Desarrollo de aplicaciones de control basadas en visión para el Dron DJI AIR 2S utilizando sensores internos.

Wilder Guerrero Tamayo

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Desarrollo de aplicaciones de control basadas en visión para el Dron DJI AIR 2S utilizando sensores internos.

Trabajo de graduación presentado por Wilder Guerrero Tamayo para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Desarrollo de aplicaciones de control basadas en visión para el Dron DJI AIR 2S utilizando sensores internos.

Trabajo de graduación presentado por Wilder Guerrero Tamayo para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

Vo.Bo.:

(f) _____

M. Sc. Carlos Esquit

Tribunal Examinador:

(f) _____

M.Sc. Carlos Esquit

(f) _____

M. Sc. Miguel Enrique Zea Arenales

(f) _____

Ing. Kurt Emmanuel Kellner

Fecha de aprobación: Guatemala, 13 de febrero de 2025.

Índice

Lista de figuras

Resumen

En este proyecto se desarrolló e implementó un sistema de control por visión autónomo para el dron DJI Air 2S utilizando OpenCV y el Mobile SDK (MSDK) de DJI en el entorno de Android Studio. Se logró integrar satisfactoriamente ambas tecnologías, permitiendo el procesamiento en tiempo real de datos visuales capturados por la cámara del dron y la implementación de un controlador PID para ajustar su posición con base en la detección de rostros.

Se realizaron pruebas que validaron la funcionalidad del sistema, logrando extraer y registrar datos como los valores de *pitch* y *yaw* generados por el controlador PID. Estos datos evidenciaron el correcto envío de señales a los rotores del dron en respuesta a la información procesada por la cámara. Además, se optimizó el rendimiento del sistema reduciendo la resolución de los fotogramas y ajustando la frecuencia de procesamiento, lo que mejoró la fluidez del *stream* de video y el desempeño general del dron.

Abstract

This project developed and implemented an autonomous vision-based control system for the DJI Air 2S drone using OpenCV and DJI's Mobile SDK (MSDK) within the Android Studio environment. The successful integration of both technologies enabled real-time processing of visual data captured by the drone's camera and the implementation of a PID controller to adjust its position based on face detection.

Tests validated the system's functionality, achieving the extraction and recording of data such as the pitch and yaw values generated by the PID controller. These values demonstrated the correct transmission of signals to the drone's rotors in response to information processed by the camera. Additionally, the system's performance was optimized by reducing the resolution of video frames and adjusting the processing frequency, improving the video stream's fluidity and the drone's overall performance.

CAPÍTULO 1

Introducción

En la actualidad, la integración de herramientas de visión por computadora con sistemas autónomos representa una de las áreas más innovadoras y desafiantes en el desarrollo tecnológico. Este proyecto surge con el propósito de explorar y aplicar estas tecnologías avanzadas en el dron DJI Air 2S, específicamente mediante el uso del Mobile SDK (MSDK) de DJI y la biblioteca de visión por computadora OpenCV. El objetivo principal fue diseñar e implementar un sistema autónomo que permita al dron detectar y rastrear objetos visuales, con un enfoque particular en el uso de técnicas de control PID basadas en la cámara del dron.

En primera instancia, se buscó reinstalar y validar las herramientas desarrolladas en trabajos previos, evaluando exhaustivamente sus funcionalidades y asegurando una base sólida para el desarrollo posterior. Adicionalmente, se abordó la recopilación, evaluación y visualización de los datos proporcionados por los sensores del dron, incluyendo la cámara, la unidad de medición inercial (IMU) y otros sensores integrados. Por último, se exploró el uso de estos datos para implementar técnicas avanzadas de control basado en visión, incrementando la autonomía del dron en distintos entornos, tanto dentro como fuera del Robotat.

Este documento se organiza de la siguiente manera: en los primeros capítulos se describen los fundamentos teóricos y técnicos que sustentan el proyecto, incluyendo una revisión de la literatura relevante y un análisis detallado de las herramientas utilizadas. Posteriormente, se

presenta el desarrollo del proyecto, detallando las etapas de instalación, integración y optimización de los sistemas implementados. En los capítulos finales, se discuten los resultados obtenidos, se presentan las conclusiones derivadas del trabajo y se ofrecen recomendaciones para futuros desarrollos en este ámbito. Este enfoque proporciona una visión integral del proceso llevado a cabo y de los avances logrados, posicionando este proyecto como un paso importante hacia la implementación de sistemas autónomos basados en visión por computadora.

CAPÍTULO 2

Antecedentes

El dron DJI AIR 2S es una herramienta versátil que ofrece numerosas cualidades, haciéndolo ideal para el desarrollo de diversos proyectos. Estas características resultan especialmente atractivas para estudiantes de las facultades de Ingeniería Mecatrónica y Electrónica. Debido a su potencial, ya se han realizado trabajos previos con este dron. En las siguientes secciones, se podrán conocer otros proyectos interesantes realizados con drones, así como el proyecto anterior desarrollado en la Universidad del Valle de Guatemala.

2.1. Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica

La realización de este proyecto[1] surgió de la necesidad de la universidad de contar con un espacio dedicado exclusivamente a la experimentación robótica. Para ello, se desarrolló y montó una red local inalámbrica de comunicación Wi-Fi que, junto con un sistema de captura de movimiento de la marca *OptiTrack*, conformó un ecosistema robótico capaz de obtener información de la pose. Este sistema permitía a varios agentes obtener sus poses en tiempo real dentro del ecosistema, lo cual es fundamental para diversos experimentos en robótica. Se logró conectar un máximo de 11 dispositivos al ecosistema, obteniendo mediciones con un error del 5.28 por ciento en comparación con la medida real.

2.2. Integración del dron DJI Air 2S con el ecosistema Robotat empleando el *mobile SDK* del fabricante DJI.

Este trabajo [2], al ser el predecesor del actual proyecto, tuvo como objetivo principal la creación de una aplicación para la plataforma móvil Android. Esta aplicación permitiría la conexión entre Robotat y el dron DJI AIR 2s, todo dentro de la misma interfaz. El proyecto se basó en los datos que se obtuvieron del sistema Robotat, con el fin de desarrollar un sistema de movimiento automatizado que utilizara los datos de vuelo del dron y la información de movimiento de Robotat en tiempo real. El SDK proporcionado por DJI permitió acceder a diversos sensores, como el módulo GPS, brújula, velocidad de vuelo y altitud, así como a la cámara del dron. Esto facilitó conocer su ubicación aproximada. La integración de la aplicación de control del dron y el ecosistema de robots permitió una ubicación más precisa.

Mediante el protocolo de comunicación TCP/IP, se logró enviar datos a la aplicación de control del dron, permitiendo su autonomía según las necesidades. Se desarrolló una aplicación funcional de Android capaz de recibir y procesar los datos de los sensores propioceptivos del dron. Sin embargo, la comunicación entre esta aplicación y Robotat, a través de una computadora con el software MATLAB, utilizando el mismo protocolo TCP/IP, resultó defectuosa debido al excesivo retraso en la transmisión de datos. Esta limitación afectó el desarrollo de algoritmos para el funcionamiento autónomo del dron, ya que la información recopilada por la aplicación de Robotat no era eficiente debido a las fallas en la comunicación.

Figura 1

Imagen del dron DJI 2S en funcionamiento

Nota. Adaptada de [2].

2.3. *Build Your Own Visual-Inertial Drone: A Cost-Effective and Open-Source Autonomous Drone*

Se requería una plataforma efectiva en costos y de grado de investigación que incorporara funcionalidades de inercia visual, así como capacidades de aterrizaje y despegue vertical (VTOL) asistido por odometría. Para lograr esto, se empleó un sensor VI (inercial visual) disponible comercialmente, una computadora integrada en el sistema y una plataforma de cuadricóptero. Todos estos componentes, al ser productos de fábrica, no fueron alterados después de su compra. Esto permitió mantener las especificaciones estándar, facilitando que otras personas pudieran replicar el proyecto con los mismos componentes calibrados de manera similar.

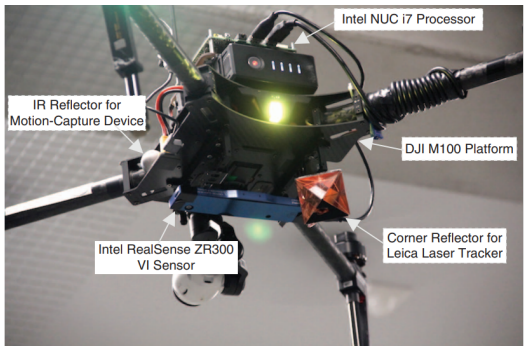
Con base en lo anterior, se desarrolló un sistema de calibración e identificación que facilitó el uso de odometría VI y la fusión de múltiples sensores, así como la implementación de modelos de control predictivo utilizando únicamente productos comerciales[3]. Como resultado, se evitó parcialmente el tedioso proceso de ajuste de parámetros para generar un sistema completo de control.

El sistema fue probado tanto en interiores como en exteriores. En interiores, se utilizó

un sistema de captura de movimiento, mientras que en exteriores se empleó un sistema de seguimiento por láser. Durante estas pruebas, el sistema realizó tareas de seguimiento de trayectorias en presencia de perturbaciones externas como el viento. Se logró recorrer una distancia considerable, alcanzando los 180 metros en una granja. La precisión obtenida en relación con las referencias establecidas fue de 0.036 metros

Figura 2

Imagen del cuadrotor utilizado con sus dispositivos señalados



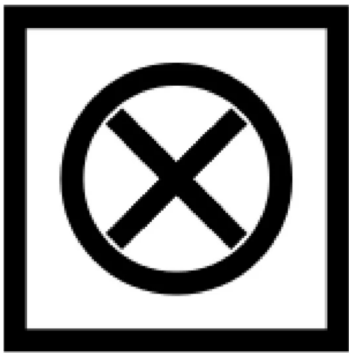
Nota. Adaptada de [3].

2.4. *Visual Servoing Approach to Autonomous UAV Landing on a Moving Vehicle*

Actualmente, muchas de las aplicaciones en robots aéreos requieren la habilidad de aterrizar en objetos en movimiento, como camiones de entrega, botes de búsqueda, entre otros. Este trabajo presento un método para aterrizar en un vehículo móvil a 15 km/h [4]. Para lograr esto, se utilizó un controlador de servo visual que emplea comandos de velocidad en las tres direcciones, calculados en el espacio de imagen de la cámara, para determinar la distancia y posición al punto de aterrizaje, como se puede ver en la Figura 3. Este sistema fue probado en entornos simulados y reales, logrando aterrizajes exitosos en vehículos en movimiento, demostrando la efectividad del método incluso en condiciones desafiantes como cambios de luz y reflejos solares, tal como se muestra en la Figura 7.

Figura 3

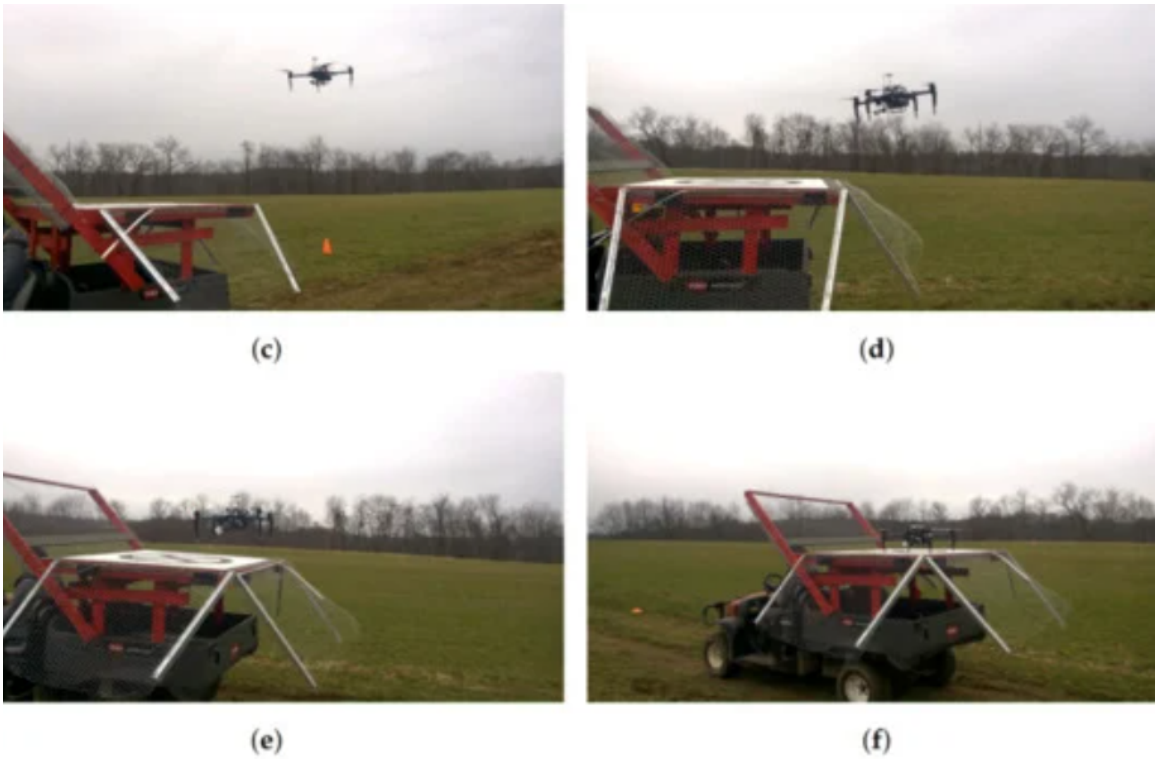
Imagen de la base donde aterrizar



Nota. Adaptada de [4].

Figura 4

Dron aterrizando en vehículo en movimiento



Nota. Adaptada de [4].

CAPÍTULO 3

Justificación

La universidad dispone de un conjunto de drones DJI AIR 2 S, adquiridos mediante una donación de USAID, con el objetivo de fortalecer el uso de plataformas autónomas en sus programas académicos. La intención primordial es desarrollar sobre estos drones y utilizarse en practicas de laboratorios de clases del departamento de Ingenieria Mecatrónica y Electrónica, como Sistemas de Control y Robótica. La única modalidad para realizar desarrollos sobre estos drones es a través de plataformas comerciales, específicamente utilizando un SDK móvil proporcionado por DJI para dicho dron. Esto brinda a los usuarios la capacidad de crear aplicaciones para su control, representando así una oportunidad para el desarrollo de aplicaciones autónomas. Sagastume en [2] se dedicó al desarrollo de una aplicación destinada a controlar el dron y a optimizar su autonomía, integrándose en el ecosistema de Robotat [1]. Durante el proceso de desarrollo, se estableció una comunicación con el sistema de captura de movimiento OptiTrack mediante el protocolo de comunicación TCP/IP. Esta integración posibilitó la adquisición eficiente de posiciones dentro del campo de visión del sistema de captura, facilitando de este modo el desarrollo de plataformas autónomas. Sin embargo, tras una serie de pruebas, surgieron diversos obstáculos, siendo el más destacado el tiempo de respuesta a los comandos, que en ocasiones alcanzaba los cinco segundos. Esta demora significativa en la respuesta a los comandos limitó la capacidad de aprovechar al máximo las capacidades del dron.

Eliminando la limitación de operar dentro del ecosistema Robotat, se llevará a cabo una investigación y desarrollo directamente en el SDK móvil para aprovechar al máximo las capacidades del dron, centrándose en los sensores integrados en el propio dispositivo orientándonos al uso de la cámara del dron para aplicaciones basadas en visión. Esta nueva perspectiva constituye una vía prometedora hacia la autonomía, y plantea la oportunidad de investigar los alcances que pueden lograrse empleando exclusivamente los sensores integrados en el dron.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar las capacidades de control basado en visión del dron DJI AIR 2S , enfocadas en aplicaciones didácticas, fuera del ecosistema de Robotat.

4.2. Objetivos específicos

- Reinstalar y validar nuevamente las herramientas desarrolladas en trabajos previos. Evaluar exhaustivamente la aplicación móvil desarrollada, poniendo a prueba todas las funciones implementadas.
- Recibir, recopilar, evaluar y mostrar los datos emitidos por los sensores del dron, incluyendo los sensores de proximidad, la Unidad de Medición Inercial, la cámara, entre otros.
- Utilizar la información de los sensores, principalmente la cámara con el fin de investigar y aplicar técnicas de control basado en visión para aumentar la autonomía del sistema fuera o adentro del Robotat.

CAPÍTULO 5

Alcance

Este proyecto logró integrar herramientas avanzadas de visión por computadora con el dron DJI Air 2S, utilizando OpenCV y el Mobile SDK (MSDK) de DJI. A pesar de los avances logrados, la implementación enfrentó múltiples desafíos significativos debido a las limitaciones inherentes de las herramientas de software proporcionadas por DJI.

El MSDK, diseñado exclusivamente para desarrollo móvil, resulta limitado en comparación con los SDK más robustos que DJI ofrece para sus drones de gama alta, los cuales permiten desarrollo en plataformas de PC con mayor flexibilidad. Esta decisión de DJI responde a una estrategia comercial que busca restringir las capacidades avanzadas en modelos más económicos, como el DJI Air 2S. Además, la documentación del MSDK es escasa y desactualizada, y los ejemplos prácticos son mínimos, lo que complicó aún más su integración con OpenCV, una biblioteca que está más orientada a entornos basados en C++ o Linux y no tanto a sistemas Android, que dependen de lenguajes como Java y Kotlin.

Estos factores no solo ralentizaron el desarrollo, sino que también elevaron la complejidad técnica del proyecto, ya que fue necesario integrar y alinear dos SDK con enfoques completamente distintos, gestionando además las dificultades de programación inherentes a Android Studio. Por otro lado, la legislación nacional también influyó en las pruebas prácticas. La normativa requiere una licencia específica emitida por la Dirección General de Aeronáutica Civil para operar drones, y los drones pertenecientes a la universidad están

registrados bajo su nombre. Cualquier accidente durante las pruebas podría haber implicado responsabilidades legales para la institución. Por esta razón, las pruebas se realizaron sin instalar las hélices, limitando la validación directa de los algoritmos de control PID y afectando parcialmente los objetivos originales planteados.

A pesar de estas limitaciones, se logró validar el comportamiento del sistema observando las respuestas de los cuadrorotores y verificando los valores generados por el PID en condiciones simuladas. Este trabajo establece una base sólida para investigaciones futuras, y se recomienda crear un entorno controlado y seguro que permita realizar pruebas completas con las hélices instaladas.

CAPÍTULO 6

Marco teórico

El presente marco teórico está diseñado para abordar los tres ejes fundamentales que sustentan este proyecto: el desarrollo en el entorno Android, la utilización del Mobile SDK (MSDK) proporcionado por DJI y la integración de OpenCV como herramienta principal para el procesamiento de imágenes. Estos elementos son críticos para la implementación del sistema de control autónomo del dron DJI Air 2S, ya que cada uno de ellos plantea desafíos específicos que deben ser resueltos para lograr los objetivos propuestos.

En primer lugar, se exploran las capacidades y características del entorno Android Studio y el lenguaje Kotlin, que constituyen la base para el desarrollo de la aplicación móvil. Posteriormente, se analiza el MSDK de DJI, la herramienta oficial para interactuar con el dron, destacando tanto sus capacidades como sus limitaciones. Finalmente, se profundiza en OpenCV, una biblioteca esencial para las aplicaciones de visión por computadora, cuya integración con el SDK de DJI y el entorno Android plantea retos significativos debido a sus distintas naturalezas técnicas.

6.1. Android Studio y Kotlin

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android. Este entorno proporciona una amplia gama de herramientas y carac-

terísticas diseñadas para facilitar la creación de aplicaciones de alta calidad para dispositivos Android. Entre las funcionalidades más destacadas se encuentra el emulador de Android, que permite a los desarrolladores probar y depurar sus aplicaciones en dispositivos virtuales con diferentes configuraciones de hardware y versiones del sistema operativo, eliminando la necesidad de contar con dispositivos físicos.

Otra herramienta clave es el sistema de compilación basado en Gradle, una plataforma flexible que gestiona las dependencias y automatiza tareas de construcción, simplificando el proceso de desarrollo y permitiendo configuraciones personalizadas para diversos entornos. Adicionalmente, el diseñador de interfaz de usuario (UI) facilita la creación y previsualización de interfaces mediante herramientas visuales de arrastrar y soltar, optimizando el diseño visual y el ajuste de los componentes.

Android Studio también integra sistemas de control de versiones como Git, lo que simplifica la gestión del código fuente y fomenta la colaboración en equipos de desarrollo. Estas características hacen de este entorno una herramienta robusta y eficiente para el desarrollo de aplicaciones Android [5].

Los desarrolladores que trabajan con Android Studio, generalmente utilizan Kotlin como lenguaje de programación estándar, el cual es un lenguaje moderno y eficiente para el desarrollo en esta plataforma. La adopción de tal en el desarrollo de aplicaciones Android ha crecido significativamente debido a sus ventajas sobre Java en términos de simplicidad, seguridad y funcionalidad. Los desarrolladores encuentran que Kotlin reduce la cantidad de código necesario y mejora la calidad general del software. Además, la interoperabilidad con Java permite una transición gradual y sin problemas para los equipos que ya están familiarizados con el ecosistema de Java [6].

6.2. Dron DJI Air 2S

El DJI Air 2S es un dron plegable de alto rendimiento que combina características avanzadas para diversas tareas [8].

Este dron está equipado con una cámara capaz de grabar videos en resolución 5.4K a

30 fps y 4K a 60 fps, lo que lo hace ideal para aplicaciones que requieren alta calidad de imagen y detalle. La cámara está montada en un gimbal de tres ejes que estabiliza en los movimientos de inclinación, balanceo y giro (*roll, pitch y yaw*), asegurando tomas fluidas y libres de vibraciones.

El sistema de vuelo del DJI Air 2S permite un tiempo de vuelo de hasta 31 minutos en condiciones ideales. Además, utiliza el sistema de transmisión OcuSync 3.0, que ofrece un alcance de hasta 12 kilómetros y permite alcanzar velocidades de hasta 19 m/s en modo Sport. En cuanto a seguridad, el dron cuenta con sensores en cuatro direcciones (adelante, atrás, arriba y abajo) y un sistema de posicionamiento dual GPS + GLONASS que mejora la precisión y estabilidad durante el vuelo.

La batería del DJI Air 2S tiene una capacidad de 3500 mAh y el dispositivo tiene un peso total de 595 gramos, lo que lo hace eficiente y ligero para su categoría. Por último, el dron puede ser controlado mediante la aplicación DJI Fly, que permite al usuario ver la transmisión en vivo de la cámara, así como editar y compartir imágenes y videos capturados.

Estas características hacen del DJI Air 2S una herramienta versátil y confiable para tareas que van desde la captura de imágenes de alta calidad hasta operaciones avanzadas de vuelo autónomo.

6.3. SDK del fabricante DJI

Un SDK, por sus siglas en inglés (Software Development Kit), es un conjunto de herramientas y recursos que los desarrolladores utilizan para crear aplicaciones específicas para una plataforma determinada, como un sistema operativo, un lenguaje de programación o un dispositivo. Estos kits proporcionan a los desarrolladores los componentes y funcionalidades necesarios para facilitar la creación de aplicaciones, lo que incluye bibliotecas, herramientas de desarrollo, documentación y ejemplos de código. Las bibliotecas contienen código preescrito que implementa funcionalidades comunes, como el acceso a hardware, manejo de datos o comunicación con redes. Las herramientas de desarrollo, como compiladores, depuradores y emuladores, facilitan el proceso de programación, mientras que la documentación ofrece guías, tutoriales y referencias para el uso de estas herramientas. Además, los ejemplos de

código ilustran casos prácticos de implementación en diferentes contextos.

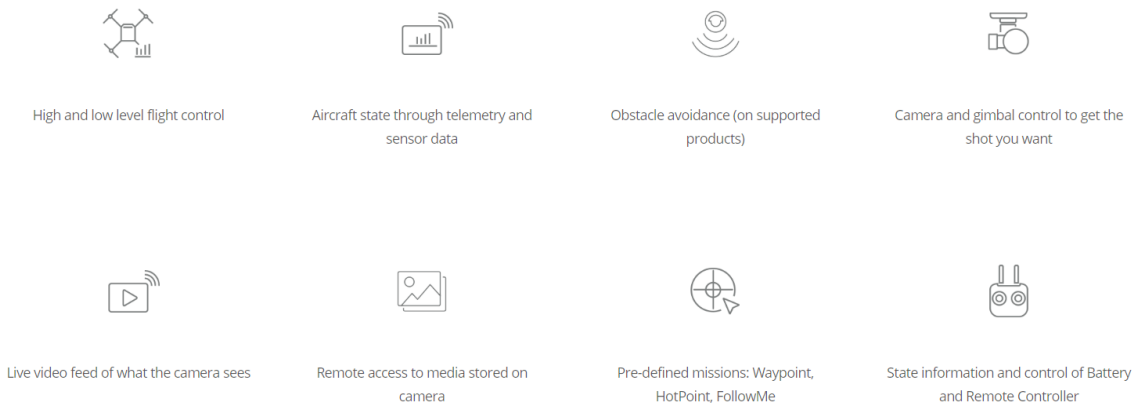
Los SDK son fundamentales para el desarrollo de software, ya que agilizan el proceso de creación de aplicaciones y permiten a los desarrolladores aprovechar las características específicas de una plataforma. En este proyecto, se empleó el MSDK V4, diseñado por DJI, como la herramienta principal para la integración con el dron DJI Air 2S. Este SDK ofrece diversas características que se describen a continuación [8].

En cuanto al control de vuelo, el SDK permite comandos tanto de alto nivel, como despegues y aterrizajes, como de bajo nivel, que brindan mayor precisión para ajustar aceleración y movimientos en los ejes de inclinación, balanceo y giro (*roll*, *pitch* y *yaw*). En el área de monitoreo del estado, proporciona información en tiempo real sobre niveles de batería, ubicación GPS, altitud, orientación y datos de sensores como acelerómetros, giroscopios y magnetómetros. Estas capacidades permiten un seguimiento detallado del estado del dron durante las operaciones.

Por último, el monitoreo de la batería se extiende tanto al dron como a su control remoto, asegurando un manejo eficiente de la energía. Estas características pueden ser gestionadas de manera más eficiente mediante el uso del DJI UX SDK, que proporciona elementos de interfaz de usuario predefinidos para optimizar el desarrollo de aplicaciones, en este proyecto no se profundizo en el uso de esta herramienta [8].

Figura 5

Imagen de características del dron DJI AIR 2S



Nota. Adaptada de [8].

6.4. *Visual Servoing*

El *visual servoing* o control basado en visión es una técnica que utiliza información visual obtenida de cámaras para controlar el movimiento de un robot. Este método, previamente empleado en robots manipuladores encargados de rastrear y captar objetos en cintas transportadoras en movimiento, se basa en la retroalimentación visual para guiar el robot hacia una posición o trayectoria deseada [9]. Es especialmente útil en aplicaciones que demandan precisión y adaptabilidad en entornos dinámicos.

Existen dos enfoques principales dentro del *visual servoing*, conocidos como *visual servoing basado en imagen* (IBVS, por sus siglas en inglés) y *visual servoing basado en posición* (PBVS). El primero calcula los errores directamente en el espacio de la imagen y genera comandos de control que minimizan estos errores, ajustando el movimiento del robot hasta que las características visuales captadas por la cámara coincidan con las posiciones deseadas [10]. Por otro lado, el enfoque basado en posición utiliza la información visual para estimar la ubicación tridimensional del objetivo en el espacio del robot. A partir de esta estimación, se generan comandos de control que dirigen al robot hasta que su posición coincida con la del objetivo [11].

El IBVS es menos susceptible a errores derivados de la calibración de la cámara, lo que lo hace adecuado para escenarios donde la precisión de los parámetros intrínsecos y extrínsecos de la cámara no es ideal. Sin embargo, el PBVS puede ofrecer un control más preciso en aplicaciones específicas que requieren un modelado más detallado del entorno. Ambos enfoques son complementarios y pueden combinarse para mejorar el rendimiento general del sistema, dependiendo de los requisitos de la aplicación.

6.5. OpenCV

OpenCV (*Open Source Computer Vision Library*) es una biblioteca de código abierto diseñada para aplicaciones de visión por computadora y aprendizaje automático. Fue desarrollada originalmente por Intel en el año 2000 y cuenta con el respaldo de una amplia comunidad de desarrolladores y contribuyentes. Su principal objetivo es proporcionar

herramientas avanzadas para el procesamiento de imágenes y video, permitiendo la implementación de soluciones en áreas como detección de objetos, seguimiento de movimiento, reconocimiento facial, calibración de cámaras, entre muchas otras aplicaciones en el ámbito de la visión por computadora [12].

Una de las características más destacadas de OpenCV es su naturaleza multiplataforma, que le permite ser compatible con diversos sistemas operativos, como Windows, macOS, Linux, iOS y Android. Esto facilita su implementación en una amplia gama de dispositivos, desde computadoras y teléfonos inteligentes hasta drones [13]. Además, OpenCV es compatible con diversos lenguajes de programación, incluidos C++, Python, Java y MATLAB, lo que amplía sus posibilidades de integración en proyectos variados [14].

La biblioteca se distingue también por su capacidad para procesar imágenes y videos en tiempo real, optimizando el uso de recursos computacionales. Esta fortaleza la convierte en una herramienta clave para aplicaciones donde la rapidez y la eficiencia son esenciales [15]. Asimismo, OpenCV incluye una amplia gama de funcionalidades, distribuyendo sus herramientas en módulos especializados. Entre ellos se encuentran herramientas para el procesamiento de imágenes, que abarcan tareas como conversión de colores, filtrado y detección de bordes; módulos para el reconocimiento de patrones, como la detección facial, clasificación de objetos y análisis de movimiento [16]; y funcionalidades para cámaras y calibración, que corrigen distorsión y mejoran la calidad de las imágenes capturadas [17]. Además, ofrece soporte para visión tridimensional, permitiendo la reconstrucción 3D y cálculos de profundidad [18].

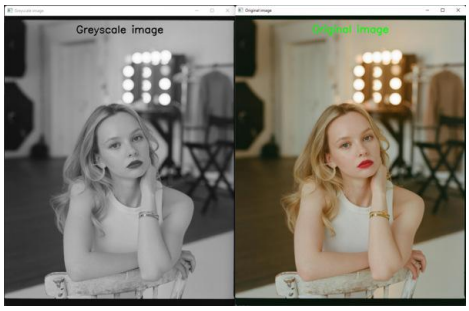
Entre los filtros y técnicas de procesamiento de imágenes que ofrece OpenCV, se encuentra la conversión a escala de grises (*grayscale*), la cual reduce la complejidad al centrarse exclusivamente en la intensidad de los píxeles, como se puede ver en la Figura 6, y el suavizado (*smoothing*), que atenúa el ruido al promediar los valores de los píxeles vecinos, tal como se observa en la Figura 7. Asimismo, cuenta con el filtrado de afilado (*sharpening*), empleado para resaltar detalles y contornos en la imagen (véase la Figura 8), y la detección de bordes mediante el método Canny, mostrada en la Figura 9. Para la identificación de formas geométricas, se incluyen la transformada de Hough para líneas (como se aprecia en la Figura 10) y para círculos (Figura 11). Por otro lado, la técnica de *thresholding* permite

segmentar la imagen de forma binaria según un umbral de intensidad (véase la Figura 12). En cuanto al reconocimiento de características, destacan ORB (*Oriented FAST and Rotated BRIEF*), capaz de detectar y describir puntos clave (Figura 13), y SIFT (*Scale-Invariant Feature Transform*), que identifica patrones invariantes a escala (Figura 14).

Para el análisis y detección de rostros, uno de los métodos más utilizados es el clasificador en cascada de Haar, respaldado por modelos preentrenados como *haarcascade-frontalface-default.xml*, que agilizan la localización de regiones faciales [20], tal como se muestra en la Figura 15. Además, la biblioteca incluye herramientas para el uso de marcadores ArUco [21], los cuales se basan en patrones binarios únicos para el rastreo de objetos y la calibración en aplicaciones de visión por computadora, como se ilustra en la Figura 16. Esta flexibilidad en la integración de múltiples técnicas y filtros hace que OpenCV sea altamente versátil para el desarrollo de proyectos de visión artificial en diversos contextos.

Figura 6

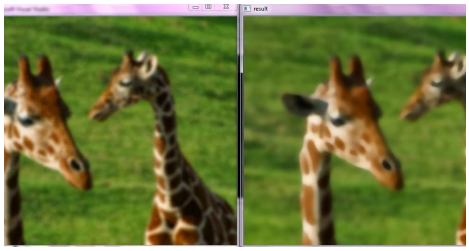
Imagen de ejemplo de filtro escala de grises



Nota. Adaptada de [24].

Figura 7

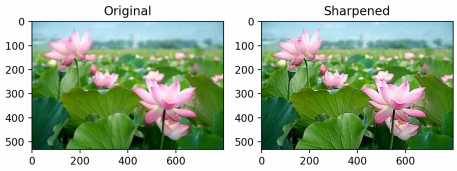
Imagen de ejemplo de filtro suavizado



Nota. Adaptada de [24].

Figura 8

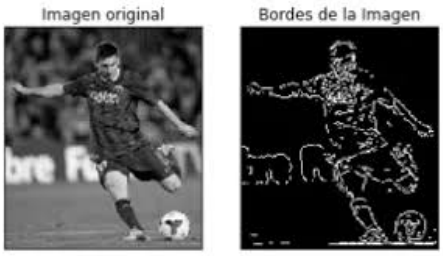
Imagen de ejemplo de filtro afilado



Nota. Adaptada de [24].

Figura 9

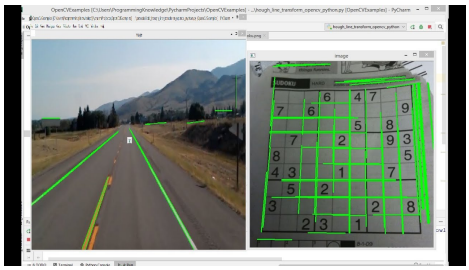
Imagen de ejemplo de filtro bordes de Canny



Nota. Adaptada de [24].

Figura 10

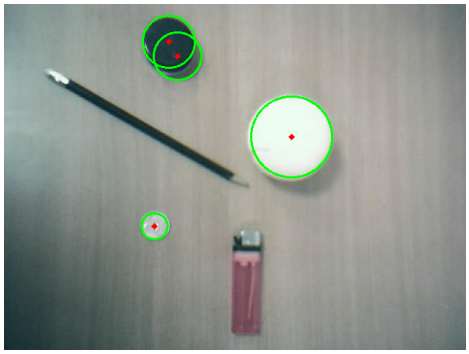
Imagen de ejemplo de filtro líneas de Hough



Nota. Adaptada de [24].

Figura 11

Imagen de ejemplo de filtro círculos de Hough



Nota. Adaptada de [24].

Figura 12

Imagen de ejemplo de filtro thresholding

Nota. Adaptada de [24].

Figura 13

Imagen de ejemplo de filtro ORB

Nota. Adaptada de [24].

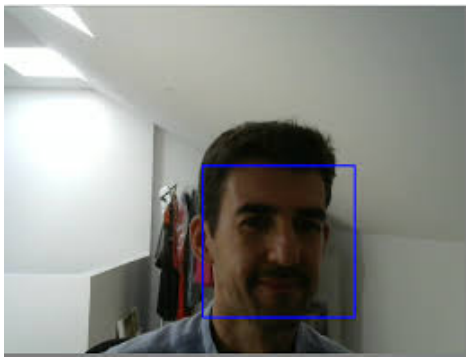
Figura 14

Imagen de ejemplo de filtro SIFT

Nota. Adaptada de [24].

Figura 15

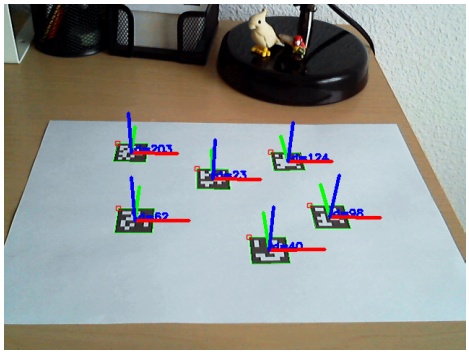
Imagen de ejemplo de detección de rostros



Nota. Adaptada de [20].

Figura 16

Imagen de ejemplo de marcadores ArUco



Nota. Adaptada de [21].

CAPÍTULO 7

Configuración del entorno de trabajo.

Este capítulo detalla las acciones realizadas para configurar un entorno funcional que permitiera la integración y validación del dron DJI Air 2S en el proyecto. Se describen actividades clave como la familiarización con las capacidades del dron, la instalación y configuración de herramientas esenciales como Android Studio y el MSDK, así como la evaluación del código desarrollado en trabajos previos. Cada sección aborda los retos técnicos enfrentados y las soluciones implementadas, sentando las bases para el desarrollo del sistema autónomo propuesto.

7.1. Familiarización con el dron

Se tomó la tarea de familiarizarse con el dron, para lo cual se instaló la aplicación DJI FLY en un dispositivo Samsung S10, la cual ofrece DJI para poder tener acceso a todas las capacidades para la mejor experiencia de usuario: uso de la cámara, grabar o tomar fotos, entre otros. Esto para familiarizarse con las capacidades del dron, en las figuras 17, 18 y 19 se pueden observar las experimentaciones realizadas con la cámara del dron.

Figura 17

Foto capturada con el dron accionado

Nota. Imagen capturada desde la interfaz de DJI para el control del dron.

Figura 18

Imagen panorámica

Nota. Imagen capturada desde la interfaz de DJI para el control del dron.

Figura 19

Imagen 360 grados

Nota. Imagen capturada desde la interfaz de DJI para el control del dron.

7.2. Instalación de Android Studio e integración del Mobile SDK

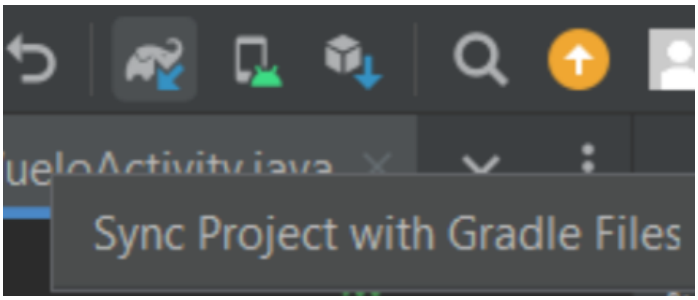
El uso de Android Studio en el desarrollo del proyecto implicó la adquisición de conocimientos específicos sobre esta herramienta. Para ello, se realizó un análisis detallado del trabajo previo [2], complementado con una investigación que incluyó la consulta de la documentación oficial de Android Studio y el estudio de tutoriales especializados. El trabajo se llevó a cabo en una computadora con sistema operativo Windows 11 Home, versión 23H2, lo que proporcionó un entorno adecuado para la ejecución de Android Studio y las herramientas asociadas. Este proceso estuvo acompañado de iteraciones constantes basadas en pruebas y ajustes, durante las cuales se enfrentaron diversos obstáculos técnicos. Uno de los primeros retos técnicos fue lograr la correcta compilación de un proyecto de prueba proporcionado por DJI, lo cual resultó fundamental para establecer la funcionalidad básica del entorno de desarrollo.

Se enfrentaron dificultades persistentes para compilar los códigos de muestra, a pesar de seguir todas las guías oficiales, las cuales resultaron estar desactualizadas según se confirmó a través de la experimentación y el apoyo de varios blogs de desarrolladores. Inicialmente, se intentó compilar en la versión más reciente de Android Studio en ese momento, Android Studio Koala | 2024.1.2, ya que varias guías recomendaban trabajar con la última versión. Sin embargo, no se logró compilar los proyectos de prueba.

Siguiendo la recomendación de [2], se optó por trabajar con Android Studio Dolphin | 2021.3.1 Patch 1, donde se descubrió que el problema radicaba en la incompatibilidad de las versiones de Gradle disponibles con el código. Fue necesario actualizar Gradle a versiones superiores a la 7.1, esto se realizó ingresando a la pestaña de *file*, y seleccionando *project structure* como se puede ver en la Figura 22 y asegurarse de utilizar un API mínimo compatible, eligiendo la versión 23 que ofrece un 98 por ciento de compatibilidad con los dispositivos. Todo esto se logró gracias a la documentación de Android Studio y a un proceso continuo de experimentación, que permitió identificar los requisitos necesarios para ejecutar correctamente el proyecto como se muestra en la Figura 23 . Otro obstáculo que retrasó considerablemente el proceso fue la dificultad para conectar el dispositivo a programar. A pesar de que el administrador de dispositivos del computador lo reconocía, Android Studio no lograba identificarlo. Se investigó exhaustivamente este problema, consultando videos y documentación hasta descubrir que era necesario actualizar unas herramientas del SDK para poder establecer la conexión con el dispositivo móvil. Esta herramienta se muestra en la Figura 24. Una vez actualizada, se pudo instalar el programa en el dispositivo y comenzar con la validación del trabajo previo.

Figura 20

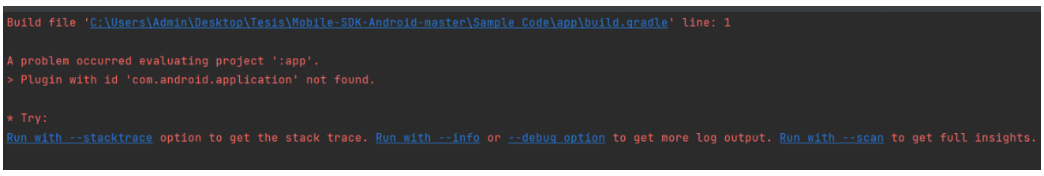
Botón para sincronización de archivos Gradle, necesarios para la implementación de SDKs



Nota. Imagen capturada por el autor desde Android Studio durante la configuración inicial del entorno.

Figura 21

Error constante experimentado a la hora de compilar antes de configuraciones de Gradle



Nota. Captura obtenida por el autor durante la etapa de compilación previa a la sincronización con Gradle.

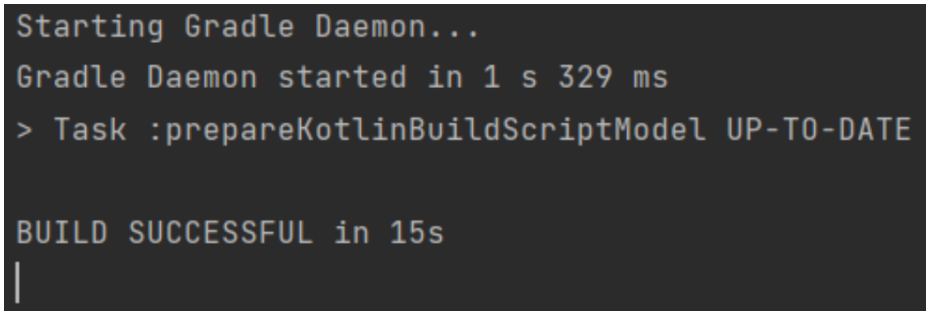
Figura 22

Configuraciones de Gradle en project structure

Nota. Imagen obtenida por el autor al configurar compatibilidad con SDKs.

Figura 23

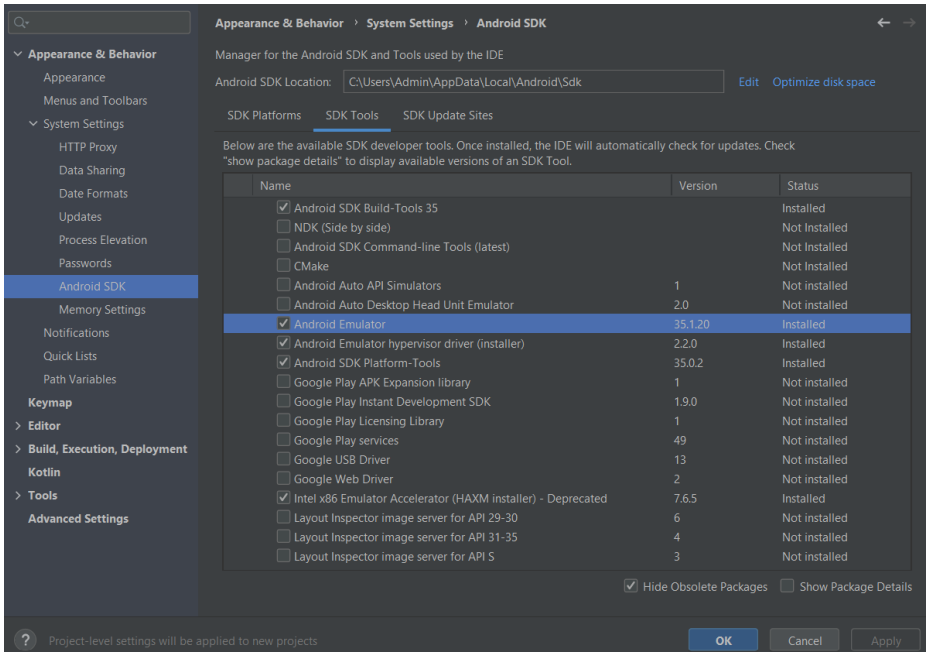
Compilación exitosa, gracias a actualización de archivos Gradle



Nota. Captura tomada por el autor mostrando compilación exitosa en Android Studio.

Figura 24

Actualización de herramientas de SDK de Android para trabajar correctamente en el software



Nota. Imagen capturada por el autor durante la actualización de herramientas del SDK en Android Studio.

7.3. Evaluación de la aplicación móvil

Gracias a un análisis exhaustivo del código de[2] y a una serie de experimentos de prueba y error, se logró comprender en detalle el funcionamiento de la aplicación creada. Se identificó cómo se procesa la información dentro del dron y cómo el MSDK interactúa con ella. Además, se comprendió el proceso mediante el cual se recibe la información de la Unidad de Medición Inercial (IMU) y cómo se envían los comandos necesarios a los rotores para que el dron se mueva.

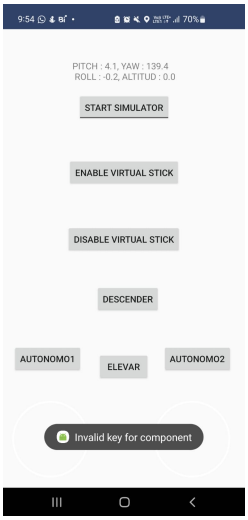
En la Figura 25 se muestra la interfaz de la aplicación desarrollada en el proyecto anterior. Es importante destacar que, si no se genera correctamente una *API Key* que es fundamental para el funcionamiento de cualquier aplicación que interactúe con un dron DJI, se producirá un error que impedirá el acceso a la aplicación, al cual se muestra en Figura25.

En la Figura 26, el recuadro rojo destaca los datos proporcionados por la IMU, donde se pueden observar las variables de *roll, pitch, yaw* y altitud. En el recuadro azul se encuentran los botones para habilitar y deshabilitar los *joysticks* virtuales, así como los comandos para ascender y descender. Cabe mencionar que el botón de descenso presento ciertas dificultades, ya que al presionarlo, el dron desciende hasta una cierta altura, pero no aterriza por completo. Por último, en el recuadro verde, se muestran los *joysticks* virtuales, que se activan al presionar el botón de *enable virtual joysticks*.

Tras la revisión del código, se ajustaron valores más controlados para las magnitudes de velocidad del dron. Inicialmente, el dron volaba a gran velocidad, pero después de configurar estos parámetros se pudo controlar el vuelo de manera más precisa y segura.

Figura 25

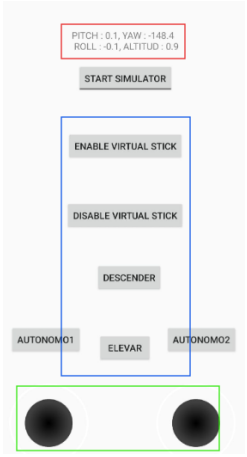
Mensaje de error generado por la aplicación debido a una configuración incorrecta o ausencia de la API Key



Nota. Imagen capturada por el autor durante pruebas con la aplicación sin configurar la clave API.

Figura 26

Interfaz de usuario de la aplicación en funcionamiento



Nota. Imagen obtenida por el autor mostrando la interfaz operativa tras una configuración exitosa.

Con ello, se logró validar lo desarrollado en el trabajo de Sagastume, estableciendo una base sólida para iniciar la exploración del control del dron mediante técnicas de visión por computadora.

CAPÍTULO 8

Desarrollo de control basado en visión

Los elementos fundamentales para un controlador basado en visión incluyen, en primer lugar, la adquisición de información proveniente de la cámara. Esta información debe ser procesada y utilizada como retroalimentación para generar las señales de control que permiten el movimiento de las hélices. A partir de este enfoque, se inicia el análisis de la conexión con la cámara, con el objetivo de profundizar en el manejo y procesamiento de los datos visuales para el control del dron.

8.1. Conexión a la cámara del dron

Uno de los principales problemas que surgieron durante el intento de conectar la cámara del dron fue el uso de una versión de código y documentación desactualizada. Se descargó un código de ejemplo para *First Person View* (FPV), basado en el MSDK versión 4.15 de DJI[22], mientras se revisaba simultáneamente la documentación oficial de DJI. Sin embargo, dicha documentación estaba desactualizada y no reflejaba los cambios implementados en versiones más recientes.

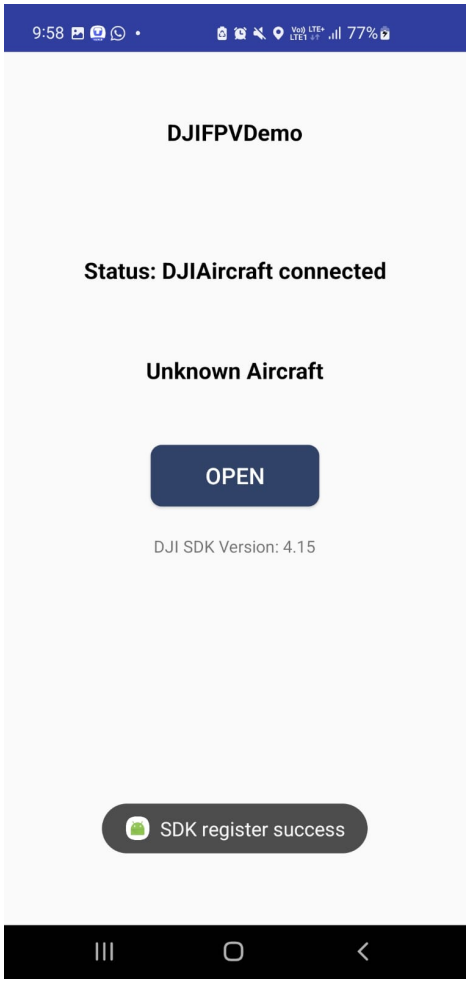
La confusión se originó porque la documentación no estaba alineada con la versión actual de Android Studio ni con las actualizaciones recientes del SDK. Las versiones más modernas de Android Studio presentan un entorno más optimizado y organizado, lo que hizo que se experimentara dificultades al trabajar con versiones antiguas tanto de código como de documentación. Esto resultó en pérdida de tiempo durante el proceso de depuración, ya que algunas instrucciones proporcionadas en la documentación no eran aplicables en el entorno actual.

Al descargar el código de muestra del repositorio de GitHub, se encontraron problemas ya que, aunque el código parecía estar bien estructurado, la versión del SDK (4.15) no era compatible con el dron DJI Air 2S que se estaba utilizando. Aunque lograba registrar la aplicación sin problemas, esta no reconocía el dron, se puede observar en la Figura 27 dicho suceso. Por lo tanto, tampoco accedía a la cámara del mismo, como se observa en la Figura 28. Tras una investigación más detallada, se descubrió que la versión del SDK 4.15 no ofrecía soporte para el modelo Air 2S, lo que generaba errores en la identificación del dispositivo.

Después de revisar nuevamente la documentación, a pesar de estar desorganizada y desactualizada, se observó que las versiones más recientes del MSDK sí ofrecían compatibilidad con drones más nuevos, incluyendo el DJI Air 2S. Finalmente, al actualizar a a la versión 4.17 , la cual sí soportaba el dron, se resolvió el problema de conexión y se pudo avanzar en el desarrollo de la aplicación, lo que se puede observar en las figuras 29 y 30.

Figura 27

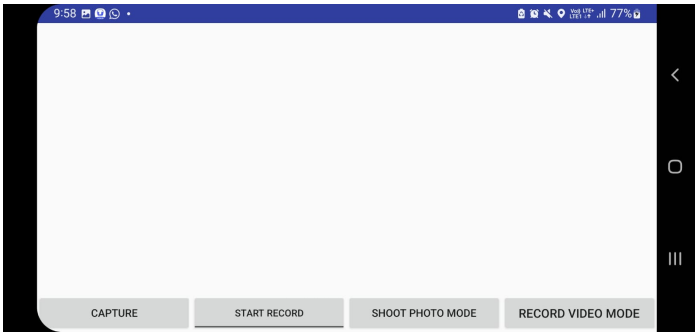
Inicio de aplicación de acceso a cámara sin identificar dron



Nota. Imagen obtenida por el autor desde la interfaz de la aplicación durante el arranque sin conexión.

Figura 28

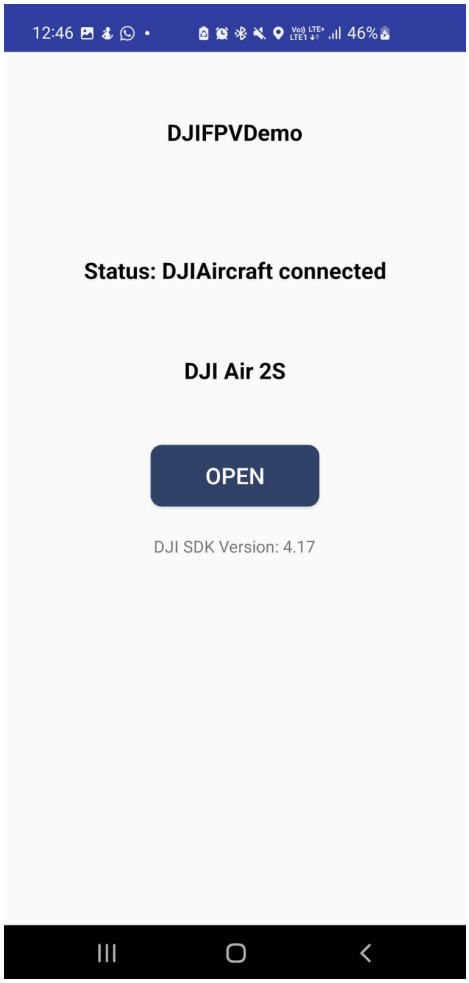
Aplicación abierta sin mostrar ninguna imagen



Nota. Captura tomada por el autor al ejecutar la aplicación sin recepción de señal de video.

Figura 29

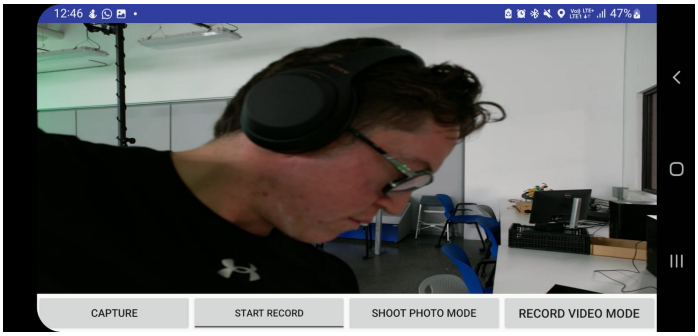
Inicio de aplicación de cámara con dron identificado



Nota. Imagen capturada por el autor desde la interfaz de la aplicación tras identificar correctamente el dron.

Figura 30

Cámara funcionando



Nota. Imagen obtenida por el autor con la cámara del dron funcionando correctamente en la aplicación.

8.2. Integración y desarrollo con OpenCV

El desarrollo del proyecto requirió la combinación de las capacidades de procesamiento de imágenes de OpenCV con las herramientas avanzadas proporcionadas por el SDK de DJI, enfocándose en la integración con el dron DJI Air 2S. Este proceso permitió no solo validar la compatibilidad entre ambas herramientas, sino también establecer una base técnica para el desarrollo de aplicaciones autónomas basadas en visión por computadora. Durante esta etapa, se realizaron pruebas iniciales con la cámara de un dispositivo móvil para evaluar el rendimiento de las herramientas en un entorno controlado antes de su implementación final en el dron.

La integración de OpenCV se llevó a cabo mediante la descarga de su SDK en su versión 4.10.0 para Android. El proceso incluyó su incorporación como un módulo en el proyecto, para lo cual fue necesario modificar los archivos de configuración del entorno de desarrollo (`settings.gradle` y `build.gradle`). A pesar de seguir las guías oficiales, se identificaron problemas técnicos relacionados con incompatibilidades en las versiones de Gradle y las librerías de Kotlin, que interrumpían la sincronización del proyecto y generaban errores durante la compilación.

Para resolver estas dificultades, se realizaron ajustes en la configuración de las herramientas, incluyendo la actualización de las dependencias de Kotlin y la implementación de versiones específicas de Gradle que fueran compatibles con los módulos del SDK, esto se puede observar en la Figura 31. Además, se diseñó un procedimiento de validación mediante registros (*logs*) que confirmaron la correcta integración del SDK de OpenCV, como se muestra en la Figura 32. Este proceso demostró la importancia de seguir con precaución, las recomendaciones oficiales [13], ya que cualquier paso omitido podía producir errores críticos durante la compilación o implementación de las funcionalidades.

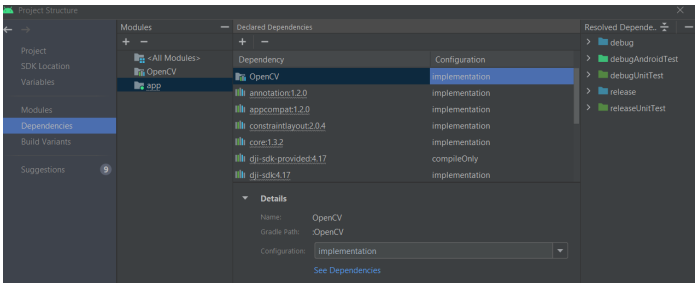
Se realizaron pruebas iniciales de los códigos de prueba incorporados en la descarga del SDK de Open CV, con la cámara de un dispositivo móvil, un Samsung S10, para evaluar el funcionamiento de la librería en tareas básicas como la detección de objetos y la grabación de video. En las figuras 33 y 34, se presentan ejemplos representativos de estas pruebas. Estos experimentos confirmaron el correcto desempeño de las herramientas en un entorno

controlado, lo que facilitó su posterior extensión al dron DJI Air 2S.

Este trabajo establece la base para integrar OpenCV de manera funcional con el SDK móvil de DJI, permitiendo implementar técnicas avanzadas de procesamiento de imágenes en aplicaciones autónomas para el dron. Este avance marca un paso clave hacia el desarrollo de sistemas más complejos y versátiles en el ámbito de la visión por computadora aplicada.

Figura 31

Captura de project structure, en donde se editan: módulos, dependencias, etc



Nota. Imagen obtenida por el autor en Android Studio durante la configuración del entorno de desarrollo.

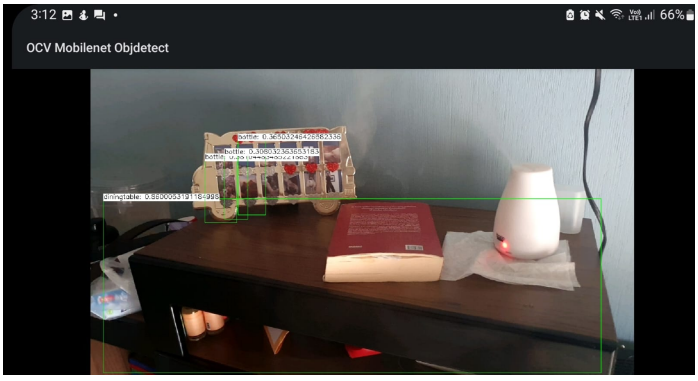
Figura 32

Captura en donde se muestra la implementación exitosa del SDK de OpenCV

Nota. Imagen capturada por el autor tras validar correctamente la integración del SDK de OpenCV.

Figura 33

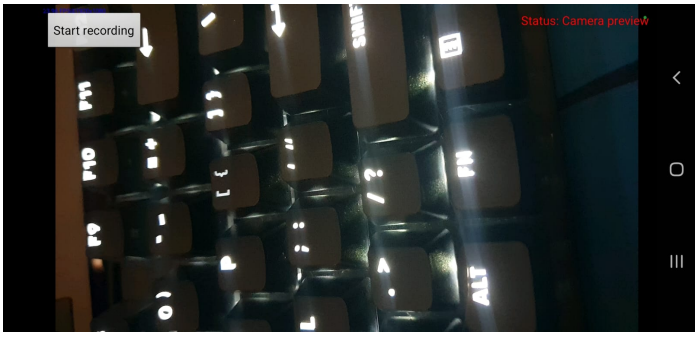
Detección de objetos aplicado a la cámara del dispositivo móvil



Nota. Imagen generada por el autor durante pruebas de detección de formas geométricas con OpenCV.

Figura 34

Grabación de video en la cámara del dispositivo móvil



Nota. Imagen obtenida por el autor al grabar video con la cámara del móvil.

8.3. Integración de OpenCV con el DJI Air 2S y optimización del *stream*

Una vez establecida la base de Open CV con las pruebas en el dispositivo móvil, se procedió a integrarse con lo realizado en la conexión a la cámara del dron, o sea, fusionar Open CV con las herramientas del MSDK. Inicialmente, los datos visuales del dron se capturaron empleando el componente *texture view*, que recibe la data del flujo de video ya decodificada del *DJICodecManager*, el cual transformaba el formato H.264 enviado por el dron[1]. Sin embargo, este enfoque presentó limitaciones significativas en la tasa de actualización del *stream*, lo que generaba retrasos perceptibles que afectarían la respuesta del controlador en tiempo real.

Para abordar este problema, se implementaron diversas estrategias de optimización. Una de las primeras medidas fue reducir la resolución de los *frames* capturados a 640x480 píxeles, lo que disminuyó la carga computacional sin comprometer en exceso la calidad del video procesado. Además, se adoptó un procesamiento por intervalos, en el cual se analizaba solo un *frame* de cada diez recibidos, permitiendo un balance entre la fluidez del *stream* y la eficiencia del procesamiento. Esta configuración fue ajustable, lo que permitió realizar experimentos con diferentes tasas de procesamiento para encontrar la más adecuada. Para mantener la estabilidad del sistema durante sesiones prolongadas, se reutilizaron objetos

[1] H.264 es un estándar de codificación de video ampliamente utilizado por su capacidad de ofrecer alta calidad de imagen con una tasa de bits eficiente, optimizando la transmisión de datos visuales en aplicaciones de video en tiempo real.

clave como *Mat* y *Bitmap*, evitando así saturar la memoria y reduciendo el impacto del *Garbage Collector*.

A pesar de estas optimizaciones, se identificaron limitaciones inherentes al uso del *texture view*, que requería convertir los datos crudos H.264 a un formato compatible con OpenCV. Este proceso añadía una sobrecarga computacional significativa. Explorando la documentación del MSDK, se descubrió una herramienta que permitía acceder directamente a los datos visuales en formato RGB[2], eliminando la necesidad de conversiones intermedias [23]. Este cambio mejoró sustancialmente el flujo de trabajo, al permitir un procesamiento más fluido y en tiempo real. Con este nuevo enfoque, se implementaron herramientas avanzadas de OpenCV, como detectores de rostros basados en cascadas Haar y filtros para detección de bordes, además de preparar el sistema para técnicas futuras, como el uso de marcadores ArUco y seguimiento de objetos[16].

Los resultados obtenidos reflejaron una mejora notable en la tasa de refresco del *stream* y en la estabilidad general del sistema. Este paso fue fundamental para garantizar que, en un futuro, la implementación de técnicas de detección de objetos no retrasara el tiempo de respuesta del controlador de los rotomotores del dron, optimizando así el rendimiento del sistema autónomo en condiciones reales. La Figura 35 ilustra el flujo de datos actualizado, mostrando las mejoras implementadas.

[2]El formato RGB (Red, Green, Blue) es un modelo de representación de imágenes basado en tres canales de color. En este proyecto, su uso permitió eliminar pasos intermedios de conversión, optimizando el flujo de datos entre el *MSDK* de DJI y OpenCV para aplicaciones de visión por computadora.

Figura 35

Flujo de datos visuales utilizando el formato RGB del MSDK de DJI

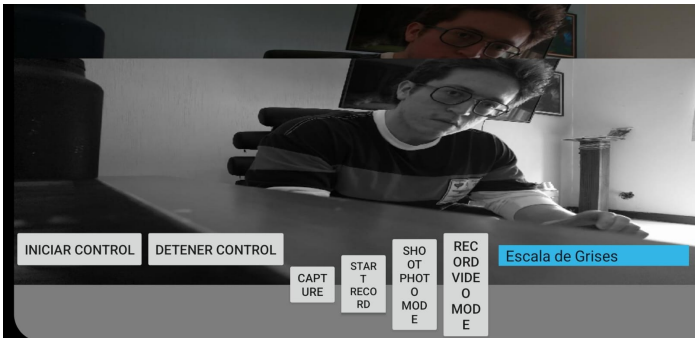
Nota. Diagrama elaborado por el autor para ilustrar el flujo optimizado de video deel dron.

8.4. Implementación de filtros y herramientas de procesamiento de imágenes

Una vez que se logró establecer una transmisión estable con una buena tasa de actualización, se procedió a realizar pruebas exploratorias para evaluar las capacidades de la cámara del dron DJI Air 2S. Para evaluar dichas capacidades y explorar el potencial de las herramientas proporcionadas por OpenCV, se realizaron pruebas aplicando distintos filtros de procesamiento de imágenes [24]. Estos filtros se utilizaron exclusivamente como una etapa exploratoria para entender el comportamiento y las capacidades del sistema en la detección de características visuales, identificando aquellos que serían útiles para tareas específicas, como la detección de rostros. Durante esta fase, se implementaron filtros de escala de grises (Figura 36), suavizado (Figura 37), afilado (Figura 38), detección de bordes con Canny (Figura 39), transformada de Hough para líneas (Figura 40) y círculos (Figura 41), *thresholding* (Figura 42), y detección de características mediante ORB (Figura 43) y SIFT (Figura 44).

Figura 36

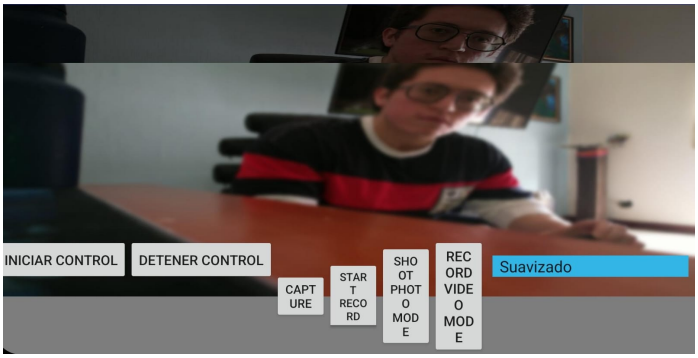
Transformación de la imagen a escala de grises



Nota. Imagen obtenida por el autor durante pruebas de conversión a escala de grises con OpenCV.

Figura 37

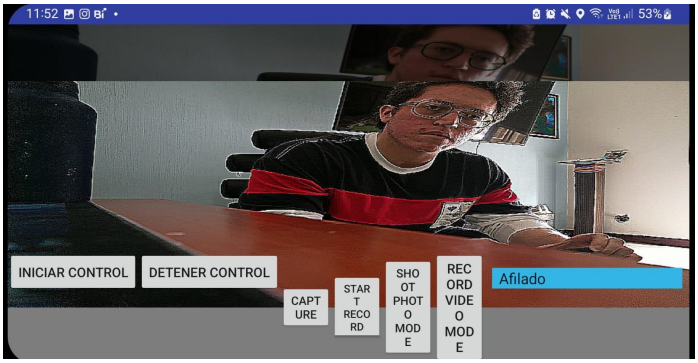
Imagen procesada con filtro de suavizado



Nota. Imagen generada por el autor al aplicar un filtro Gaussiano en el entorno de pruebas con OpenCV.

Figura 38

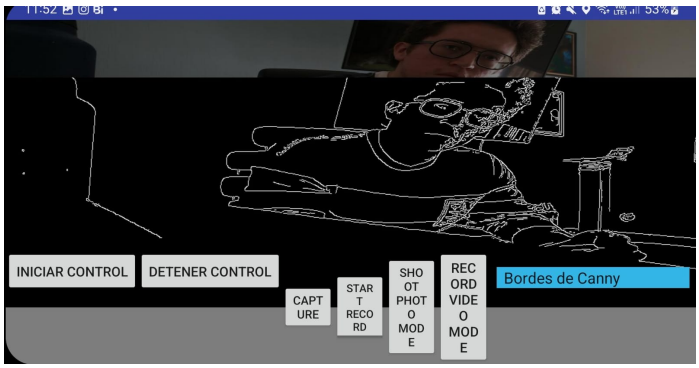
Imagen procesada con filtro de afilado



Nota. Imagen obtenida por el autor al aplicar un filtro de afilado durante pruebas de realce de bordes.

Figura 39

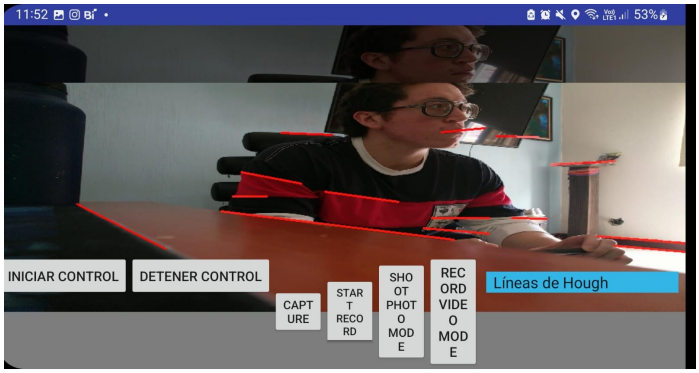
Detección de bordes mediante el método de Canny



Nota. Imagen generada por el autor durante la implementación del detector de bordes de Canny con OpenCV.

Figura 40

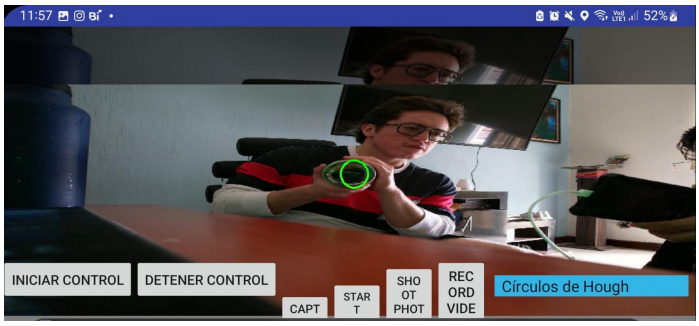
Detección de líneas utilizando la transformada de Hough



Nota. Imagen obtenida por el autor aplicando la transformada de Hough para detectar líneas rectas.

Figura 41

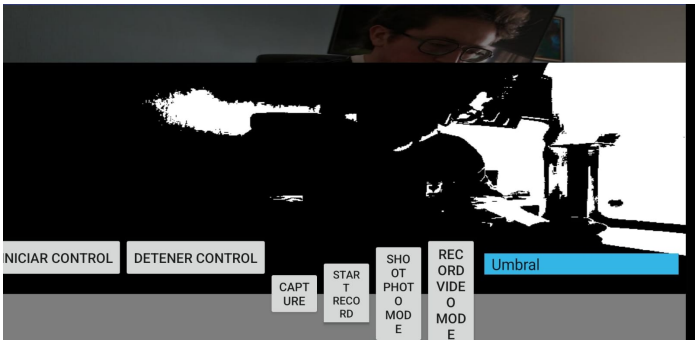
Detección de círculos con la transformada de Hough



Nota. Imagen capturada por el autor al implementar detección de círculos con la transformada de Hough.

Figura 42

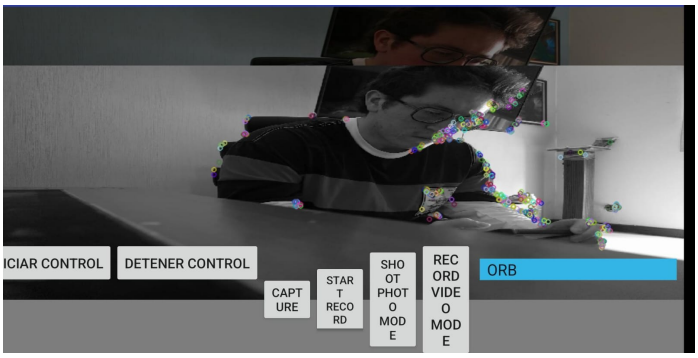
Imagen procesada utilizando thresholding



Nota. Imagen obtenida por el autor durante pruebas de segmentación mediante *thresholding*.

Figura 43

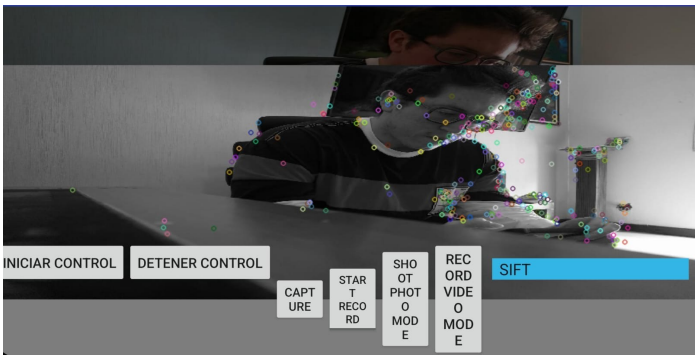
Detección de características con ORB



Nota. Imagen obtenida por el autor al aplicar ORB en pruebas con la cámara del dron.

Figura 44

Detección de características con SIFT



Nota. Imagen generada por el autor durante la aplicación del algoritmo SIFT con OpenCV.

Después de probar los filtros mencionados y validar que OpenCV ofrecía un potencial significativo al implementar visión por computadora con la cámara del dron, se procedió a

buscar una forma de identificar objetos y generar una referencia que serviría como base para diseñar la autonomía del sistema.

Inicialmente, se comprobó que el reconocimiento facial funcionara como una herramienta para identificar rostros en tiempo real. Durante esta etapa, el proceso de detección incluyó varios pasos clave. Primero, la imagen capturada por la cámara del dron fue convertida a escala de grises para reducir la complejidad computacional, permitiendo que los algoritmos operaran más eficientemente al enfocarse en la intensidad de los píxeles en lugar de la información de color. Posteriormente, se aplicó el clasificador en cascada[20] para identificar regiones que correspondían a rostros humanos.

Una vez validado su funcionamiento, se exploraron otras alternativas para generar referencias, como la implementación de marcadores ArUco. Estos marcadores ofrecían, en teoría, una referencia visual robusta. Durante la implementación de marcadores ArUco [21], se identificaron múltiples obstáculos técnicos que dificultaron su uso en el proyecto. En primer lugar, las herramientas relacionadas con ArUco, como `aruco.Dictionary-get` y `aruco.detectMarkers`, dependen de configuraciones nativas diseñadas para entornos basados en C++. A pesar de tener bien instalado el SDK de Open CV, al intentar utilizarlas en el entorno de Java, surgieron errores recurrentes que indicaban que las librerías necesarias no estaban correctamente vinculadas. Esto reveló una configuración incompleta de las librerías nativas de OpenCV, lo que requirió modificaciones adicionales en el archivo `CMakeLists.txt` y la instalación de dependencias específicas que no pudieron resolverse dentro del tiempo asignado al proyecto.

Otro factor que complicó la implementación de ArUco en este proyecto fue la falta de documentación detallada para Java en comparación con otros lenguajes como Python. Mientras que OpenCV proporciona múltiples ejemplos y recursos bien desarrollados para Python, la implementación en Java requiere una configuración más compleja y personalizada, lo que incrementó la dificultad de integración.

Adicionalmente, las limitaciones de recursos y tiempo llevaron a priorizar el uso de herramientas de OpenCV que pudieran ser implementadas directamente en Java sin la necesidad de configuraciones complejas o dependencias adicionales. Estas dificultades representaron una barrera técnica significativa para su implementación efectiva en el proyecto, lo que llevó

a descartar su uso y a profundizar en la detección de rostros como método principal para generar referencias visuales.

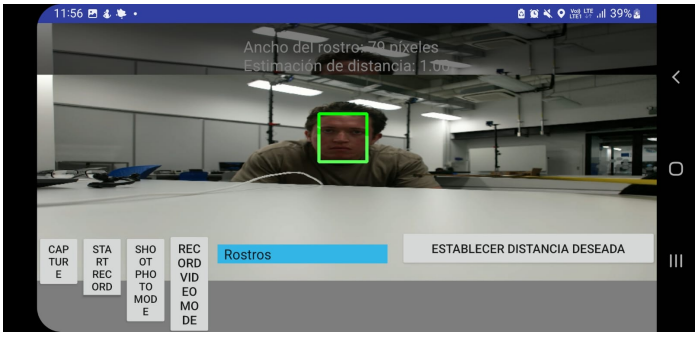
El reconocimiento facial ofreció ventajas significativas al permitir calcular el ancho del rostro detectado y su posición en la pantalla. Estas referencias se utilizaron para desarrollar un sistema de autonomía, donde el dron ajustaba los valores de control de *pitch* y *yaw*, variables que controlan los rotores para cumplir estos movimientos respectivos, validando el envío de estas señales que tienen la intención de mantener al dron a una distancia fija y orientado perpendicularmente al rostro detectado. Para calcular la distancia, se utilizó el ancho del rostro como métrica: mediante prueba y error, se estableció un ancho de referencia que correspondía a un metro de distancia. En cuanto a la orientación, se utilizó el componente en X del centroide del rostro detectado como guía para alinear el dron.

El procedimiento de detección de rostros resultó computacionalmente pesado, afectando tanto la fluidez del *stream* de video como la tasa de actualización, reduciendo la velocidad de procesamiento de los cuadros. Para mitigar este problema, experimentalmente se editaron los valores de la tasa de *frame* a procesar ya mencionada anteriormente, logrando mantener un rendimiento aceptable. Sin embargo, este enfoque también presentó limitaciones en escenarios con iluminación deficiente. Estas condiciones afectaron negativamente la detección, especialmente en ambientes con sombras pronunciadas o variaciones bruscas de luz, lo que resaltó la necesidad de optimizaciones adicionales en este tipo de entornos.

A pesar de los desafíos, el sistema demostró ser eficiente en la detección de rostros bajo condiciones controladas, logrando generar datos precisos para la orientación y distancia del dron, como se muestra en la Figura 45. Este resultado valida el uso del reconocimiento facial como una referencia viable para sistemas de control en visión por computadora, aunque subraya la importancia de futuras mejoras en la robustez del sistema, particularmente en ambientes no controlados y con variaciones dinámicas.

Figura 45

Detección de un rostro en tiempo real con su rectángulo delimitador



Nota. Imagen capturada por el autor durante la ejecución del sistema de detección facial en el entorno.

8.5. Implementación y validación del sistema de control autónomo basado en PID

Para la implementación del sistema autónomo, fue fundamental contar con referencias claras que sirvieran como base para el control del dron. Una vez establecidas estas referencias, se desarrolló un controlador PID encargado de regular los grados de libertad necesarios para que el dron pudiera alinearse y mantenerse a una distancia fija de un metro respecto a un rostro detectado.

El primer desafío fue comprender cómo modificar las señales enviadas a los cuatro rotores del dron para garantizar un control estable y preciso. Para abordar este reto, se tomó como referencia la implementación de *joysticks* virtuales desarrollada en el trabajo previo [2], lo que facilitó el entendimiento del envío de comandos a los rotores mediante el modo *Virtual Stick* del *MSDK* de DJI[25]. A partir de este conocimiento, se identificó cómo enviar las señales adecuadas a los valores de *pitch* y *yaw*, sentando las bases para un posible ajuste dinámico de la orientación y posición del dron.

Una vez identificado cómo modificar estos valores, se procedió con el diseño del controlador PID. El primer paso fue establecer la lógica de control para los dos grados de libertad a regular. En el caso del *yaw*, se comparó constantemente la posición del centroide en el eje X del rostro detectado con la referencia deseada, que debía ser cero para garantizar la alineación. Sin embargo, en esta etapa no se logró la alineación efectiva del dron, debido a

la limitación de no poder utilizar las hélices. En su lugar, se generaron las señales de control y, en primera instancia, se imprimieron en la terminal de Android Studio, como se muestra en la Figura 46. Esto permitió verificar que las señales se estaban generando correctamente. Una vez validada su existencia y su variación en función de la posición del rostro, se registraron los datos para su posterior análisis y graficación.

Para validar el correcto funcionamiento del sistema, se generaron gráficas a partir de los datos registrados, reflejando la variación de las señales de *yaw* y *pitch* en función de la posición del rostro detectado. Para la validación de *yaw*, se registró la generación de señales en un caso aislado en el que el ancho del rostro se mantenía constante mientras su centroide variaba. El objetivo era evidenciar la relación entre la variación del *yaw* y la posición del rostro. En la Figura 47 se observa la distribución del rostro detectado , mientras que en la Figura 48 se presenta la gráfica generada en base a la distribución recopilada. Se puede apreciar que la señal de *yaw* sigue el comportamiento esperado, con una transición de valores negativos a cero, lo que refleja el movimiento del rostro desde un extremo del campo de visión hasta el centro. Sin embargo, se identificaron puntos atípicos en la señal de *pitch*, ocasionados por falsos positivos en la detección de rostros debido a variaciones en la iluminación y accesorios utilizados por el usuario. En particular, se observó que elementos como lentes y barbas podían influir en la detección facial, generando inconsistencias.

De manera similar, se implementó un control sobre el *pitch* con el objetivo de mantener una distancia estable entre el dron y el rostro detectado. Para ello, se estableció un ancho de referencia experimentalmente estimado, equivalente a una distancia de un metro entre el dron y el rostro. Este valor se comparó continuamente con el ancho actual del rostro detectado, permitiendo identificar si el rostro estaba demasiado cerca, lo que se reflejaría en un mayor ancho, o demasiado lejos, resultando en un ancho menor.

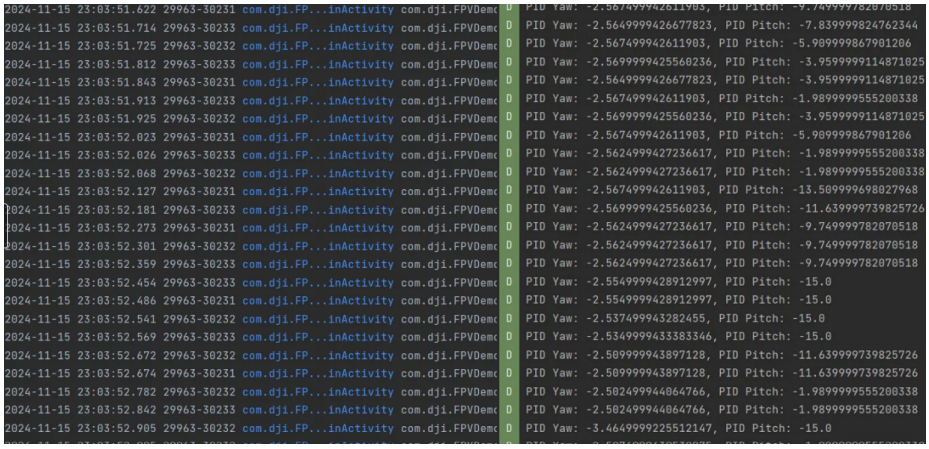
Para validar el correcto funcionamiento del sistema en términos de *pitch*, se registraron los datos en un caso aislado donde el centroide se mantenía constante y solo variaba el ancho del rostro detectado. En las figuras 49 y 50 se ilustra esta situación. Posteriormente, se generaron gráficas que muestran la variación de las señales de *pitch* y *yaw* en función del ancho del rostro. En la Figura 51, se observa que la señal de *pitch* varía de valores negativos a positivos, lo que concuerda con el caso registrado, donde el usuario primero se encontraba

cerca de la cámara, luego se alejaba hasta alcanzar la distancia de referencia de un metro y finalmente continuaba alejándose. Esta transición evidencia la intención del sistema de ajustar la distancia con base en la detección del rostro. No obstante, se identificaron ciertos puntos atípicos, atribuidos a los factores previamente mencionados.

Las gráficas obtenidas confirmaron que las señales de control respondían de manera coherente a los cambios en la detección. Como complemento, se observó experimentalmente la variación en el sonido de los rotores, lo que proporcionó una confirmación adicional del impacto de las señales enviadas. La combinación de estos métodos permitió validar la respuesta del sistema y su capacidad de adaptación ante los cambios detectados.

Figura 46

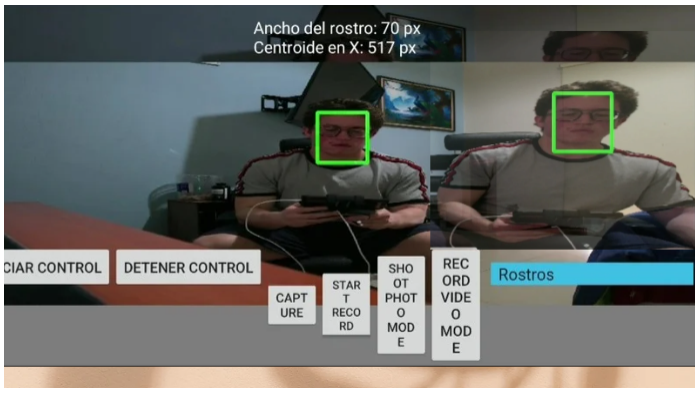
LOG de variación de valores PID en vivo



Nota. Imagen obtenida por el autor desde la consola de Android Studio.

Figura 47

Demostración visual de variación del centroide



Nota. Imagen capturada por el autor durante pruebas visuales del desplazamiento del centroide facial.

Figura 48

Variación del PID respecto a la posición del rostro en el tiempo (yaw)

Nota. Imagen generada por el autor para validación de respuesta.

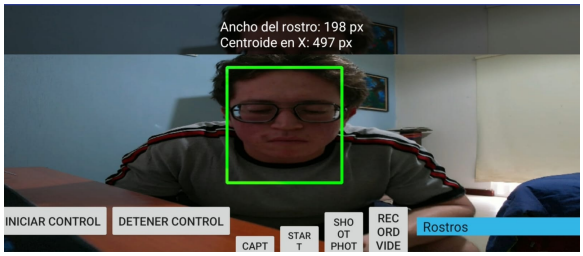
Figura 49

Demostración visual de variación del ancho (acercamiento)

Nota. Imagen obtenida por el autor durante una prueba de alejamiento del rostro ante la cámara del dron.

Figura 50

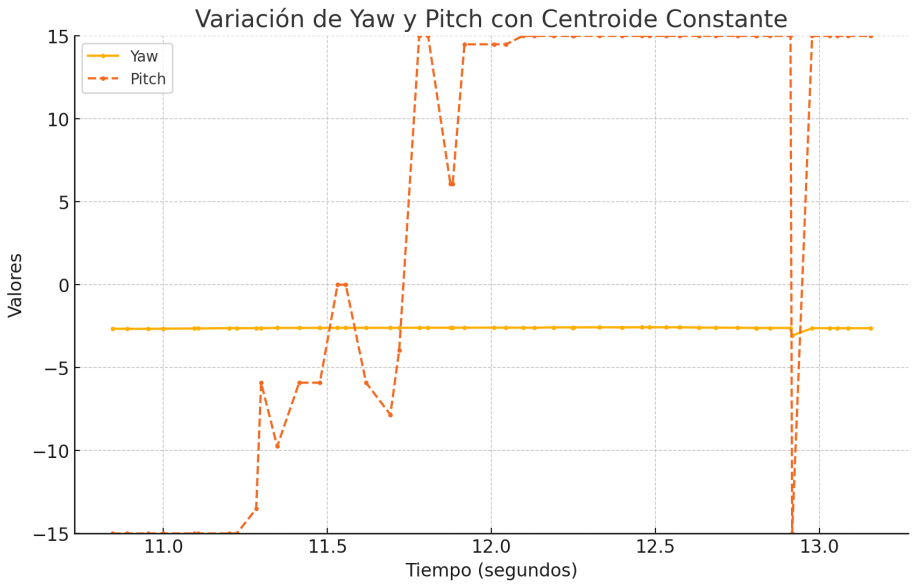
Demostración visual de variación del ancho (alejamiento)



Nota. Imagen capturada por el autor durante una prueba de acercamiento del rostro respecto al dron.

Figura 51

Variación del PID respecto a la posición del rostro en el tiempo (pitch)



Nota. Imagen generada por el autor para validación de respuesta.

CAPÍTULO 9

Conclusiones

- Se reinstalaron y validaron las herramientas desarrolladas en trabajos previos, evaluando exhaustivamente la aplicación móvil original. Durante este proceso, se identificaron y corrigieron errores relacionados con la configuración de Gradle y el SDK de DJI, mejorando la integración con dispositivos actualizados.

- Se logró recibir, recopilar y analizar los datos emitidos por la cámara del dron DJI Air 2S, optimizando la tasa de refresco del *stream* a través de una combinación de estrategias: reducción de resolución, procesamiento de *frames* en intervalos específicos y acceso directo a los datos visuales en formato RGB. Estos datos fueron procesados para la generación de referencias destinadas a la implementación autónoma y representados gráficamente, lo que permitió un análisis detallado del comportamiento del sistema y validó la interacción en tiempo real con el Mobile SDK de DJI.

- Se implementaron técnicas de control basadas en visión, utilizando la cámara del dron como fuente principal de datos. La integración de OpenCV permitió desarrollar un sistema de seguimiento visual de rostros, en el que un controlador PID ajustó dinámicamente los valores de *pitch* y *yaw* del dron. Si bien las pruebas se realizaron sin hélices, los resultados obtenidos evidencian su potencial para futuras implementaciones con las hélices instaladas, siempre que se configure cuidadosamente un entorno controlado y se ajusten con precisión los valores de control. Este avance representa un

paso significativo hacia la operación autónoma del dron.

CAPÍTULO 10

Recomendaciones

- Se recomienda explorar e implementar sistemas más estables para la detección e identificación de objetos, como *ArUco Markers*, *AprilTags* u otras técnicas similares, que ofrecen un seguimiento más preciso y estable en comparación con el *face detector*. Para ello, es fundamental profundizar en la investigación de librerías nativas y C++ en *Android Studio*.

- Evaluar el comportamiento del dron en vuelo real para validar el envío correcto de los valores PID a los rotores con sus helices instaladas. Este proceso debe iniciarse con valores muy bajos para minimizar riesgos y garantizar la estabilidad del sistema durante las pruebas iniciales.

- Incorporar protectores para las hélices del dron, que están disponibles comercialmente y son compatibles con el DJI Air 2S, para así reducir el riesgo de daño tanto al dron como a su entorno durante las pruebas de vuelo.

- Construir una jaula de seguridad en paralelo con los protectores de hélices. Esta jaula permitirá realizar pruebas de vuelo en un entorno controlado, protegiendo el dron y a los operadores mientras se realizan ajustes o validaciones del sistema.

- Considerar la migración a una versión más moderna de Android Studio para aprovechar las herramientas y funcionalidades avanzadas que ofrecen.

- Velar por la implementación de la depuración mediante conexión Wi-Fi en lugar de conexión por cable. Dado que el dron debe estar conectado directamente al control, utilizar depuración por Wi-Fi facilita el proceso de desarrollo.

CAPÍTULO 11

Referencias

[1] C. P. Montoya, "Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica," *Universidad del Valle de Guatemala*, 2021.

[2] C. R. S. González, "Integración del dron DJI Air 2S con el ecosistema Robotat empleando el mobile SDK del fabricante DJI.," *Universidad del Valle de Guatemala*, 2023.

[3] I. Sa, M. Kamel, M. Burri et al., "Build Your Own Visual-Inertial Drone: A Cost-Effective and Open-Source Autonomous Drone," *IEEE Robotics and Automation Magazine*, vol. 25, n.º 1, págs. 89-103, 2018. DOI: 10.1109/MRA.2017.2771326.

[4] A. Keipour, G. A. Pereira, R. Bonatti et al., "Visual Servoing Approach to Autonomous UAV Landing on a Moving Vehicle," *Sensors*, vol. 22, n.º 17, pág. 6549, 2022, © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/). DOI: 10.3390/s22176549. dirección: https://www.mdpi.com/journal/sensors.

[5] Dirección: https://developer.android.com/studio/intro?hl=es-419.

[6] Dirección: https://kotlinlang.org/docs/getting-started.html.

[7] K. Telli, O. Kraa, Y. Himeur et al., *A comprehensive review of recent research trends on Unmanned Aerial Vehicles (uavs)*, ago. de 2023. dirección: https://www.mdpi.com/2079-8954/11/8/400.

[8] DJI, 2021. dirección: https://dl.djicdn.com/downloads/DJI_Air_2S/DJI_Air_2S_User_Manual_v1.0_en1.pdf.

[9] T. V. S. N. Venna, S. Patel y T. Sobh, "Application of Image-Based Visual Servoing on Autonomous Drones," en *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, IEEE, Bridgeport, Connecticut, USA: IEEE, 2020, págs. 579-585. DOI: 10.1109/ICIEA48937.2020.9248384. dirección: https://ieeexplore.ieee.org/document/9248384.

[10] F. Chaumette y S. Hutchinson, "Visual servo control. I. Basic approaches," *IEEE Robotics and Automation Magazine*, vol. 13, págs. 82-90, 1998. DOI: https://ieeexplore.ieee.org/document/4015997.

[11] S. Hutchinson, G. Hager y P. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, n.º 5, págs. 651-670, 1996. DOI: 10.1109/70.538972.

[12] *OpenCV Official Site*, Accessed: 2024-11-18. dirección: https://opencv.org/.

[13] O. Team, *Development with OpenCV on Android*, Último acceso: 18 de noviembre de 2024, 2024. dirección: https://docs.opencv.org/4.x/d5/df8/tutorial_dev_with_OCV_on_Android.html.

[14] *Programming with OpenCV*, Accessed: 2024-11-18. dirección: https://docs.opencv.org/4.x/d1/dc5/tutorial_introduction_to_opencv.html.

[15] *Performance of OpenCV in Real-Time Applications*, Accessed: 2024-11-18. dirección: https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html.

[16] O. Team, *Object Detection with OpenCV*, Accessed: 2024-11-18, 2024. dirección: https://docs.opencv.org/4.x/d5/d54/tutorial_face_detection.html.

[17] O. Team, *Camera Calibration with OpenCV*, Accessed: 2024-11-18, 2024. dirección: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html.

[18] O. Team, *3D Processing with OpenCV*, Accessed: 2024-11-18, 2024. dirección: https://docs.opencv.org/4.x/d7/d53/tutorial_py_pose.html.

[19] S. Anju, *Integrating OpenCV with TensorFlow and PyTorch*, Accessed: 2024-11-18, 2024. dirección: https://medium.com/@sdranju/android-studio-step-by-step-guide-for-setting-up-opencv-sdk-4-9-on-android-740547f3260b.

[20] O. Team, *Cascade Classifier Training*, Disponible en la documentación oficial de OpenCV. Consultado el 24 de enero de 2025, 2023. dirección: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

[21] O. Team, *ArUco Marker Detection*, Accessed: 2025-01-24, 2024. dirección: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

[22] D. D. Team, *DJI FPV Sample Application*, Accessed: 2025-01-24, 2025. dirección: https://developer.dji.com/document/06724d27-23cf-4741-b128-fc17d2891981.

[23] D. D. Team, *DJICodecManager Documentation*, Accessed: 2024-11-27, 2024. dirección: https://developer.dji.com/api-reference/android-api/Components/CodecManager/DJICodecManager.html.

[24] O. Team, *Image Filtering Techniques*, Accessed: 2025-01-24, 2024. dirección: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html.

[25] D. Developer, *Mobile SDK Documentation*, Accessed: 2024-11-18, n.d.

CAPÍTULO 12

Anexos

En este capítulo se presenta el enlace directo al repositorio de GitHub que contiene todo el proyecto desarrollado. En dicho repositorio se incluyen el código fuente, los SDK implementados y cualquier otro recurso necesario para la reproducción o el análisis de este trabajo.

Repositorio del código del proyecto

El código del proyecto se encuentra disponible en el siguiente repositorio de GitHub:

`https://github.com/Wilder-Guerrero/TESIS_18561.git`