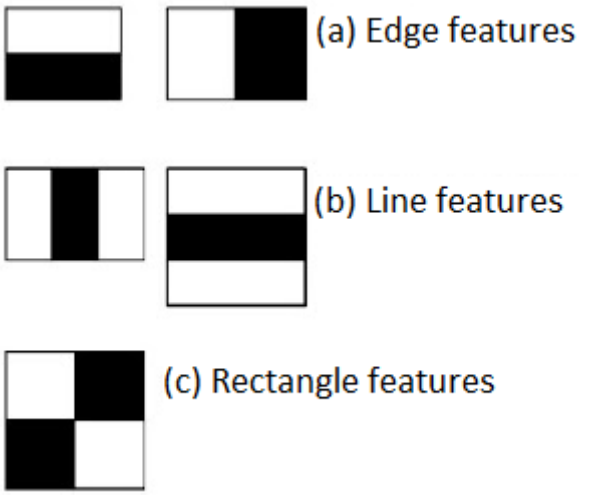


Figura 4. *Máscaras de Haar para extraer las características utilizadas [11].*

Para reducir el tiempo de cálculo de todas estas características, utilizan una tabla del sumatorio de áreas o "imagen integral" que les permite obtener el resultado de la evaluación de una característica en tiempo *constante* tras la primera iteración. Para calcular la imagen integral (ii) de un píxel (x,y), se utiliza la siguiente fórmula:

$$ii(x,y) = \sum_{x'<x,y'<y} i(x',y')$$

$$donde\ la\ función\ i\ es\ el\ valor\ de\ la\ intensidad\ del\ pixel$$

Utilizando este procedimiento de imagen integral se puede calcular el valor de regiones enteras de manera inmediata a partir del esquema mostrado en la **Figura 5**.

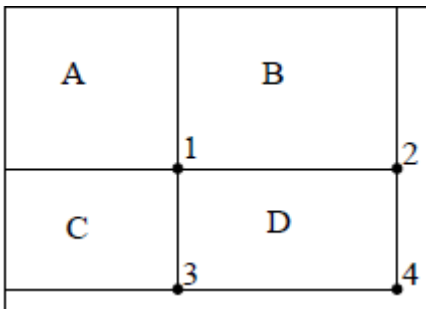


Figura 5. *Cálculo de regiones [11].*

$$D = ii(4) + ii(1) - (ii(3) + ii(2))$$

Se utiliza un sistema de aprendizaje automático para identificar y clasificar las características críticas de la imagen, obteniendo el conjunto mínimo necesario para realizar la identificación del objeto. A continuación los clasificadores se ordenan en forma de árbol, dando prioridad a los que puedan descartar con mayor rapidez una región. Para realizar la identificación del objeto se divide la imagen en ventanas, se calcula el valor de las

características y se aplica el siguiente algoritmo basado en una secuencia de clasificadores, **Figura 6**.

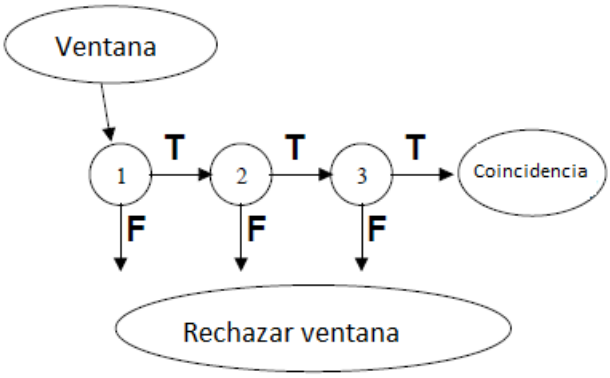


Figura 6. *Comprobación de ventanas.*

Se considera que una ventana es una coincidencia con el objeto buscado cuando todos los clasificadores coinciden.

- **Conjunto extendido de características Haar**

Diseñado por Rainer y col. [12], se basa en el algoritmo propuesto por Viola y col. Amplía el conjunto de características evaluadas introduciendo ventanas con rotación de 45°, tal y como se muestra en la **Figura 7**.

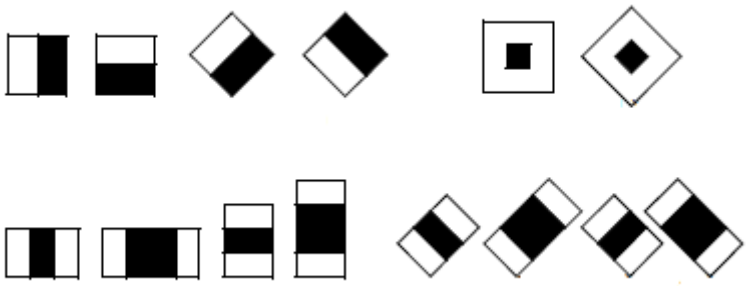


Figura 7. *Características Rainer et al. [12].*

El uso del conjunto extendido de características permite reducir el ratio de error en un 10% respecto al conjunto simple propuesto por Viola y col. [11], sin suponer un aumento en el tiempo de ejecución, ya que el cálculo se realiza en la etapa de entrenamiento.

Además, se introducen optimizaciones que se realizan en cada etapa del entrenamiento variando los parámetros del sistema de aprendizaje automático. Con estas optimizaciones se persigue seleccionar un clasificador óptimo, lo cual se consigue en lo que respecta a la efectividad del procedimiento, logrando un ratio de error de aproximadamente un 12.5%.

CAPÍTULO 3. Análisis y diseño

La primera fase del proyecto comienza con la especificación de requisitos del sistema a desarrollar junto con el análisis de las tecnologías disponibles. Se determina que es necesario diseñar un procedimiento que, tras recibir una imagen como entrada, realice las operaciones de análisis necesarias desde el punto de vista del tratamiento y reconocimiento de imágenes para identificar dentro de la imagen la zona en la que se encuentra la plataforma.

En principio se contaba con diseñar una aplicación de escritorio que permitiera cargar la imagen de manera sencilla, mostrar y manipular el resultado para facilitar el análisis. Posteriormente, debido a que la manipulación de las librerías utilizadas resulta ciertamente compleja con una carga computacional elevada, se decidió además diseñar un servidor web que permitiera al usuario subir la imagen a analizar, realizando el procesamiento de la imagen en cuestión y enviando el resultado a una dirección de correo electrónico previamente indicada. Este planteamiento, desde el punto de vista de los drones en relación a la identificación de la plataforma, puede tener cabida en tanto en cuanto es posible que desde el dron se envíe la imagen a un centro de control en tierra, donde se pueda procesar la imagen con mayor potencialidad y a su vez desde dicho centro se envíe al dron la información relativa al posicionamiento de la plataforma dentro de la imagen. No obstante, en este caso, el envío de la información no sería por correo electrónico sino mediante radioenlaces o vía Wi-Fi, que son los métodos de comunicación establecidos en el proyecto SALACOM.

3.1 Requisitos

En esta sección se enumeran los requisitos recogidos a partir de la definición inicial del proyecto desde el punto de vista del usuario final. Estos requisitos recogen la funcionalidad y condiciones que debe cumplir el sistema final y constituyen la base que se ha utilizado para realizar el diseño y la implementación de los diferentes componentes de la aplicación.

3.1.1 Aplicación de escritorio

En relación a la aplicación de escritorio se contemplan una serie de operaciones que dan lugar a las distintas funcionalidades que se describen a continuación.

1) Carga y visualización de imágenes para su posterior procesamiento.

Como requisito se exige que el usuario, a través de la aplicación, pueda seleccionar una imagen situada en su sistema de archivos, mostrándose ésta por pantalla.

Esto hace posible que el usuario pueda analizar la imagen antes de pasar al tratamiento propiamente dicho, dándole la posibilidad de confirmar que la imagen seleccionada es la adecuada, pudiendo seleccionar otra en caso contrario. Sólo se podrá cargar una imagen en cada proceso.

Se debe poder seleccionar y mostrar imágenes de tamaños y definición variables por igual. Para facilitar su manipulación, se dará la opción de aumentar/ disminuir el efecto de "zoom" sobre la imagen. Estas operaciones, que con toda probabilidad no se realizarán a bordo del dron, facilitan el análisis relativo a la efectividad de los métodos a utilizar.

2) Procesamiento de la imagen.

Una vez realizada la carga y la inspección visual de la imagen, el usuario debe poder lanzar el proceso de tratamiento, que se ejecutará recibiendo como entrada la imagen anteriormente seleccionada. Durante este proceso, se aplicarán las técnicas de detección implementadas y se identificará el área en la que se determine que se encuentra la plataforma para que el usuario pueda distinguirla.

3) Visualización del resultado.

Cuando el tratamiento de la imagen haya finalizado, la aplicación debe recibir el resultado obtenido y mostrarlo al usuario de forma sencilla y comprensible. En este caso también será necesario que el usuario pueda manipular el zoom de la imagen resultante para facilitar la inspección.

3.1.2 Aplicación web

En relación a la aplicación web se contemplan una serie de operaciones que dan lugar a las distintas funcionalidades que se describen a continuación.

1) Carga de imágenes en el servidor.

El objetivo de este requisito es hacer posible el tratamiento de la imagen en el servidor, utilizando como entrada una imagen obtenida del dispositivo del usuario. La imagen se lee en el dispositivo utilizando tecnologías de *front-end* y después se transmite al servidor, que recibe la imagen y comienza el siguiente proceso *back-end*. Sólo se podrá enviar una imagen al servidor para su procesamiento en cada petición. Debido a las limitaciones de la comunicación de red se podrá establecer un límite al tamaño en bytes de la imagen.

2) Procesamiento de la imagen.

Una vez el servidor recibe la imagen transmitida, mediante los métodos de procesamiento y reconocimiento instalados en el servidor, se debe iniciar el proceso de detección sobre ésta, aplicando las mismas técnicas de detección que se utilizan en la aplicación de escritorio. En este caso la carga de procesamiento se traslada al sistema servidor, sin consumir recursos del dispositivo de origen.

3) Transmisión del resultado al usuario.

Cuando el proceso de detección haya finalizado en el servidor, el usuario debe recibir la imagen resultado del proceso mediante un mecanismo accesible, de forma que el usuario pueda consultar y manipular el resultado tantas veces como sea necesario. Se decide que este envío sea a través de correo electrónico a una dirección previamente validada, para evitar un uso indebido de la aplicación web. Para ello será necesario implementar un sistema de autenticación o utilizar técnicas de Single Sign On (SSO).

3.1.3 Sistema de detección

El procedimiento de tratamiento y reconocimiento de imágenes debe: recibir como entrada una imagen, aplicar las técnicas implementadas y marcar como resultado la región o regiones de interés donde se determine que se encuentra la plataforma.

La imagen de entrada debe poder ser de tamaño y definición variables. Además, no será obligatorio que la imagen contenga el objeto de interés en cuestión. Se analizarán diversas propiedades de la plataforma y se aplicarán durante el procesamiento de la imagen de entrada para realizar la identificación.

3.2 Diseño

Finalizada la fase de especificación de requisitos comienza el diseño propiamente dicho. En este periodo se lleva a cabo la tarea de elaborar una guía documental de la funcionalidad a implementar utilizando diagramas basados en UML.

Debido a que en la especificación de requisitos final sólo interactúa con la aplicación un usuario, se decidió utilizar los diagramas de actividad como base del diseño. Los diagramas de actividad ofrecen una visión más detallada del flujo de ejecución de la aplicación sin entrar en cuestiones de implementación propiamente dichas.

A continuación se incluyen los diseños obtenidos en esta fase. En primer lugar se recoge la especificación de requisitos en la documentación funcional utilizando las fichas de casos de uso. En segundo lugar se proporcionan los diagramas de actividad para los diferentes subsistemas que componen el proyecto. En tercer lugar, y con posterioridad a la implementación, se elaboran los diferentes diagramas de despliegue, en los que se pueden apreciar los distintos componentes de la aplicación.

3.3 Casos de uso

Las siguientes fichas se han extraído a partir de los requisitos anteriormente expuestos y recogen de forma estructurada el flujo a alto nivel de las diferentes funcionalidades. Se han utilizado para definir la interacción entre el usuario y la aplicación.

3.3.1 Para la aplicación de escritorio

A continuación se detallan los casos de uso asociados a la aplicación de escritorio:

Nombre	Cargar imagen
Descripción	El usuario carga una imagen a la aplicación
ID	APP – 1
Actor principal	Usuario
Precondiciones	Ninguna
Flujo normal 1. El usuario pulsa en la opción de Abrir. 2. Selecciona la imagen en su sistema de ficheros y acepta. 3. La imagen se abre y se muestra en la aplicación.	
Flujo alternativo 3.a) El archivo no tiene el formato adecuado o no se encuentra. 1. No se muestra nada en el área designada. 2. Se informa al usuario del error	
Postcondición	La imagen se muestra en el área designada

Nombre	Ampliar imagen
Descripción	El usuario podrá ampliar/reducir el zoom sobre la imagen
ID	APP – 2
Actor principal	Usuario
Precondiciones	APP – 1
Flujo normal 1. El usuario desplaza la barra de zoom. 2. La dimensión de la imagen se ajusta con el desplazamiento	
Flujo alternativo No se contempla	
Postcondición	La imagen se amplía/reduce acordemente

Nombre	Procesar imagen
Descripción	El usuario podrá iniciar el procesamiento de la imagen
ID	APP – 3
Actor principal	Usuario
Precondiciones	APP – 1

Flujo normal

1. El usuario pulsa la opción Procesar.
2. La aplicación ejecuta el algoritmo de detección de la plataforma sobre la imagen cargada
3. Terminado el algoritmo de detección, muestra la imagen con el resultado en el área designada.

Flujo alternativo

3.a) Error en el proceso de detección.

1. Se informa del error al usuario

Postcondición	La imagen resultado se muestra en el área designada

3.3.2 Para la aplicación web

Para la aplicación web se identifican los siguientes casos de uso:

Nombre	Login
Descripción	El usuario acceder a la aplicación con credenciales de acceso
ID	WEB – 1
Actor principal	Usuario
Precondiciones	Ninguna

Flujo normal

1. El usuario introduce usuario/contraseña en un formulario de login.
2. Valida las credenciales.
3. El usuario es redirigido hasta la página principal de la aplicación (CU: WEB - 2).

Flujo alternativo

2.a) Las credenciales introducidas no son correctas.

1. Se informa al usuario del error.
2. Se redirige al usuario al formulario de login

Postcondición	El usuario está ahora autorizado

Nombre	Cargar imagen al front-end
Descripción	El usuario carga una imagen a la aplicación de front-end
ID	WEB – 2
Actor principal	Usuario
Precondiciones	WEB – 1

Flujo normal

1. El usuario selecciona la opción de abrir imagen.
2. Selecciona la imagen en su sistema de ficheros y acepta.
3. La imagen se abre y carga en el formulario.

Flujo alternativo

No se contempla

Postcondición	La imagen se carga en el front-end

Nombre	Enviar para procesar
Descripción	El usuario envía la imagen al back-end para su procesamiento
ID	WEB – 3
Actor principal	Usuario
Precondiciones	WEB – 2

Flujo normal

1. El usuario pulsa la opción Enviar.
2. El front-end envía la petición back-end.
3. Cuando el back-end recibe la petición, ejecuta el algoritmo de detección, guarda la imagen resultado y la envía por correo electrónico al usuario

Flujo alternativo

3.a) El archivo no tiene el formato adecuado o se encuentra.

1. No se muestra nada en el área designada.
2. Se informa al usuario del error

Postcondición	La imagen se carga en el front-end

3.4 Diagramas de actividad

Los siguientes diagramas muestran el flujo de trabajo de los diferentes componentes de la aplicación de forma gráfica y fácilmente comprensible.

3.4.1 Tratamiento de la imagen

En la Figura 8 se muestra la funcionalidad clave del proyecto relativa al tratamiento de la imagen. En su diseño se ha pretendido que sea reutilizable tanto para el servidor web como para la aplicación de escritorio. En este diagrama se han incluido: un parámetro de entrada, la ruta en la que se encuentra la imagen a procesar, y un parámetro de salida, la ruta donde se guardará la imagen marcada.

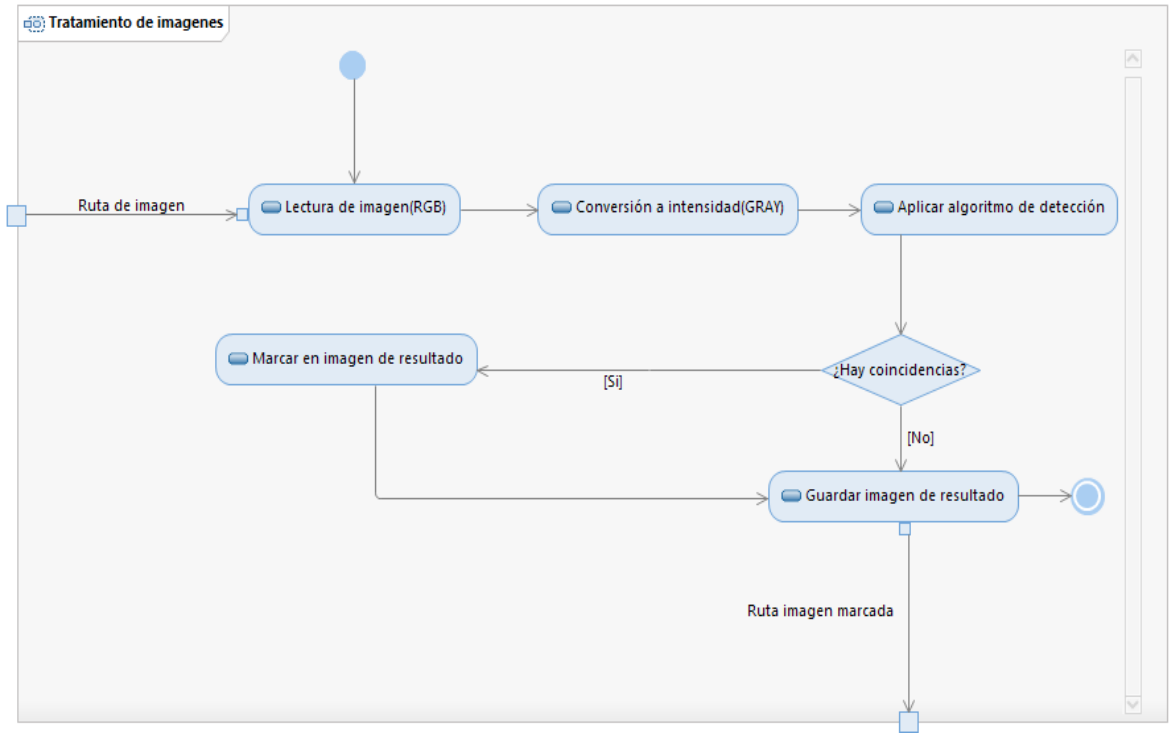


Figura 8. *Diagrama de actividad tratamiento de imagen.*

Éste es el flujo que seguirá el programa para el tratamiento de la imagen. Recibe como parámetro la ruta en el sistema de archivos donde se encuentra el archivo. El primer paso consiste en leer el archivo en RGB, creando una copia en memoria. El segundo paso es crear otra copia en escala de intensidad (GRAY), que es sobre la que se aplicarán todas las técnicas de análisis. Si se detecta alguna coincidencia en la etapa de análisis, las zonas detectadas se marcarán en la imagen RGB almacenada en memoria. Para finalizar, la imagen marcada se guardará en un fichero nuevo cuya ruta será el valor de salida de la actividad.

3.4.2 Aplicación de escritorio

En la **Figura 9** se puede apreciar que en la aplicación de escritorio se incluye la actividad de la **Figura 8.** El usuario deberá haber cargado una imagen para ejecutar el algoritmo de

detección. Una vez cargada, la imagen se utiliza como parámetro de entrada para la actividad de tratamiento. Cuando el proceso de detección haya terminado, la aplicación recibirá la ruta de la imagen marcada como parámetro de salida de la actividad y la mostrará en el área designada. El usuario podrá cargar otra imagen distinta y realizar otra vez la actividad completa sin tener que cerrar la aplicación.

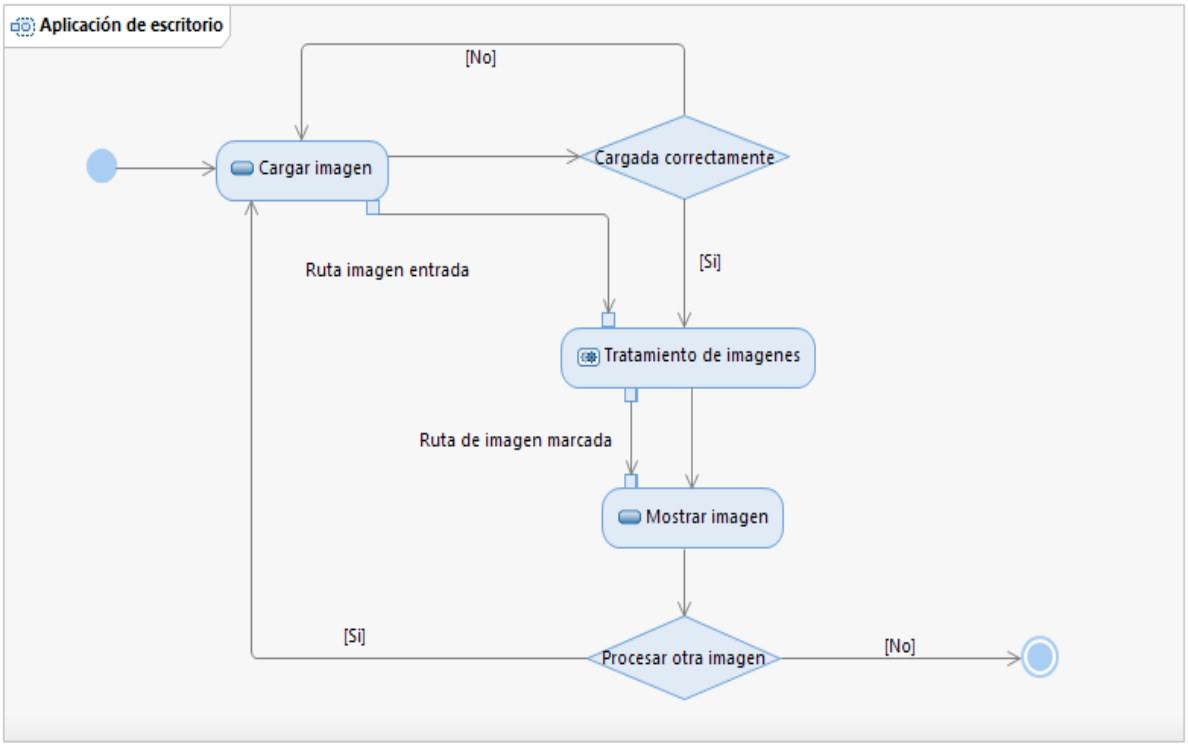


Figura 9. *Diagrama de actividad de aplicación de escritorio.*

3.4.3 Aplicación web

En la **Figura 10** se muestra el diagrama de actividad relativa a la aplicación web. El flujo de la aplicación web comienza con autenticación del usuario utilizando credenciales autorizadas. Para aumentar la seguridad de la aplicación se ha optado por implementar un sistema de SSO utilizando la API de autenticación de GitHub. Este sistema es fácilmente extensible a otros proveedores de autenticación y asegura que la dirección de correo del usuario ha sido previamente validada.

Como se puede apreciar también reutiliza la actividad de tratamiento de imagen. En este caso la carga de la imagen se llevará a cabo utilizando tecnologías web de tipo front-end (HTML, JavaScript), se enviará al servidor para su procesamiento a través de una petición POST y el procesamiento de la imagen se realizará en el lado del servidor. Una vez finalizado el proceso se enviará un correo electrónico con la imagen marcada o, en caso de que haya ocurrido un error en la petición, informando al usuario.

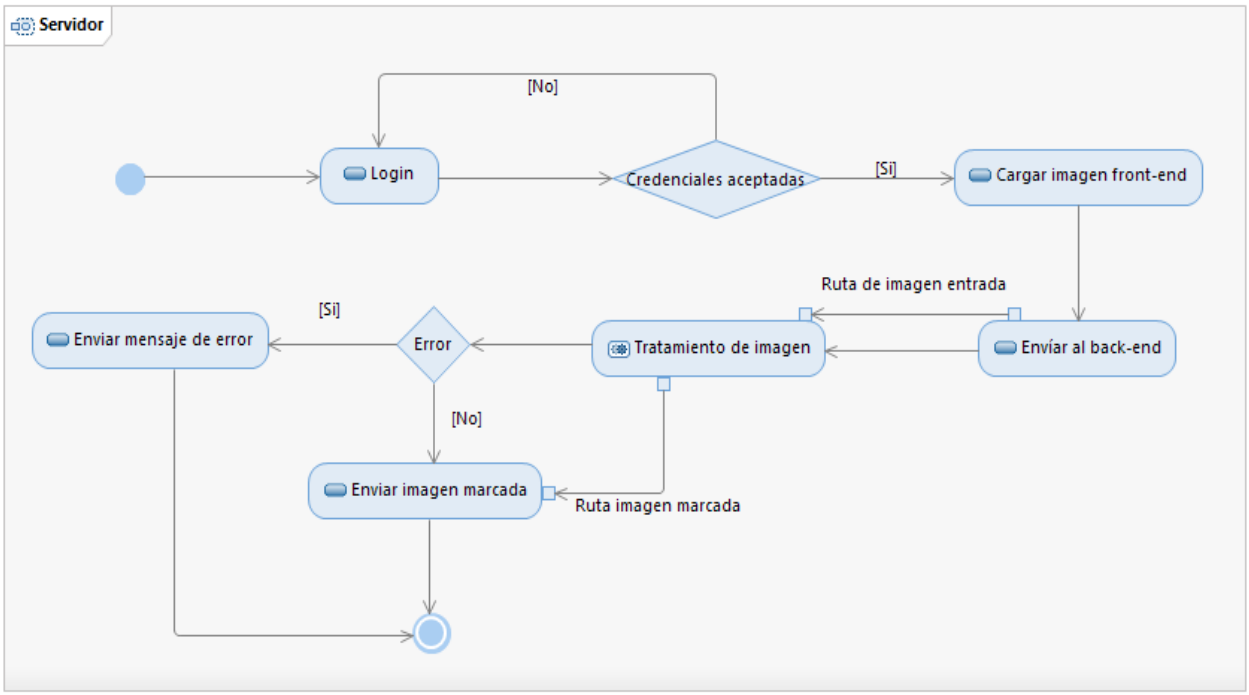


***Figura 10**. Diagrama de actividad de aplicación web.*

CAPÍTULO 4. Implementación

En este capítulo se exponen las tecnologías utilizadas, el resultado obtenido y los diagramas de despliegue elaborados a posteriori. Todas las librerías seleccionadas para llevar a cabo la implementación son de libre disponibilidad para fines académicos/científicos y cuentan con una comunidad oficial de desarrolladores de respaldo.

4.1 Tecnologías para el tratamiento de imágenes

Para implementar el algoritmo de detección de la plataforma, era necesario disponer de una librería que permitiera manipular imágenes y con la que se pudieran aplicar las diferentes operaciones. Para ello se realizó una inspección de las librerías gráficas disponibles que mejor se ajustaran a nuestras necesidades y a los requisitos planteados, es decir:

- Orientación a tiempo real, asegurando que las operaciones disponibles están completamente optimizadas y que se pueden utilizar en sistemas donde el tiempo de respuesta es mínimo.
- Basadas en lenguajes de programación conocidos o de fácil aprendizaje para poder realizar un análisis certero de la funcionalidad ofrecida por la librería.

4.1.1 OpenCV

Al final de este proceso de selección la librería elegida fue OpenCV [13]. OpenCV es una librería de visión artificial bajo una licencia BSD, basada en C/C++ y orientada a aplicaciones en tiempo real. Soporta entornos Linux, Windows, Android e IOS. Además se puede utilizar con C++, Python y Java(a partir de la versión 3.0).

Al estar orientada a aplicaciones de tiempo real, ofrece un amplio nivel de paralelización configurable: TBB, OpenMP, OpenCL y más. El proceso de compilación de la librería permite indicar que tecnología se quiere utilizar.

Uno de los puntos fuertes de esta librería es que cuenta con varios algoritmos de aprendizaje automático ya optimizados. Entre ellos se encuentra una implementación del algoritmo de detección de Viola y col. [11] y la modificación propuesta por Rainer y col. [12], ambos basados en características de Haar.

4.1.2 Python

Esta librería ofrece, en la versión utilizada en este proyecto (v2.4.13), soporte nativo para C/C++ y Python [14].

Se ha optado por utilizar Python como lenguaje para realizar el desarrollo, aunque comparativamente sea el lenguaje con menor rendimiento, debido a las siguientes razones:

1) Es un lenguaje sencillo y atractivo, con una curva de aprendizaje mínima. Una de las causas de su "lentitud" en comparación con C++ se debe a que abstrae características de los lenguajes de bajo nivel, haciendo su gestión transparente para el programador. Un ejemplo claro es el modelo de gestión de memoria, realizado en su mayor parte por el intérprete. Este detalle facilita enormemente la escritura de código a cambio de la pérdida de rendimiento.
2) Python cuenta con una de las comunidades más amplias y activas del sector.
3) Con más de 80.000 librerías desarrolladas por terceros, disponibles a través del gestor de paquetes oficial, Python Package Index [15]. Incluye módulos sobre comunicación, gestión de redes, manejo de bases de datos, multimedia, interfaces gráficas, aplicaciones científicas entre otras muchas.
4) OpenCV ofrece soporte nativo para Python a través de un interfaz. La funcionalidad de la librería que se encarga del tratamiento de las imágenes está escrita, optimizada y compilada en C++. Esta funcionalidad se hace accesible desde código escrito en otros lenguajes. Con ello se consigue que la carga computacionalmente compleja se pueda ejecutar sobre el código en C++ desde Python sin disminuir el rendimiento.

4.2 Aplicación de escritorio

Para realizar el desarrollo de la aplicación de escritorio se utiliza una librería gráfica con la que diseñar los diferentes componentes gráficos. Debido a que el lenguaje seleccionado para el desarrollo del algoritmo de la aplicación es Python, en el proceso de selección se tuvo en cuenta que se ofreciera soporte para este lenguaje. La librería escogida fue Qt 4.8 [16].

4.2.1 Qt

Qt es una biblioteca multiplataforma de código abierto utilizada para diseñar interfaces de usuario ampliamente reconocida [17]. Al igual que OpenCV, soporta entornos de escritorio, embebidos y móviles.

Cuenta con un IDE, QtDesigner, con el que se puede realizar el desarrollo de la interfaz gráficamente, sin tener que compilar o ejecutar código. Aunque la biblioteca está basada en C++, los componentes diseñados utilizando el entorno de desarrollo se almacenan en un código intermedio. Este código es posteriormente compilado en el lenguaje que seleccione el desarrollador. Existen varias librerías implementadas por terceros y soportadas por Qt que permiten realizar la compilación del código intermedio sobre diversos lenguajes [18]. Para generar código Python a partir de la interfaz diseñada, se ha utilizado PyQt [19].

4.2.2 PyQt – Python bindings para Qt

PyQt es un conjunto de *bindings* para el entorno Qt desarrollado por Riverbank Computing Limited [20]. Este módulo redefine, utilizando Python, todos los elementos de Qt. Contiene más de 1.000 clases re-implementadas, utilizadas para dar soporte a todas las funcionalidades que ofrece para C++.

Utilizando el ejecutable pyuic4.exe incluido dentro de esta librería, se puede indicar como entrada un fichero .UI obtenido con el IDE QtDesigner y se obtendrá como salida código Python.

4.3 Servidor web

A continuación se detallan las tecnologías utilizadas en el desarrollo de la aplicación web. En este apartado se entiende por tecnología elementos tales como librerías, entornos o módulos. No se detallan herramientas como JavaScript, HTML & CSS por considerarse inherentes al desarrollo web, aunque también han sido utilizados.

La pieza clave en este apartado es NodeJS [21], que provee las herramientas necesarias para construir el servidor web.

4.3.1 NodeJS

NodeJS es un *framework* basado en V8 [22], un motor ejecución de JavaScript diseñado y utilizado por Google Chrome, de código abierto y multiplataforma. Está enfocado a la creación de servidores web altamente escalables y cuenta con un gran número de módulos implementados por terceros. Esta herramienta permite ejecutar código JavaScript en el lado servidor, en contraposición al uso tradicional que se le da en el *front-end*.

Las razones por las que se ha escogido NodeJS sobre otros *frameworks* web tradicionales como PHP o Django son:

- Sigue un modelo I/O asíncrono. Esto permite que el sistema destine menos recursos a la gestión de procesos y pueda atender más peticiones/s [23].
- Históricamente, PHP y JavaScript han sido lenguajes interpretados. Mientras que esta característica añade portabilidad a las aplicaciones desarrolladas, tiene repercusiones negativas en el rendimiento respecto a lenguajes compilados. Sin embargo, el motor V8 implementa una estrategia de compilación *Just-In-Time,* permitiendo que las regiones de código en las que hay un nivel de tránsito alto sean compiladas para obtener una mejora de rendimiento.
- NPM, el gestor de paquetes de NodeJS, cuenta con un gran número de módulos sobre *webapps*, enrutado, aplicaciones móviles u otras, implementados por terceros y de fácil instalación. Además, NPM también puede ser utilizado como gestor de dependencias para nuestro propio proyecto, simplificando el proceso de despliegue de la aplicación.

- Para utilizar NodeJS no es necesario instalar ni configurar ningún servicio en el sistema. Basta con instalar el *framework* y éste se encargará del manejo de los procesos cuando ejecutemos nuestra aplicación servidor.

4.3.2 Express

Este módulo de NPM es prácticamente un *framework* dentro del propio NodeJS que simplifica el proceso de implementar un servidor web. Implementar un servidor HTTPS utilizando la funcionalidad básica de NodeJS exige que el desarrollador implemente la lógica de enrutado, la gestión de cookies y filtrado de las peticiones por POST/GET/UPDATE/DELETE/PUT cuando sea necesario.

Express [24] ya proporciona toda esta lógica sin necesidad de que el desarrollador la implemente. Tan sólo hay que asociar la *app*, cómo se denomina en este módulo al objeto JavaScript que engloba toda la funcionalidad del servidor, a un puerto de red abierto y configurarla como sea necesario.

4.3.3 Passport & Github OAuth API

El módulo Passport [25] es un *middleware* que implementa de forma automática estrategias de autenticación y comunicación con proveedores como Facebook, Google o Twitter. Se puede utilizar para gestionar un sistema de SSO y para acceder al resto de APIs que ofrecen estos proveedores.

Para cada proveedor hay configurada una estrategia que determina el flujo de la comunicación entre el servidor demandante y el servidor del proveedor para que la autenticación del usuario se realice correctamente y el demandante obtenga un *token* de acceso para el usuario.

En este proyecto, se ha implementado una estrategia de SSO utilizando la API de Github [26]. A través de esta estrategia, se obtiene la dirección de correo, que ya ha sido validada previamente por el gestor de usuarios del mismo proveedor, a la que se enviarán las notificaciones.

4.3.4 Nodemailer

Nodemailer [27] es un módulo de NPM que simplifica el proceso de envío automático de correos electrónicos. Permite enviar mensajes utilizando el protocolo SMTP, incluir archivos adjuntos y da acceso a todos los campos del correo.

Utilizando éste módulo se envía el resultado de la ejecución en el servidor al usuario.

CAPÍTULO 5. Detección de la plataforma

Para la implementación del algoritmo de detección de la plataforma se han explorado diversas metodologías o técnicas que la librería OpenCV nos permite implementar.

5.1 Metodologías de análisis de imágenes - OpenCV

Existen tres alternativas para realizar la detección de una región de interés en una imagen utilizando esta librería:

1) Utilizar transformaciones primitivas sobre la imagen: se realiza un análisis de las propiedades gráficas de la plataforma y se utiliza el resultado para detectar la zona de interés. Esto permite escoger y priorizar las características relevantes para considerar una región de la imagen como "coincidencia".
2) Utilizar *feature-matching* o emparejamiento de características: SIFT & ORB. Estos dos algoritmos se pueden utilizar en aplicaciones de reconocimiento de objetos aunque con aproximaciones diferentes. OpenCV cuenta con una implementación que recibe como entrada dos imágenes: el objeto de interés y la imagen donde se debe efectuar la búsqueda. Cómo se detalla en el capítulo 2, ambos comienzan analizando las características de la imagen indicada, identificando los "puntos clave". Después escanean la imagen destino intentando encontrar coincidencias. Cuando se alcanza el número mínimo de coincidencias se considera que se ha encontrado el objeto. La diferencia entre ellos radica en los descriptores que utilizan para identificar los puntos clave.
3) Utilizar el algoritmo de clasificador en cascada basado en características de Haar [28]. Esta funcionalidad de OpenCV corresponde a la implementación del algoritmo propuesto por Viola y col. [11] y la posterior ampliación de Rainer y col. [12]. Aunque en estos artículos académicos se realiza la comparativa con otros algoritmos de detección automática de rostros, también se pueden utilizar para el reconocimiento de objetos específicos. Es un algoritmo de aprendizaje automático para que el cual hay que realizar una etapa de entrenamiento, de la que se obtiene el clasificador que después se puede reutilizar tantas veces como sea necesario.

5.2 Alternativa seleccionada

Después de implementar y comparar todas las opciones, se optó por utilizar una combinación de clasificador de Haar y el análisis de las propiedades, atendiendo a las siguientes razones:

a) SIFT es capaz de realizar la identificación del objeto aunque la escala sea diferente respecto a la de la plantilla y el objeto haya sido rotado debido a que

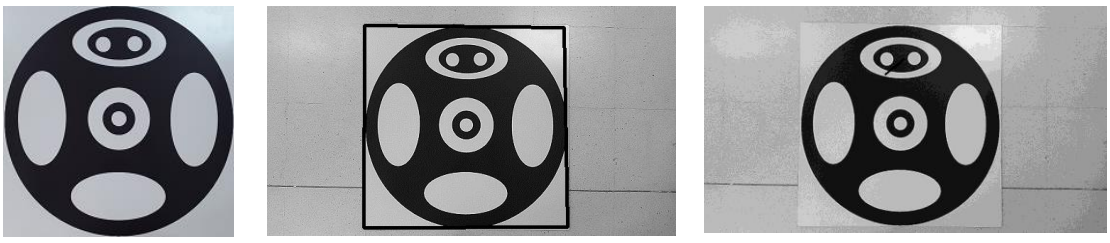
utiliza descriptores invariantes en escala y rotación. La Figura 11 muestra en (a) la imagen de la plataforma original y en (b) la imagen reconocida y enmarcada a pesar de encontrarse girada.

(a) (b)

***Figura 11**. (a) Imagen utilizada para realizar la identificación; (b) Coincidencia en imagen con rotación*

Cuando lo que varía de una imagen a otra es la iluminación del objeto, el algoritmo es incapaz de encontrar coincidencias de descriptores entre las dos imágenes [29]. Además resulta ser el algoritmo con mayor tiempo de ejecución.

b) ORB ha sido diseñado específicamente para ser más rápido que SIFT. Utiliza descriptores FAST, de cálculo muy rápido pero que no son invariantes en escala. Si la escala del objeto a identificar varía significativamente entre una imagen y otra, el algoritmo no será capaz de realizar la identificación [10]. En la **Figura 12** se muestra la imagen de la plataforma, junto a dos escalas de la misma, observándose que la variación de la escala permite reconocer la imagen (b) pero no la (c).



(a) (b) (c)

***Figura 12**. (a) Imagen utilizada para realizar la búsqueda, tamaño 200x200 píxeles; (b) Coincidencia en imagen, tamaño 576x324 píxeles; (c) No coincidencia en misma imagen, tamaño 1152x638 píxeles.*

c) Aplicar una gran cantidad de operaciones primitivas sobre una imagen como la binarización con umbral adaptativo, la extracción de bordes y regiones tiene un alto coste computacional comparado con otros métodos. Si no se implementa un mecanismo para descartar regiones rápidamente, se estarán aplicando operaciones sobre regiones que nunca contendrán una coincidencia. Para solucionar esta problemática se utiliza el clasificador Haar.

d) El clasificador Haar es capaz de detectar objetos independientemente del tamaño. Las características generadas por el entrenador son invariantes en escala y el algoritmo itera evaluando ventanas de proporción variable respecto a la imagen, desde 1:1 hasta el tamaño deseado, disminuyendo con cada iteración. Además, es capaz de reconocer objetos con diferentes grados de inclinación/rotación, siempre y cuando estos ejemplos se incluyan en la fase de entrenamiento. Lo mismo ocurre con la variación en la iluminación. El factor más relevante es que este algoritmo es el más rápido. Está especialmente diseñado para descartar regiones muy amplias de la imagen en las primeras etapas de evaluación, reduciendo considerablemente el coste computacional y manteniendo un ratio mínimo de falsos negativos. A cambio, el ratio de falsos positivos es considerable, alrededor de un 20-30% utilizando la mejora de Rainer y col. [12] con un clasificador correctamente entrenado.

5.3 Resultados obtenidos

El algoritmo comienza creando una copia de la imagen original en RGB y transformándola para obtener la representación de la intensidad en escala de grises, utilizando el modelo Y'(*luma*) citado anteriormente. Esta imagen transformada será utilizada en los pasos posteriores, **Figura 13.**

(a) (b)

Figura 13*. (a)Imagen original en RGB; (b) Imagen en escala de intensidad.*

Sobre la imagen de intensidad y utilizando un clasificador previamente entrenado, se aplica el algoritmo de detección automática propuesto por Viola y col. [11], implementado en el método **CascadeClassifier::detectMultiScale** de la librería OpenCV. Este método devuelve las ventanas donde se ha obtenido una coincidencia para el objeto, tal y como se muestra en la **Figura 14.**

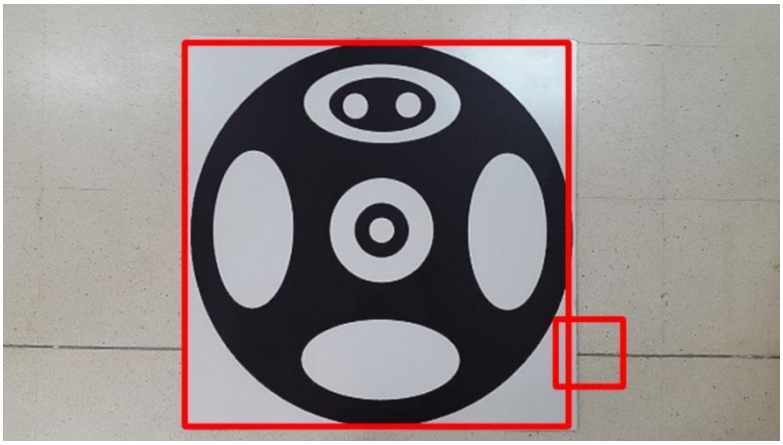


***Figura 14**. Detección que incluye una región de falso positivo.*

El resultado de este algoritmo puede incluir regiones de falsos positivos. Dependiendo de la eficacia del clasificador entrenado, el número de falsos positivos obtenidos en esta etapa será menor y las regiones que se procesen manualmente serán a su vez menores, reduciendo el tiempo de ejecución necesario.

A continuación se aplica una máscara sobre la imagen de intensidad original para obtener las regiones de interés (ROI), utilizando las ventanas obtenidas en el paso anterior. Los resultados se muestran en la **Figura 15.**

(a) (b)

***Figura 15**. (a) Máscara generada; (b) Resultado de la aplicación de la máscara.*

De esta forma se consigue minimizar el área dónde se va a llevar a cabo el análisis manual, con mayor coste computacional. Con las regiones de interés delimitadas, utilizando un umbral variable de Otsu, se obtiene una imagen binaria, **Figura 16.**

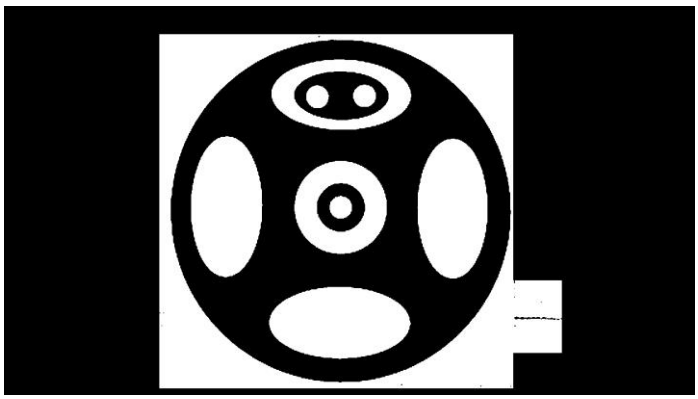


***Figura 16**. Imagen binaria utilizando el umbral variable de **Otsu**.*

La imagen binarizada se utiliza posteriormente para la detección y extracción de contornos, que delimitan las regiones que definen la plataforma, **Figura 17**.

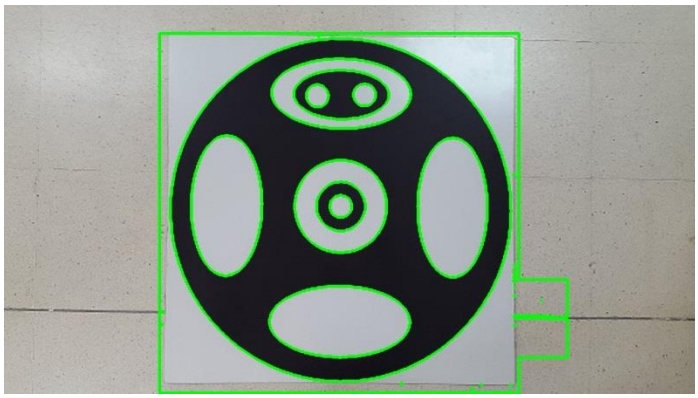


***Figura 17**. Contornos obtenidos utilizando el umbral de la Figura 15.*

Para cada contorno, se calculan propiedades tales como el número de Euler, solidez e intensidad media y se comparan con los valores prefijados. Si son coincidentes, el contorno se marca sobre la imagen inicial en RGB. Una vez finalizado el análisis de las propiedades, se muestra la imagen inicial con las coincidencias marcadas, como se muestra en la **Figura 18.**

Figura 18. Coincidencia encontrada al final del proceso.

CAPÍTULO 6. Conclusiones y trabajo futuro

El objetivo principal de este proyecto era desarrollar una aplicación que, aplicando técnicas de procesamiento gráfico sobre una imagen, pudiera realizar la detección de una plataforma de aterrizaje de un dron.

La aplicación aportada cumple el propósito principal del proyecto de forma satisfactoria, permitiendo cargar imágenes y procesarlas con el fin de extraer la región de interés (plataforma). Como funcionalidad adicional se ha realizado el desarrollo del servidor web, que permite al usuario ejecutar el algoritmo en un sistema externo utilizando un terminal con acceso a Internet y recibir los resultados de manera conveniente, sin tener que realizar la instalación o configuración de ningún componente software.

Para llevar a cabo la implementación del proyecto se ha realizado la especificación de requisitos en base a los objetivos de la aplicación, el análisis y diseño utilizando herramientas de diseño software.

Además, se han explorado diferentes técnicas de detección soportadas por la librería gráfica seleccionada. Entre éstas se incluyen algoritmos de detección, comparación de características o *features* y sistemas de aprendizaje automático.

Respecto a futuras líneas de trabajo observables una vez alcanzados los objetivos iniciales, cabe mencionar:

e) Reutilizando la técnicas de aprendizaje automático, realizar el entrenamiento de un clasificador óptimo, para el que se estima que son necesarias entre 20.000 – 30.000 imágenes de ejemplos positivos/negativos [30].
f) Aplicar el algoritmo de detección sobre una secuencia de imágenes en forma de vídeo, que permita no sólo realizar la detección de la plataforma sobre una imagen estática sino un conjunto de ellas, de forma secuencial, permitiendo realizar un seguimiento del objeto.
g) Desarrollar una aplicación aprovechando el soporte de OpenCV y Qt para los sistemas operativos móviles más utilizados, Android y IOS, que permita utilizar el hardware de captación de imágenes de los dispositivos móviles para aplicar el algoritmo de detección, ya sea sobre una imagen estática o un *stream* de vídeo.
h) Analizar diversas propiedades gráficas de la plataforma como los centroides, orientación de las componentes, color, etc. y utilizar el resultado para mejorar la tasa de acierto del algoritmo.

6.1 Conclusions and future works

The main objective of this project was to develop a software application which, using graphic processing techniques on an image, could detect the landing platform of a drone.

The application supplied in this project meets the main objective satisfactorily, allowing the user to load images and process them in order to extract the region of interest, in this case the platform. As additional functionality, a web server has been implemented, which allows the user to run the detection program on a remote server using a terminal with Internet access and receive the result in a convenient manner, without having to install or configure any software component.

To implement this project, definition of requirements, analysis and the design have been carried out using the objectives set for the application and software design tools.

Also, several detection methods supported by the chosen graphic library have been explored. Among these detection algorithms, feature comparison and automated learning systems.

Regarding the future lines of work to pursue initial objectives have been met, it's worth mentioning:

a) Reusing the automated learning techniques, perform the training of an optimized classifier, for which more than 20.000 – 30.000 images are needed between positive/negative examples [30].
b) Apply the detection algorithm on a sequence of images as a video stream, which allows not only performing the detection in a static image but on a sequence of them, tracking the object.
c) Develop an application using OpenCV and Qt libraries' support for most commonly used mobile OS, Android and IOS, which allows the user to capture images using the device's camera and processing to perform the detection, be it on a static image or a video stream.
d) Analyze platform design's graphic properties like centroids, components orientation or mean color. And use the result to improve the algorithm's rate of successful detection.

Referencias

[1] Pajares, G. Overview and current status of remote sensing applications based on unmanned aerial vehicles (UAVs). Photogrammetric Engineering & Remote Sensing 81 (4), 281-329, 2015.

[2] SALACOM (DPI2013-46665-C1). Sistema Autónomo para la Localización y Actuación ante Contaminantes en el Mar (SALACOM). Ministerio de Economía y Competitividad, 2013.

[3] Cruz, J.M, Sánchez, D., Pajares, G. Sistema de aproximación a una plataforma de un vehículo no tripulado mediante análisis visual. Patente nº 201001592, 2013.

[4] Pajares, G. y Cruz, J.M. Visión por computador: imágenes digitales y aplicaciones. RA-MA, Madrid, 2007.

[5] Pajares, G. y Cruz, J.M. Ejercicios resueltos de Visión por Computador. RA-MA, Madrid, 2007.

[6] Alegre, E. Pajares, G. y Escalera, A. (Eds.) Conceptos y métodos en Visión por Computador. CEA, Madrid, 2016. Disponible en internet: http://intranet.ceautomatica.es/conceptos-y-metodos-en-vision-por-computador. Accedido Agosto 2016.

[7] Gevers, T. Gijsenij, A. van de Weijer, J y Geusebroek, J. Color in Computer Vision: Fundamentals and Applications. John Wiley & Sons, 2012.

[8] Miscellaneous Image Transformations, OpenCV 2.4.13 Official Documentation. http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html. Accedido Agosto 2016.

[9] G. Lowe, David. Distinctive Image Features from Scale-Invariant Keypoints. Publicado en *International Journal of Computer*, Noviembre 2004, Volumen 60, 2º edición, pp 91-110. http://link.springer.com/article/10.1023%2FB%3AVISI.0000029664.99615.94. Accedido Agosto 2016.

[10] Rublee, Ethan; Rabaud, Vincent; Konolige, Kurt y Bradsky, Gary. ORB: an efficient alternative to SIFT or SURF. Publicado en *International Conference on Computer Vision*, Noviembre 2011, pp 2564-2571.

[11] Viola, Paul y J. Jones, Michael. Robust Real-Time Face Detection. Publicado en *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference*, Diciembre 2001, Volumen 1, pp 511-518.

http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf. Accedido Agosto 2016.

[12] Lienhart, Rainer y Maydt, Jochen. An Extended Set of Haar-like Features for Rapid Object Detection. Publicado en *Image Processing. 2002. Proceedings. 2002 International Conference*, Septiembre 2002, Volumen 1, pp 900-903. http://www.lienhart.de/Prof. Dr. Rainer Lienhart/Source Code files/ICIP2002.pdf. Accedido Agosto 2016.

[13] OpenCV. Open Source Computer Vision. http://opencv.org/. Accedido Agosto 2016.

[14] Python. https://www.python.org/. Accedido Agosto 2016.

[15] Python Package Index. https://pypi.python.org/pypi. Accedido Agosto 2016.

[16] Qt Wiki. http://wiki.qt.io/About_Qt. Accedido Agosto 2016.

[17] Listado de aplicaciones que hacen uso de la librería gráfica Qt, Wikipedia. https://es.wikipedia.org/wiki/Qt_(biblioteca)#Organizaciones_que_utilizan_Qt. Accedido Agosto 2016.

[18] Qt Bindings. http://wiki.qt.io/Category:LanguageBindings. Accedido Agosto 2016.

[19] PyQt. Riverbank. https://riverbankcomputing.com/software/pyqt/intro. Accedido Agosto 2016.

[20] Riverbank Computing Limited. https://riverbankcomputing.com/news. Accedido Agosto 2016.

[21] NodeJS. https://nodejs.org/en/. Accedido Agosto 2016.

[22] V8, desarrollado por Google. https://developers.google.com/v8/. Accedido Agosto 2016.

[23] Node vs PHP benchmark. http://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/. Accedido Agosto 2016.

[24] Express, web oficial. https://expressjs.com/. Accedido Agosto 2016.

[25] Passport, web oficial. http://passportjs.org/. Accedido Agosto 2016.

[26] Github OAuth. https://developer.github.com/v3/oauth/. Accedido Agosto 2016.

[27] Nodemailer. https://nodemailer.com/. Accedido Agosto 2016.

[28] OpenCV Haar Feature-based Cascade Classifier for Object Detection, OpenCV 2.4.13 Official Documentation.

http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_ 1.1 Conclusions and future worksclassification.html. Accedido Agosto 2016.

[29] SIFT – The scale invariant feature transform. http://www.inf.fu-berlin.de/lehre/SS09/CV/uebungen/uebung09/SIFT.pdf. Accedido Agosto 2016.

[30] Abhishek Kumar Annamraju y Akash Deep Singh. Analysis and Optimization of Parameters used in Training a Cascade Classifier, Society for Science and Education. http://scholarpublishing.org/index.php/AIVP/article/view/1152/626. Accedido Agosto 2016.

[31] Ubuntu. http://www.ubuntu.com/. Accedido Agosto 2016.

[32] Git. https://git-scm.com/. Accedido Agosto 2016.

[33] SSL, Wikipedia. https://es.wikipedia.org/wiki/Transport_Layer_Security. Accedido Agosto 2016

Anexos

A.1 Manual de despliegue del servidor web

La siguiente guía se ha elaborado utilizando una máquina virtual con un sistema operativo **Ubuntu 14.04** [31].

1. Comprobar que la función de **IP forwarding** está activa en el servidor. Para ello ejecutar el siguiente comando:

cat /proc/sys/net/ipv4/ip_forward

```
ubuntu@ip-172-31-48-52:~$ cat /proc/sys/net/ipv4/ip_forward
0
ubuntu@ip-172-31-48-52:~$
```

***Figura 19**. Comprobación IP forwarding activo.*

En caso de que el resultado sea **0**, como se muestra en la **Figura 19**, continuar con la configuración del firewall. En caso contrario, saltar al punto 2.

a. Activar IP Forwarding en el fichero de configuración */etc/sysctl.conf.*

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

***Figura 20**.Activar IP Forwarding.*

Después de haber guardado la configuración, ejecutar el siguiente comando para activar la nueva configuración:

sudo sysctl -p /etc/sysctl.conf

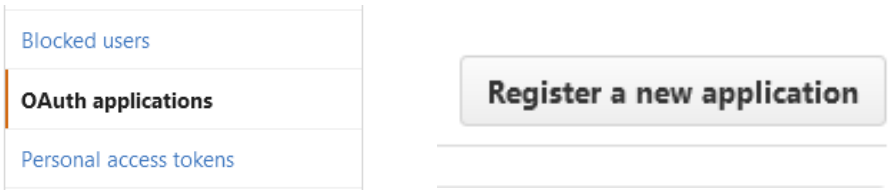
b. Ejecutar el script *fwconfig.sh* para incluir las reglas de firewall.

```
ubuntu@ip-172-31-48-52:~/PlatformRecognitionBackend$ chmod +x fwconfig.sh
ubuntu@ip-172-31-48-52:~/PlatformRecognitionBackend$ ./fwconfig.sh
Firewall rules added
```

***Figura 21**.Reglas de firewall.*

2. Ejecutar el script *setup.sh* para instalar todas las dependencias necesarias. Este script se encargará de instalar todas las librerías necesarias para ejecutar la aplicación. Entre ellas Python, NodeJS y OpenCV. La instalación de OpenCV se realiza mediante la compilación de la versión v2.4 obtenida desde el repositorio oficial utilizando Git [32].
3. Obtener los parámetros necesarios para utilizar la API de autenticación de GitHub. Para ello, desde el panel de configuración de cuenta en GitHub, seleccionar el

apartado **OAuth Applications** y crear una nueva aplicación, como se detalla en la **Figura 22**:



(a) (b)

***Figura 22**. (a) Menú de gestión de aplicaciones; (b) Opción para registrar nuevas aplicaciones.*

Para obtener las claves necesarias, será necesario rellenar el siguiente formulario:

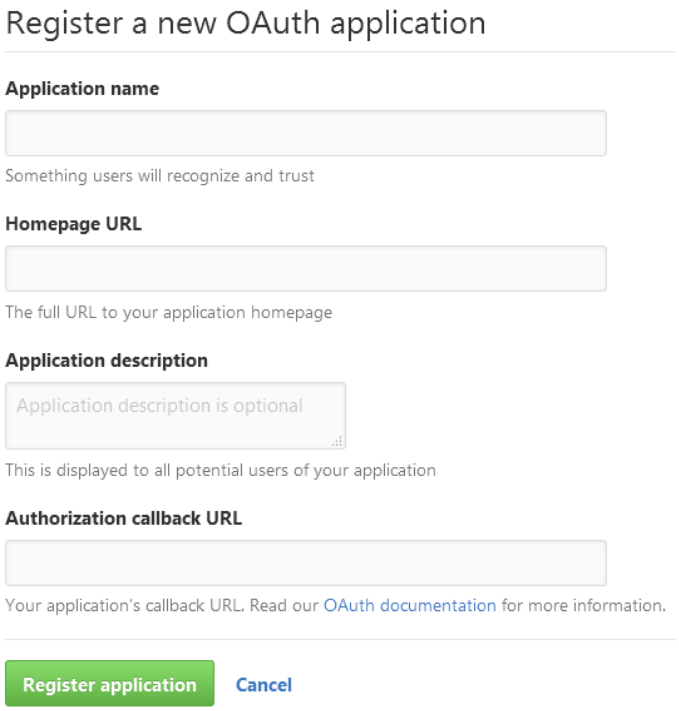


***Figura 23**. Formulario de creación de aplicaciones.*

Como **Authorization callback URL**, utilizar la URL pública del servidor y añadir el sufijo */auth/callback*. Eg:

URL pública http://example.url.com/

Authorization callback URL http://example.url.com/auth/callback

Una vez finalizado el proceso de registro, se generan dos parámetros necesarios para ejecutar la aplicación, *Client ID* y *Client Secret*, mostrados en la **Figura 24**.

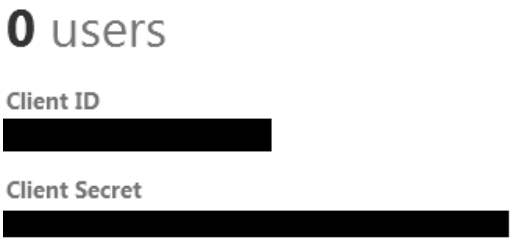


***Figura 24**. Parámetros GitHub OAuth.*

Éstos se utilizan para identificar la aplicación en el proceso de autenticación SSO. El servidor está configurado para extraer el valor de estos parámetros de la sesión del usuario, para ello se ejecutan los siguientes comandos:

export GITHUB_CLIENT_ID=<client_id>
export GITHUB_CLIENT_SECRET=<client_secret>
export GITHUB_CALLBACK_URL=<callback url>

4. Generar el certificado auto-firmado para utilizar el servicio SSL [33]. Ejecutar el siguiente comando en la consola:

   openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365

   Copiar los ficheros *key.pem* y *cert.pem* en el directorio raíz del proyecto.

5. Exportar a la sesión de usuario los parámetros de la cuenta de correo que se va a utilizar como emisor de correo electrónico:

   export MAIL_USER=<userMail@example.com>
   export MAIL_PASSWORD=<password>
   export MAIL_NAME<email account name>

6. Ejecutar el comando *npm start* en el directorio raíz del proyecto para iniciar la aplicación.

A.2 Manual de uso de la aplicación web

Para comenzar a utilizar la aplicación, acceder a la URL del servidor, la primera pantalla que se muestra es la de autenticación, **Figura 25**:

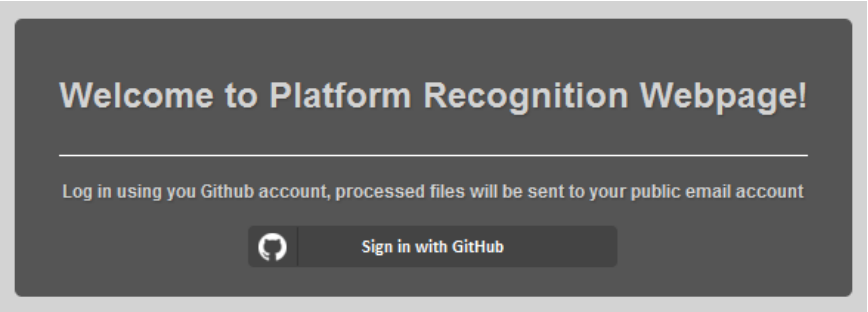
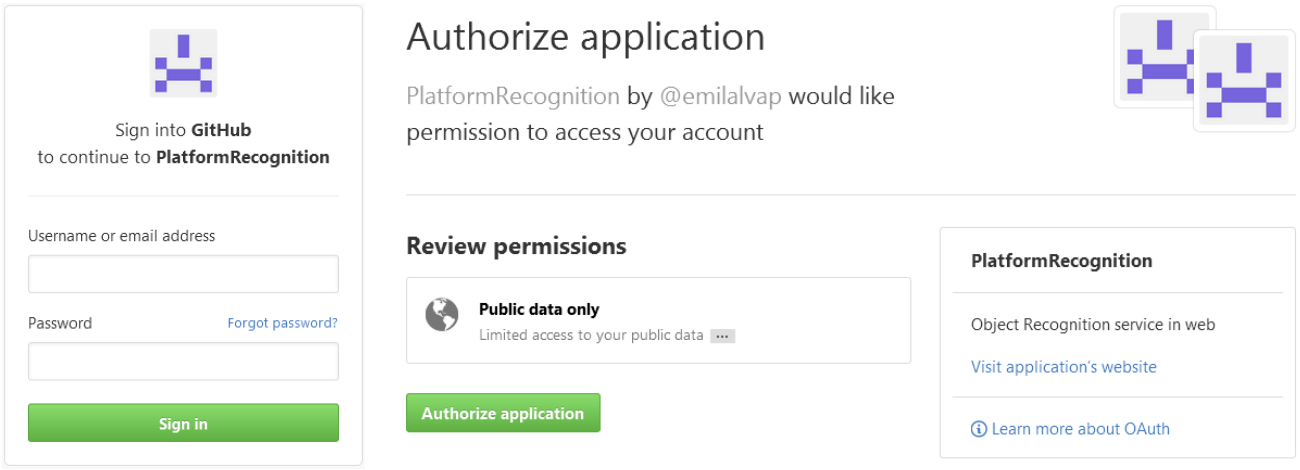


***Figura 25**. Página de autenticación.*

El enlace nos redirige a página de autenticación de GitHub, **Figura 26**. Si el usuario ya tiene una sesión abierta en esta plataforma, este paso se omite.



(a) (b)

***Figura 26**. (a) Autenticación de GitHub; (b) Autorización de aplicación.*

Si es la primera vez que se accede a la aplicación, GitHub solicitará la confirmación del usuario para permitir que la aplicación acceda a la información del perfil, **Figura 26**.

Una vez finalizado el proceso, se accederá a la página principal. En ésta, se puede seleccionar un fichero para enviar al servidor, **Figura 27**. Si la transmisión del fichero es correcta, la aplicación muestra un mensaje confirmando que el servidor está listo para realizar el procesamiento de la imagen, **Figura 28**.

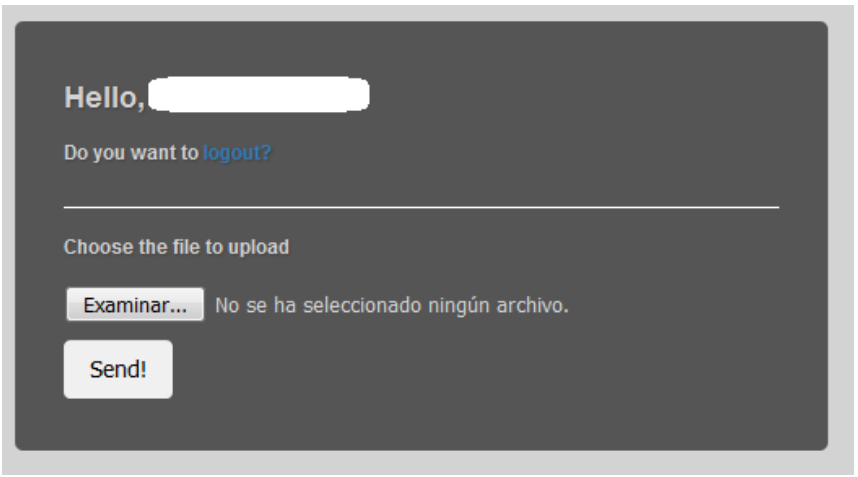


***Figura 27**. Página principal.*

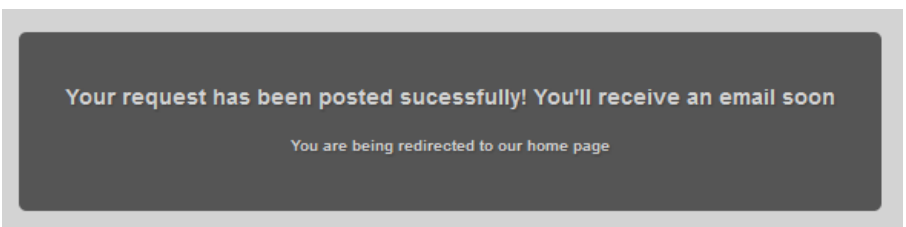


***Figura 28**. Confirmación de transmisión al servidor.*

El resultado del procesamiento de la imagen se enviará a la cuenta pública de correo configurada en el perfil de GitHub, **Figura 29**.

***Figura 29**. Correo electrónico con archivo adjunto.*

A.3 Manual de instalación de la aplicación de escritorio

- Instalación Python 2.7.

Para instalar el intérprete Python sólo hay ejecutar el instalador que incluye el DVD y en las opciones de instalación debemos habilitar la opción "ADD PATH".

- Instalación Numpy 1.10.

La instalación de este componente se realiza mediante el gestor de paquetes de Python. Situarse en la carpeta correspondiente a la versión que se quiera instalar (x86 o x64) y ejecutar el siguiente comando:

pip install <archive correspondiente versión>.whl

- Instalación PyQt 4.11.

Ejecutar el instalador incluido en la carpeta de versión seleccionada.

- Instalación OpenCV 2.4.

Para la instalación de la librería de tratamiento de imágenes, descomprimir el archivo *opencv-2.4.13* en el directorio raíz *C:* y copiar el archivo *cv2.pyd* de *\build\python\<x86 o x64>\2.7* en *<directorio de instalación Python>\Lib\site-packages*

A.4 Manual de uso de la aplicación de escritorio

Ejecutar el archivo *main.py* en el directorio raíz de la aplicación de escritorio. Se abre la ventana principal donde se encuentra la región dónde se mostrarán las imágenes, **Figura 29**.

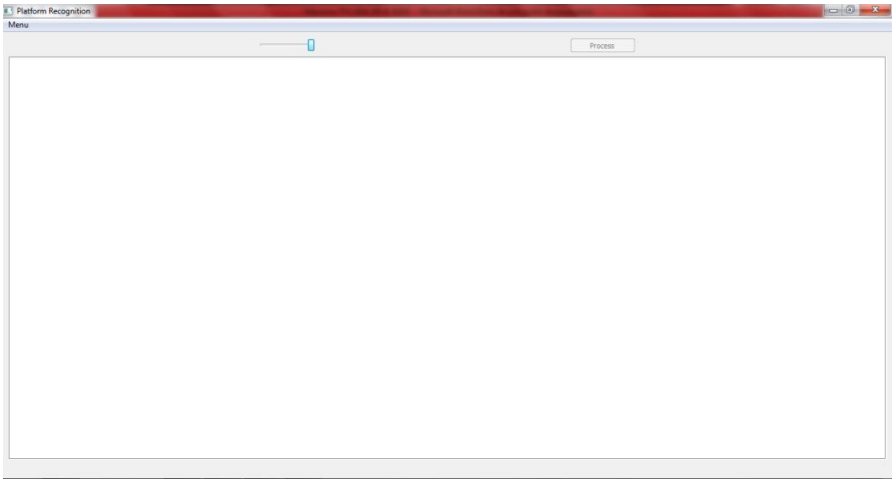


***Figura 30**. Ventana principal.*

Seleccionando la opción **Menu->Open** se abre un diálogo de selección que nos permite seleccionar una imagen para cargarla en la aplicación. Una vez seleccionada, la imagen se muestra en la región designada, **Figura 31**.

***Figura 31**. Imagen cargada en la aplicación.*

Activando la opción **Process**, comenzará el proceso de tratamiento de la imagen previamente cargada. Una vez finalizado el proceso, se abre una nueva pestaña en la que se muestra la imagen con los resultados obtenidos en el tratamiento de la imagen.

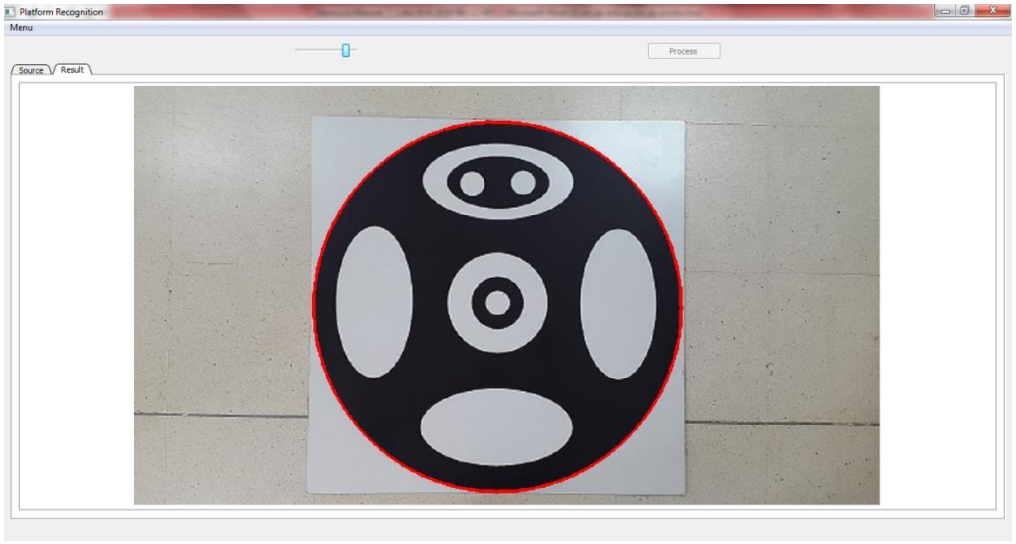


***Figura 32**. Resultado del tratamiento de imagen.*

Para repetir el proceso o utilizar otra imagen, basta con volver a cargar la imagen de nuevo en la aplicación.