UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA

FACSISTEL

INGENIERÍA EN ELECTRÓNICA Y AUTOMATIZACIÓN

TRABAJO DE INTEGRACIÓN CURRICULAR

Implementación de un sistema de navegación basado en visión por computador para el seguimiento de trayectoria aplicado a un vehículo aéreo no tripulado

Douglas Darío Yagual Beltrán

Dirigido por:
Ing. Junior Rafael Figueroa Olmedo, M.Sc.

La Libertad - 2025

DEDICATORIA

Dedico este trabajo de titulación a mis padres, Nerlis Beltrán y Plinio Yagual, quienes con su ejemplo, esfuerzo y dedicación me han enseñado a ser una persona íntegra, guiándome con principios y valores que han sido la base de mi crecimiento.

A mis hermanos, Jessica y Carlos, por ser un apoyo constante, por sus sabios consejos y por motivarme a seguir adelante incluso en los momentos más difíciles. Su presencia ha sido fundamental durante mi desarrollo académico.

Extiendo esta dedicatoria a mis abuelos, en especial a mi abuela Flérida Rosales, cuyo amor, compañía y afecto sincero han sido una fuente constante de inspiración y fortaleza. Aunque ya no esté físicamente, su recuerdo permanece siempre presente en mi corazón y ha representado un pilar importante en mi formación personal.

Este logro también es de cada uno de ustedes, pues ha sido posible gracias al respaldo, cariño y confianza que siempre me han brindado. Espero, con todo mi corazón poder retribuirles algún día todo el amor y apoyo que depositaron en mí a lo largo de este valioso proceso.

Douglas Darío Yagual Beltrán

AGRADECIMIENTO

Agradezco, en primer lugar, a Dios, por guiarme en cada decisión a lo largo de mi carrera universitaria, por iluminar mi camino, bendecirme y permitirme estudiar esta profesión que me ha brindado valiosas experiencias y conocimientos fundamentales para mi formación como profesional.

A mi familia, por estar siempre presente en cada etapa de este recorrido. A mis padres, Nerlis Beltrán y Plinio Yagual, por su apoyo constante, sus consejos y por impulsarme a seguir adelante con determinación. A mis hermanos, Jessica y Carlos, por su respaldo incondicional, sin ustedes este logro no habría sido posible. Les estoy profundamente agradecido.

A mi novia, Melany Reyes, una persona muy especial que ha estado siempre a mi lado, brindándome ánimo y motivación, especialmente en los momentos más complejos. Gracias por tu compañía, por creer en mí y alentarme a culminar este proceso con perseverancia. Tu apoyo incondicional ha sido una parte fundamental para alcanzar este logro.

Extiendo también mi agradecimiento a mi amigo Nicolás Quirumbay, compañero significativo de esta etapa. Juntos compartimos momentos de alegría, desafíos y aprendizajes que marcaron mi vida universitaria. Valoro profundamente tu amistad y el compañerismo que demostraste en cada paso del camino.

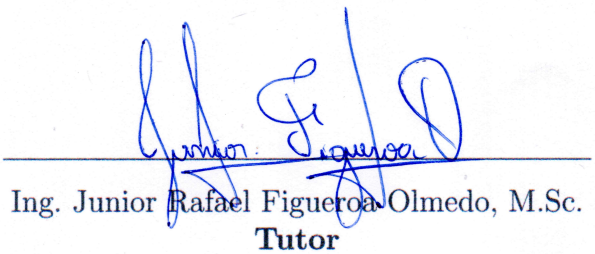
A mi tutor de tesis, Ing. Junior Figueroa, por su orientación, disposición y paciencia durante todo el proceso de desarrollo del proyecto. Gracias por compartir sus conocimientos y por acompañarme en este importante desafío académico.

Finalmente, a todos los ingenieros que formaron parte de mi formación académica, por transmitir sus saberes con vocación, por inspirarnos a superar obstáculos y por motivarnos a no rendirnos en la búsqueda de nuestros objetivos.

Douglas Darío Yagual Beltrán

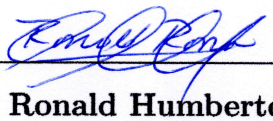
APROBACIÓN DEL TUTOR

En mi calidad de tutor del trabajo de titulación denominado: **"Implementación de un sistema de navegación basado en visión por computador para el seguimiento de trayectoria aplicado a un vehículo aéreo no tripulado"**, elaborado por el estudiante **Douglas Darío Yagual Beltrán**, de la carrera de **Ingeniería en Electrónica y Automatización** de la Universidad Estatal Península de Santa Elena, declaro que luego de haber orientado, estudiado y revisado, lo apruebo en todas sus partes y autorizo al estudiante que inicie los trámites legales correspondientes.

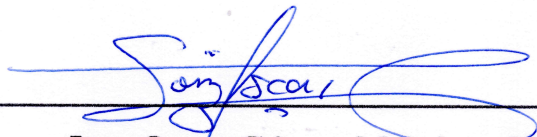


Ing. Junior Rafael Figueroa Olmedo, M.Sc.
Tutor

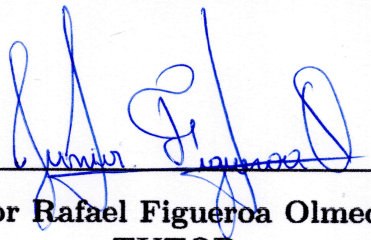
TRIBUNAL DE GRADO



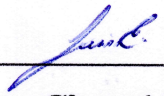
Ing. Ronald Humberto Rovira Jurado, Ph.D.
DIRECTOR DE CARRERA



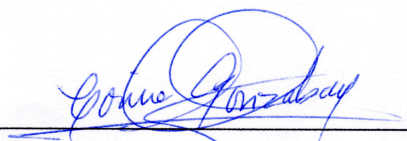
Ing. Oscar Gómez Morales , Mgtr.
DOCENTE ESPECIALISTA



Ing. Junior Rafael Figueroa Olmedo, Mgtr.
TUTOR



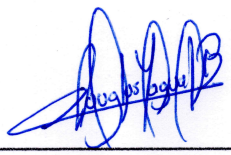
Ing. Luis Enrique Chuquimarca Jiménez, Mgtr.
DOCENTE GUÍA UIC



Ing. Corina Gonzabay De La A, Mgtr.
SECRETARIA DEL TRIBUNAL

DECLARACIÓN

Declaro que el contenido del presente trabajo de titulación es de mi absoluta responsabilidad, y que los derechos intelectuales del mismo pertenecen a la Universidad Estatal Península de Santa Elena.



Yagual Beltrán Douglas Darío
Autor

Resumen

El presente proyecto se basa en la implementación de un sistema de navegación basado en visión por computador para el seguimiento de trayectoria aplicado a un vehículo aéreo no tripulado. Se desarrolló un sistema capaz de interpretar trayectorias tanto en el plano horizontal como vertical, utilizando procesamiento de imágenes y algoritmos de seguimiento visual. El sistema fue implementado en el dron DJI Tello y desarrollado en el entorno de programación PyCharm. El procesamiento de imágenes se realizó utilizando la cámara integrada en el dron, aplicando técnicas de segmentación de imágenes en el espacio de color HSV, lo cual permitió una detección más precisa de las trayectorias. En el seguimiento horizontal, se empleó un algoritmo de línea convencional basado en sensores virtuales, el cual consiste en dividir la imagen procesada en tres regiones, a partir de las cuales se determinan acciones de traslación o rotación en el dron. En el seguimiento vertical, se diseñó un sistema lógico mediante máquinas de estados, utilizando también sensores virtuales distribuidos en franjas superior e inferior, permitiendo que el dron tome decisiones de movimiento según la zona y dirección de detección. Se realizaron varias pruebas experimentales para validar el funcionamiento del sistema en ambos planos, aplicando métricas específicas para evaluar el rendimiento del dron frente a diferentes trayectorias. Los resultados obtenidos evidencian un comportamiento funcional adecuado del sistema de navegación, demostrando la viabilidad de aplicar visión por computador como base para el control autónomo en drones.

Palabras clave: Navegación, dron, procesamiento, imágenes, detección, seguimiento de trayectoria.

Abstract

This project is based on the implementation of a computer vision-based navigation system for trajectory tracking applied to an unmanned aerial vehicle. A system capable of interpreting trajectories in both the horizontal and vertical planes was developed using image processing and visual tracking algorithms. The system was implemented on the DJI Tello drone and developed in the PyCharm programming environment. Image processing was performed using the drone's integrated camera, applying image segmentation techniques in the HSV color space, which allowed for more accurate trajectory detection. For horizontal tracking, a conventional line algorithm based on virtual sensors was used, which consists of dividing the processed image into three regions, from which translation or rotation actions are determined on the drone. For vertical tracking, a logical system was designed using state machines, also using virtual sensors distributed in upper and lower bands, allowing the drone to make movement decisions based on the detection area and direction. Several experimental tests were conducted to validate the system's operation in both planes, applying specific metrics to evaluate the drone's performance across different trajectories. The results obtained demonstrate adequate functional behavior of the navigation system, demonstrating the feasibility of applying computer vision as a basis for autonomous control in drones.

Keywords: Navigation, drone, processing, images, detection, trajectory tracking.

Índice

Índice de figuras

Índice de tablas

1. Introducción

En los últimos años, los vehículos aéreos no tripulados han ganado protagonismo en diversas áreas, desde aplicaciones recreativas hasta misiones industriales y de investigación. Dentro de este campo, el desarrollo de sistemas de navegación autónomos se ha convertido en una línea de trabajo importante para ampliar las capacidades operativas de estos dispositivos. En este contexto, el presente proyecto propone la implementación de un sistema de navegación autónoma basado en visión por computador, específicamente orientado al seguimiento de trayectoria en un entorno controlado, libre de interferencias ambientales. La propuesta consiste en desarrollar un sistema que permita a un dron detectar y seguir una trayectoria predeterminada, utilizando su cámara integrada como único dispositivo guía. Esta trayectoria podrá estar dispuesta tanto en el plano vertical, como una línea trazada sobre la pared y, en el plano horizontal, representada sobre el suelo. Para lograrlo, se emplearán técnicas de procesamiento de imágenes, detección de colores e identificación de contornos, desarrolladas principalmente en el lenguaje de programación Python. Desde un punto de vista técnico, este trabajo plantea una solución práctica para la navegación autónoma de drones en interiores. Además, al utilizar visión por computador, se evita el uso de sensores adicionales, lo que representa una ventaja en términos de peso, complejidad y costo. El enfoque metodológico incluirá la revisión de trabajos previos, el análisis comparativo de algoritmos aplicables en visión por computador y la evaluación experimental del rendimiento del sistema implementado. Con este proyecto, se busca no solo dar respuesta a una problemática tecnológica, sino también contribuir al desarrollo de soluciones accesibles y eficientes para la navegación autónoma de drones, lo cual puede tener múltiples aplicaciones en ámbitos como la inspección, la educación, vigilancia y la automatización de tareas en espacios reducidos.

1.1. Justificación

Con el pasar de los años la innovación tecnológica ha ido en aumento y esto ha generado un creciente interés entre los usuarios, especialmente en el ámbito de los drones. Ya que es interesante imaginar cómo en un dispositivo tan compacto es posible integrar una amplia variedad de circuitos electrónicos que permiten al dron despegar y ejecutar una variedad de movimientos según lo requiera el usuario. Este fenómeno ha captado el interés en la exploración aérea y el entretenimiento, abriendo nuevas posibilidades tanto en el ámbito recreativo como en aplicaciones profesionales. El uso de la inteligen-

cia artificial en un dron representa un avance significativo en la aplicación de tecnologías en los vehículos aéreos no tripulados, ya que permite alcanzar un nivel de precisión y funcionalidad que va más allá de los métodos tradicionales de control y navegación. Además, permite al dron tomar decisiones en tiempo real basadas en la información visual captada por la cámara, lo que optimiza la eficiencia operativa al seguir una línea predefinida de manera autónoma.

El desarrollo de sistemas inteligentes para vehículos aéreos no tripulados representa un avance importante tanto en el ámbito laboral como en aplicaciones en interiores, por tal motivo se propone el diseño e implementación de un sistema de navegación basado en visión por computador, enfocado en el seguimiento de trayectoria dentro de entornos estáticos y controlados. Esta propuesta busca aportar al crecimiento del conocimiento en el área de la robótica visual, con potencial de aplicación en sectores como mensajería, logística, agricultura de precisión, seguridad, vigilancia, entre otros.

El uso de visión por computador permite aprovechar la cámara integrada del dron como único sensor de navegación en espacios cerrados donde otros sistemas, como el GPS, resultan ineficaces. De esta manera, se promueve una solución accesible, adaptable y con gran potencial de escalabilidad. A partir de los beneficios mencionados, se evidencia la relevancia de implementar este tipo de sistemas para el control y guiado autónomo de drones. El enfoque del proyecto no solo permite explorar tecnologías emergentes, sino que también abre posibilidades para el desarrollo de soluciones prácticas en entornos reales donde la automatización de tareas puede marcar una diferencia significativa.

1.2. Panorama actual

Los inicios del dron se remontan al siglo XIX, específicamente en 1849, en donde se produjo un ataque a Venecia. Durante este evento, las fuerzas austrohúngaras, al no poder utilizar artillería directamente contra la ciudad, emplearon al menos 200 globos no tripulados cargados con 150 kilogramos de explosivos. En el año 1907 se creó el primer cuadricóptero por los hermanos Jacques y Louis Bréget, pero para poder estabilizarlo se necesitaba de cuatro hombres y solo lograba elevarse a dos pies de la superficie [1].

En la Segunda Guerra Mundial, los alemanes fueron los pioneros en la creación de la primera arma controlada a distancia. Consistía en una bomba de 2300 libras diseñada para hundir barcos, la cual sirvió como precursora de misiles antibuque y otras armas de precisión guiadas. Cerca de veinte años después, específicamente en 1960, se popularizaron los aviones de radio-

control, también conocidos como aviones RC. Estos modelos eran a escala que podían ser montados y controlados tanto en interiores como exteriores, destinados principalmente al entretenimiento, con una variedad de tamaños disponibles [1].

La idea de las aeronaves no tripuladas surge desde hace mucho tiempo atrás. Aunque se suele asociar a los drones con robots militares modernos, los aviones sin piloto han estados en uso de diversas formas durante décadas. En el año 1896, Samuel P. Langley desarrolló una serie de aeronaves a vapor que operaban sin piloto y lograron volar con éxito a lo largo del río Potomac, cerca de Washington. La práctica de vigilancia aérea surgió más tarde durante la Guerra Hispano-Americana de 1898, cuando las fuerzas militares montaron una cámara en un cometa, generando así una de las primeras fotografías de reconocimiento aéreo [2].

Los vehículos aéreos no tripulados poco a poco fueron formando parte en aplicaciones empresariales para facilitar la vida de personas como, por ejemplo: Wal-Mart, negocio de comercio minorista en el cual se registró recientemente el uso de drones dentro de las tiendas. La idea consiste en que cuando algo se acabe en las estanterías, la persona no tenga que buscar algún empleado que le ayude, sino más bien el UAV es encargado de realizar aquello [3].

En Amazon Prime – Air se está dando el uso de drones para la parte de mensajería, en los cuales se llevan años desarrollando este tipo de proyectos. El peso máximo que pueden soportar es de 2 kilos, sin embargo, la empresa no ha transportado tanto peso, sino más bien un 80 por ciento del mismo. Las pruebas que se han realizado han sido exitosas, y es muy útil ya que mediante aplicaciones para vuelos automáticos, el dron puede transportarse a zonas de difícil acceso de manera más rápida [3].

Durante los últimos años han sido fabricado drones modernos y de distintas marcas, existen modelos que pueden ser programados para aplicaciones que se vaya a requerir en algún momento. El lenguaje de programación más usado ha sido Python, que, gracias a un sinnúmero de investigaciones, se pudo conocer la facilidad de emplear algoritmos para el desarrollo de aplicaciones como la librería de código abierto OpenCV para Python, C++ y Java para seguimiento de trayectorias de UAV, videos de vigilancia, el seguimiento de objetos en movimiento, la detección de objetos, entre otros [4].

Uno de los métodos empleados para la detección y procesamiento de imágenes ha sido el uso de la visión por computador, la cual es una rama de la IA (Inteligencia Artificial), tiene como objetivo modelar matemáticamente

fases de proceso de percepción visual en los seres vivos y generar programas que permitan simular estas capacidades visuales por computadora. La visión artificial permite la asimilación de un mundo dinámico en tres dimensiones a partir de imágenes en dos dimensiones del mundo. Estas imágenes pueden ser de color blanco, negro u cualquier color. Se puede recibir por una o varias cámaras, estáticas o móviles. En la industria, la visión artificial abarca la informática, la óptica, la ingeniería mecánica y la automatización industrial [5].

A continuación, se detallan varios trabajos de titulación realizados y estudiados por diferentes autores en las cuales muestra un enfoque similar al problema planteado:

Según lo propuesto por los autores en [6], se desarrolló un sistema con visión por computador aplicada a la navegación y la detección de obstáculos de un dron experimental para la extinción de incendios. El presente proyecto utiliza el desarrollo de un sistema capaz de obtener de manera visual los ángulos propios de rotación del dron, a su vez el sistema será capaz de detectar obstáculos presentes en las imágenes capturadas. Para la obtención de los ángulos de rotación se incorporarán además de la cámara, dos sensores de unidad de medición inercial (IMU). El primero de ellos, modelo MPU-6050, contiene acelerómetro y giroscopio, con el que se obtendrán los ángulos de rotación. Además, se usará el sensor inercial DCM260B, ya que este posee magnetómetro, haciendo posible la obtención de la orientación del dron. El proyecto se basa en el lenguaje de programación C++, para la parte de detección de imágenes se ha usado algoritmos de la librería OpenCV. Esta propuesta de investigación tiene cierta similitud a la propuesta planteada en el proyecto, sin embargo, utiliza otro tipo de algoritmo de OpenCV y detección de obstáculos, pero se puede tomar como referencia para la solución de la problemática.

Los resultados obtenidos en la investigación presentada en [7], corresponden al diseño e implementación de un algoritmo de seguimiento de trayectorias para el minidrone Parrot Mambo. Esta propuesta tiene como objetivo incorporar dicho algoritmo al sistema de control de vuelo del dron, permitiendo vuelos autónomos. El desarrollo del algoritmo se plantea en tres fases: inicialmente, se procesan secuencias de imágenes mediante binarización, segmentación y conteo de píxeles; posteriormente, se diseña el algoritmo utilizando una máquina de estados para la adquisición y procesamiento de datos; finalmente, se valida su funcionamiento mediante Simulink en cuatro escenarios de prueba.

La investigación desarrollada en [8], propone un enfoque para el posicio-

namiento de un vehículo aéreo no tripulado (UAV) mediante visión artificial infrarroja. En este proyecto, se utiliza la librería de visión artificial OpenCV para facilitar el trabajo, gracias a sus funciones predefinidas. OpenCV está disponible para una variedad de lenguajes de programación, siendo C++ y Python los más comunes y utilizados en este proyecto. Aprovechando los detectores de características que OpenCV ofrece, se logró identificar puntos clave en cada imagen, en la cual la detección de patrones se consideró una opción viable debido a las limitaciones de tiempo y conocimientos. Se incluye también una librería específica para trabajar con patrones, conocidos como ARUCO. Estos patrones están especialmente diseñados para la localización en interiores y son ampliamente utilizados. No obstante, el objetivo final de este trabajo está orientado a los vehículos aéreos no tripulados (UAV), por ello se buscó una solución empírica que ampliase los ambientes en los que los patrones pudieran ser detectados. Esta solución se basa en el uso de la luz infrarroja, cuya longitud de onda se encuentra por encima de los 760 nm y puede ser fácilmente filtrada. Así, se pueden emplear filtros que permitan únicamente la visualización de la radiación infrarroja deseada.

En el contexto del desarrollo de tecnologías inteligentes para vehículos aéreos no tripulados, el trabajo presentado en [9], se basa en el estudio de algoritmos de visión artificial y aplicaciones para vehículos aéreos no tripulados. Este trabajo de investigación presenta una revisión completa de algoritmos de visión artificial y las aplicaciones inteligentes basadas en visión que se han podido desarrollar en el campo de los vehículos aéreos no tripulados (UAV) en la última década. En este tiempo, la evolución de tecnologías relevantes para los drones; como la miniaturización de componentes, el aumento de las capacidades computacionales y la evolución de técnicas de visión artificial han permitido un avance importante en el desarrollo de tecnologías y aplicaciones. Las técnicas de visión artificial permiten desarrollar tecnologías de vanguardia para hacer frente a las dificultades de percepción aérea; como algoritmos de navegación visual, detección y evitación de obstáculos y toma de decisiones aéreas. Este trabajo se basa en los avances más significativos capaces de resolver limitaciones técnicas fundamentales; como odometría visual, la detección de obstáculos, el mapeo y la localización, etc. Incorporando también el análisis en función de las capacidades y la utilidad potencial.

En el estudio desarrollado en [10], se presenta la implementación de un sistema automático de navegación basado en visión por computador aplicado a Mini Drones. El objetivo del presente proyecto tiene como base el conocimiento básico de un dron, los componentes que lo conforman, de igual manera como el aprendizaje y metodología de vuelo de este. Teniendo el

conocimiento clave sobre los drones se procede a la implementación de un sistema de navegación basado en técnicas de visión por computador, en el cual se hará uso de la cámara incorporada en el dron para aplicar modelos de seguimiento de rutas mediante librerías de OpenCV, el cual se basa en la detección de colores y procesamiento de imágenes.

En el trabajo desarrollado por el autor en [11], se detalla acerca del seguimiento de ruta para un dron. Este trabajo aborda la evolución tecnológica en los últimos años y como ha sido aprovechada para el beneficio de diversas áreas del quehacer humano, en particular el ámbito científico. La búsqueda constante de innovación ha impulsado el desarrollo de nuevas tecnologías, como el uso de drones y sus múltiples aplicaciones en la actualidad. En este contexto, el estudio se centra en la detección de rutas de vuelo mediante una cámara, permitiendo que un dron siga una trayectoria de forma autónoma. El proyecto desarrollado consiste en la implementación de un programa que capacite al dron para identificar y seguir una ruta preestablecida de manera autónoma. A lo largo del trabajo, se presenta una explicación detallada de todos los aspectos necesarios para la aplicación de esta tecnología, incluyendo los conceptos utilizados, las funciones y herramientas requeridas, las bases de su desarrollo, su funcionamiento y su aplicación tanto teórica como práctica en la sociedad. Todo esto se plantea desde una perspectiva funcional y acorde con los avances tecnológicos actuales.

1.3. Objetivos

1.3.1. Objetivo General

Implementar un sistema de navegación basado en visión por computador para el seguimiento de trayectoria aplicado a un vehículo aéreo no tripulado.

1.3.2. Objetivos Específicos

- Realizar la búsqueda de algoritmos que mejor se adapten para el desarrollo del sistema que se desea emplear.
- Analizar las diferentes tecnologías disponibles en drones con el fin de seleccionar un modelo que permita la programación personalizada por parte del usuario, basándose en la información proporcionada por los fabricantes.
- Investigar las metodologías y protocolos necesarios para programar el dron, utilizando la documentación técnica disponible, a fin de establecer una comunicación eficiente con el equipo.

- Implementar el código en el lenguaje de programación seleccionado para desarrollar el sistema de detección de colores y reconocimiento de la trayectoria para que el dron ejecute el seguimiento de manera autónoma.

1.4. Fundamentos teóricos

1.4.1. Robótica Aérea

La robótica aérea ha evolucionado utilizando principalmente plataformas como helicópteros, aviones, dirigibles y otros vehículos con capacidad de despegue y aterrizaje vertical. Estas estructuras destacan por su gran maniobrabilidad, una característica esencial para diversas aplicaciones robóticas. También existen modelos bioinspirados, como los drones que imitan el vuelo de aves o murciélagos, propulsándose mediante el batido de sus alas. Aunque la habilidad de mantenerse suspendidos en el aire resulta muy útil para múltiples tareas, estos sistemas suelen ser complejos de controlar y requieren personal capacitado para su manejo. No obstante, los robots aéreos ofrecen una solución práctica en numerosos campos, incluyendo la recolección de imágenes, identificación de objetivos, monitoreo de incendios, seguimiento, publicidad, cartografía, entre otros. [12].

1.4.2. Vehículos aéreos no tripulados

Los vehículos aéreos no tripulados se refieren a cualquier tipo de vehículo aéreo que no requiere un piloto a bordo y es operado a distancia por un controlador humano o mediante un software. Se los denomina UAV debido a sus siglas en inglés (Unmanned Aerial Vehicle). Este término es uno de los más utilizados cuando se habla de drones, ya que abarca una amplia variedad de estos dispositivos [13].

La denominación UAV es inclusiva y se aplica tanto a los vehículos aéreos no tripulados que pueden ser programados como a los RPA (Remotely Piloted Aircraft), que son aeronaves manejadas a distancia. Por tanto, UAV puede referirse tanto a drones autónomos como a aquellos dirigidos por un operador remoto. En la actualidad, el uso de drones ha transcendido su origen en el ámbito militar, donde se empleaban para misiones de reconocimiento y combate. Hoy en día estos dispositivos se han integrado con éxito en aplicaciones civiles, ya que los drones se utilizan en una variedad de campos y vienen equipados con cámaras y otros tipos de equipos especializados [13].

El tipo de dron que se va a usar es un cuadricóptero pequeño, el cual

tendrá características favorables de acuerdo con lo que se ha planteado como objetivos en el proyecto. Deberá poseer una estructura compacta (Figura 1) y una cámara adjunta que pueda grabar videos [14]. Uno de los parámetros que se debe tener en consideración es la capacidad de ser programable y la compatibilidad con el lenguaje de programación que se va a usar.

Figura 1: UAV (Vehículo aéreo no tripulado).
Fuente: [15]

1.4.3. Componentes de un cuadricóptero

La estructura de un dron está formada por un chasís mecánico que integra diversos sensores y actuadores. Los actuadores, equipados con hélices, permiten al dron despegar, mantenerse en el aire y maniobrar con estabilidad. Los sensores desempeñan un papel fundamental al proporcionar información en tiempo real sobre la altitud y otros parámetros esenciales para el control de vuelo. Además, los drones incorporan placas electrónicas en su circuito interno, las cuales regulan y optimizan su desempeño, garantizando un vuelo seguro y eficiente. A continuación, se describen los principales componentes que conforman un dron.

- **Chasís:** También se le conoce como la estructura principal que soporta todos los componentes del dron. En los modelos de mayor calidad, este elemento suele estar fabricado con fibra de carbono, gracias a su alta resistencia y bajo peso [16].

- **Motores:** Los motores representan el componente fundamental responsable de generar la propulsión que permite el vuelo del dron. Generalmente, se utilizan dos tipos principales: los motores con escobillas (Figura 2), que son más sencillos y económicos, y los motores sin escobillas (Figura 3), que , aunque más costosos, ofrecen una mayor eficiencia en la relación peso-potencia [17].

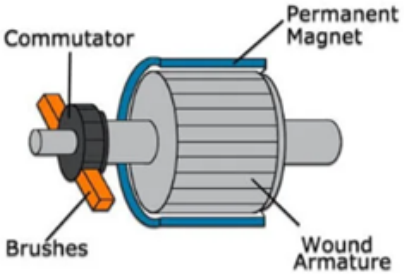


Figura 2: Motor con escobillas.

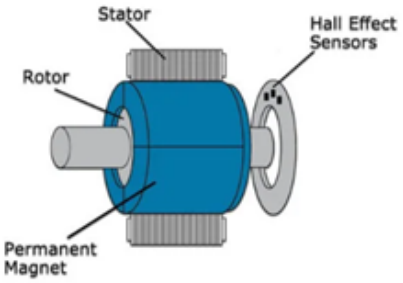


Figura 3: Motor sin escobillas.

Fuente: [17]

- **Hélices:** Las hélices son responsables de generar la fuerza de sustentación cuando los motores comienzan a girar. Existen dos tipos: unas giran en sentido horario y otras en sentido antihorario. Esto se debe a que, si todas rotaran en la misma dirección, el dron tendería a girar sobre su propio eje.

  Para mantener el equilibrio, se dispone la mitad de las hélices girando en un sentido y la otra mitad en el opuesto (figura 4). Además, las hélices pueden contar con varios pares de palas; mientras mayor sea el numero de palas, mayor será el empuje generado, aunque esto podría reducir la eficiencia [18].

Figura 4: Sentido de giro de las hélices de un dron.
Fuente: [18]

- **Controlador de motor eléctrico (ESC):** Este componente se encarga de regular la velocidad de los motores, transformando la señal de corriente continua (CC) proveniente de la batería en corriente alterna (AC). Dependiendo del diseño del dron, se pueden utilizar ESC individuales para cada motor (Figura 5) o una placa única que integre múltiples ESC en su estructura [19].

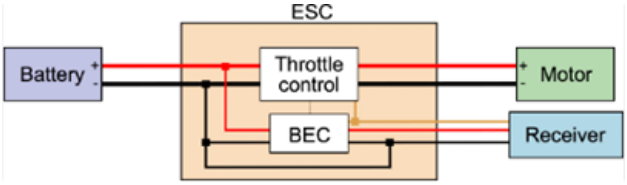


Figura 5: Controlador de motor eléctrico.
Fuente: [20]

- **Placa de distribución de energía (PDB):** Esta placa permite que la batería alimente todos los componentes internos del dron, incluyendo los motores [21].
- **Controlador de vuelo:** Es considerado el centro de control del sistema (Figura 6), encargado de determinar la velocidad adecuada para cada motor según los datos que recibe de los sensores y del receptor [22].

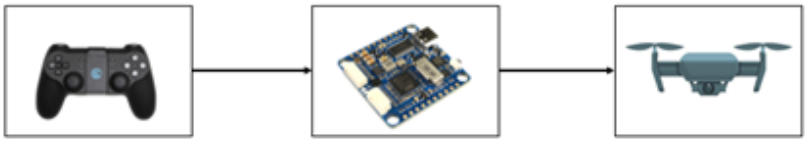


Figura 6: Controlador de vuelo
Fuente: Autoría propia

- **Batería:** Es el componente responsable de proporcionar energía a todos los elementos del dron, como los motores, la unidad de control de vuelo, los sensores y otros dispositivos adicionales como cámaras o sistemas de comunicación. La batería debe ser liviana y altamente eficiente, ya que un menor peso reduce la carga que el dron debe levantar, lo que a su vez disminuye el consumo energético durante el vuelo [23].
- **Receptor o Bluetooth:** Es el sistema encargado de transmitir las señales entre el control remoto y el dron. Frecuentemente, incluye también una cámara integrada (Figura 7) además de una tarjeta SD para almacenamiento o antenas que permiten la transmisión inalámbrica de video [24].

Figura 7: Visualización de video por medio de cámara del dron
Fuente: Autoría propia

- **Transmisor de video (VTX):** Para enviar una señal de video desde el dron, se utiliza un transmisor (Figura 8), el cual permite que el contenido visual sea emitido mediante la antena de un dispositivo móvil o un control remoto [25].

  Un VTX (transmisor de video) cumple con la función de enviar la señal captada por la cámara FPV del dron hacia un receptor VRX, ya sea autónomo o incorporado en unas gafas FPV, permitiendo así que la imagen transmitida pueda visualizarse en un monitor o directamente en las gafas [25].

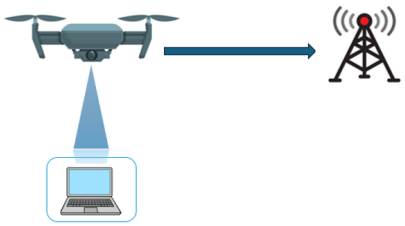


Figura 8: Transmisor de video
Fuente: Autoría propia

- **Sensores:** Los drones incorporan diversos tipos de sensores para mejorar su funcionamiento. Por ejemplo, el sensor de presión se utiliza para calcular la altitud o la distancia entre el dron y el suelo (Figura 9). El sistema GPS (Sistema de posicionamiento Global) permite determinar la ubicación exacta del dron, mientras que el IMU (Unidad de medición inercial) se encarga de registrar tanto la aceleración como la orientación del dron en el espacio [26].

Figura 9: Sensor de presión
Fuente: Autoría propia

1.4.4. Visión por computador

La visión por computador, también conocida como visión artificial, constituye un área clave dentro de la robótica y la informática, cuya evolución ha permitido el desarrollo de soluciones automatizadas más inteligentes, versátiles y eficaces. Esta disciplina utiliza diversas herramientas para que los dispositivos puedan capturar imágenes del entorno, analizarlas y extraer información significativa. Generalmente, trabaja en conjunto con sistemas de inteligencia artificial que aplican algoritmos matemáticos para interpretar dichas imágenes, reconociendo objetos, formas y patrones que sirven de base para realizar acciones específicas [27].

Como subcampo de la inteligencia artificial, la visión por computador se centra en el procesamiento, análisis de imágenes y videos, integrando técnicas que permiten a las máquinas observar su entorno y extraer información útil. Normalmente, estos sistemas constan de una cámara y un software especializado capaz de identificar y clasificar distintos elementos visuales. Son capaces de procesar diversos tipos de contenido, como imágenes, grabaciones o códigos, y pueden incluso identificar rostros humanos y detectar emociones [28]. A continuación, se muestran algunos ejemplos de aplicaciones dentro de este campo.

- **Fotogrametría**
  Se basa en obtener mediciones del espacio tridimensional a través de múltiples fotografías, lo cual facilita el análisis de superficies, el diseño de estructuras, objetos y otros elementos [29].

- **Rectificación métrica**
  Este método práctico permite corregir la perspectiva y mejorar las distorsiones provocadas por el lente (ver Figuras 10 y 11) [29].

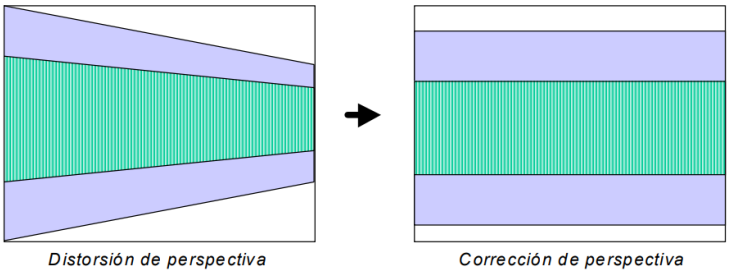


Figura 10: Ejemplo de corrección de perspectiva.
Fuente: [29]

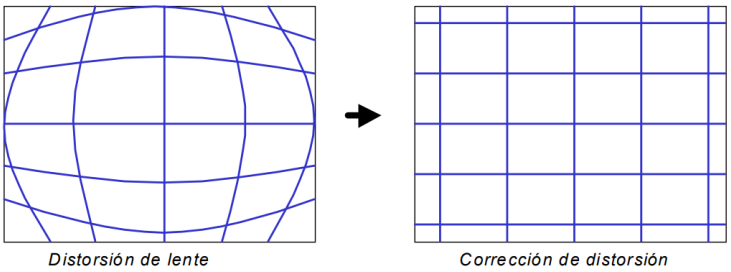


Figura 11: Ejemplo de corrección de distorsión de lente.
Fuente: [29]

- **Reconstrucción 3D**
  Una de las aplicaciones de la visualización es la generación de modelos tridimensionales mediante la técnica de triangulación. En la Figura 12 se ilustra este principio, donde, al identificar que los puntos A y B son proyecciones del mismo punto en el espacio 3D Q, es decir, puntos correspondientes, y al conocer las posiciones de los centros ópticos C_1 y C_2, es posible obtener la ubicación de Q a través de la intersección de las rectas $\langle C_1, A\rangle$ y $\langle C_2, B\rangle$ [29].

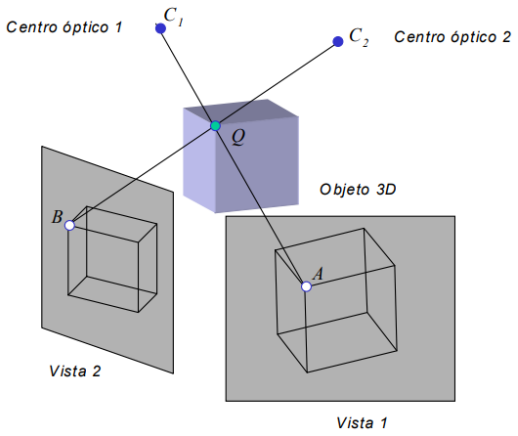


Figura 12: Triangulación: Estimación de Q a partir de A y B.
Fuente: [29]

- **Matching y Tracking**
  En esta sección práctica, mediante el uso de técnicas de Matching y Tracking, es posible identificar y analizar la correspondencia entre múltiples puntos en distintas imágenes. Estos puntos coincidentes representan la proyección de un mismo punto físico ubicado en el espacio tridimensional. La Figura 13 muestra tres vistas de un objeto capturado por una cámara al rotar el eje central de una taza. Al observar las imágenes, se puede interpretar que los puntos m_1, m_2 y m_3 en las imágenes 1, 2 y 3, respectivamente, corresponden entre sí, ya que todos son proyecciones del mismo punto m en la superficie de la taza [29].

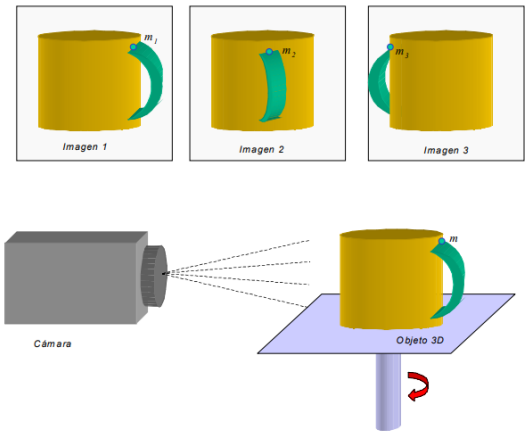


Figura 13: Correspondencia en tres puntos.
Fuente: [29]

- **Computación gráfica**
  La computación gráfica parte del principio de que, para generar un modelo de una imagen, como en el caso de una función: $f : 3D \rightarrow 2D$, es posible simular de forma gráfica las vistas en dos dimensiones que se derivan de un objeto tridimensional. Un ejemplo representativo de esta aplicación se encuentra en las tecnologías de realidad virtual [29].

- **Estimación de movimiento**
  Es posible calcular el desplazamiento de un objeto utilizando una serie de fotografías tomadas durante su movimiento, siempre que se identifiquen correctamente los puntos correspondientes entre las imágenes de la secuencia [29].

1.4.5. Modelación geométrica de un sistema de visión por computador

En esta sección se presenta un ejemplo del funcionamiento de un sistema de visión por computador y sus componentes (véase en la Figura 14). Los componentes de un sistema de visión por computador son: manipulador, la fuente de energía, el sensor, conversor análogo – digital y un computador.

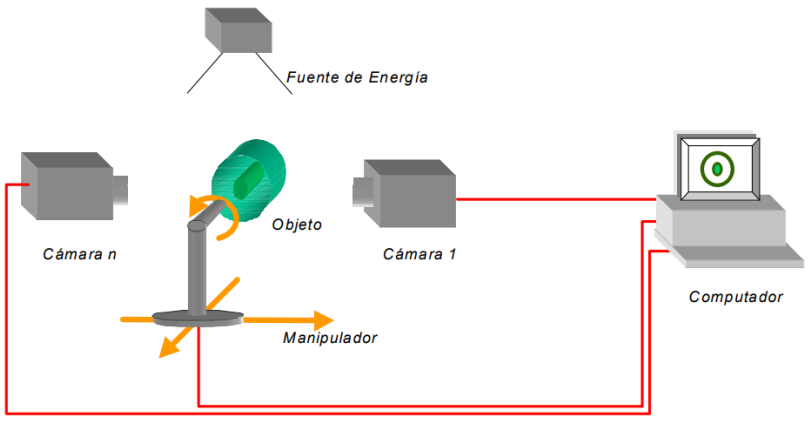


Figura 14: Sistema de visión por computador.
Fuente: [29]

- **El manipulador**
  El manipulador consiste en un dispositivo diseñado para mover y posicionar un objeto sin intervención directa del ser humano, su funcionamiento se basa en grados de libertad que indican los posibles movimientos que puede hacer para mover el objeto ya sea de traslación o

rotación. En muchos casos el manipulador es accionado mediante controladores como joysticks, otras veces por medio de una interfaz con un PLC o computador [29]. Para ejecutar los movimientos, el manipulador incorpora elementos deslizantes y de giro, permitiendo la traslación y rotación del objeto, respectivamente.

- **Fuente de energía**
  Según con el tipo de análisis que se quiere realizar sobre el objeto de estudio, es necesario seleccionar la fuente de energía adecuada para capturar la imagen. Entre las formas de energía más utilizadas se encuentran la luz visible para la fotografía, los rayos x y y para la radiografía y tomografía, el ultrasonido para la ecografía, los campos magnéticos para la resonancia magnética y el calor para la termografía, entre otros. En la mayoría de los casos, se emplea filtros para limitar el espectro de frecuencias de la energía utilizada, en el caso de la iluminación, es fundamental determinar si se requiere luz difusa o directa, así como su color o espectro [29].

- **Sensor de imagen**
  Este componente deber ser sensible a la energía empleada en el análisis. Por ejemplo, si se utiliza luz es fundamental contar con un elemento fotosensible que convierta los fotones reflejados por el objeto en una señal eléctrica. En el caso de los rayos x, debido a la baja sensibilidad de los sensores fotosensibles a estos fotones, se emplea un amplificador de imagen entre el objeto y el sensor, el cual transforma los rayos x en luz visible [29].

- **Conversor análogo – digital**
  También puede ser llamado como A/D, este componente tiene la función de transformar la señal eléctrica en código binario, permitiendo que el computador la interprete y genere una imagen digital del objeto en estudio [29].

- **Computador**
  El computador u ordenador tiene la función de procesar la información proporcionada por el conversor A/D. En un sistema de visión por computador, sus tareas principales incluyen el mejoramiento de imágenes, la segmentación, la clasificación de patrones y el análisis espacial [29].

1.4.5.1 Distorsión del lente

La curvatura del lente en las cámaras provoca una distorsión en la imagen, haciendo que las líneas rectas en el espacio tridimensional se proyecten como curvas. Este efecto suele ser insignificante en el centro de la imagen, pero se vuelve más notable en los bordes, donde la normal de la superficie del lente no es paralela al eje óptico de la proyección. Un ejemplo de este fenómeno se puede observar en la Figura 15 [30].

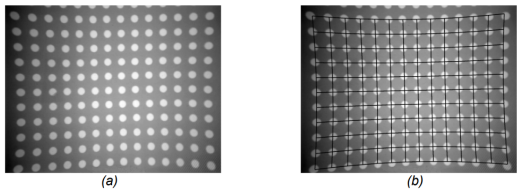


Figura 15: Ejemplo de distorsión: a) Radiografía de un objeto de calibración, b) Modelación de la distorsión.

Fuente: [29]

Cuando la distorsión de la imagen es significativa, el modelo lineal de la cámara CCD presentado en la sección anterior debe ajustarse. En la literatura se pueden encontrar diversos modelos que incorporan esta distorsión. La ide principal de estos enfoques es considerar una imagen ideal con coordenadas (x, y) y una imagen real con coordenadas (x', y'), siendo esta última la única observable [29]. Así, el modelo de proyección completo, en el cual un punto $M' = \begin{bmatrix} X' & Y' & Z' & 1 \end{bmatrix}^T$ se proyecta en la imagen como $\begin{bmatrix} u & v & 1 \end{bmatrix}^T$, se compone de cuatro elementos principales.

- **Transformación Euclídea 3D:** Mediante esta transformación, las coordenadas de un punto M' en relación con el objeto de estudio se convierte en las coordenadas del sistema de proyección, representadas como un punto M [29]. Esta transformación, se expresa como: $M = S'M'$.

- **Proyección en perspectiva:** A partir de esta proyección y observando que se obtiene a partir de un punto M un punto (x, y) en el plano de la imagen. Para recordar de mejor manera esta proyección está expresada en la ecuación: $\lambda m = PM$, con un $m = \begin{bmatrix} x & y & 1 \end{bmatrix}^T$. Esta imagen que se proyecta recibe el nombre de imagen ideal [29].

- **Modelación de distorsión:** Mediante una función de distorsión, es posible obtener la imagen real a partir de la imagen ideal. En otras palabras, a partir de un punto (x, y) en la imagen ideal, se genera

un punto (x', y') en la imagen real [29]. La función de distorsión se representa de la siguiente manera:

$$x' = x + \delta_x(x, y), \quad y' = y + \delta_y(x, y) \tag{2.1}$$

- **Proyección en la cámara:** Para realizar la formación de la imagen en la cámara CCD se puede efectuar mediante la matriz K [29]. Entonces para las nuevas coordenadas se tiene:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & s & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{2.2}$$

A continuación, se describe de manera general cómo es posible modelar la función de distorsión. Usualmente, esta distorsión se representa mediante dos componentes: una radial δ_r y otra tangencial δ_ϕ [31] Estas componentes se visualizan en la Figura 16, donde se muestra un punto sin distorsión (x_0, y_0), en el que $x = x_0$ y $y = y_0$, lo que implica que $\delta_x(x_0, y_0) = \delta_y(x_0, y_0) = 0$. Por lo general, este punto corresponde al punto principal de la imagen; sin embargo, esta condición no siempre se cumple, ya que el lente puede estar ligeramente desplazado con respecto al eje óptico de la proyección. Expresando (x, y) en coordenadas polares (r, ϕ) con origen en (x_0, y_0), la distorsión puede representarse como la suma de ambas componentes.

$$\begin{aligned} \delta_x(x, y) &= \delta_r(r)\cos(\phi) - r\delta_\phi(\phi)\sin(\phi) \\ \delta_y(x, y) &= \delta_r(r)\sin(\phi) + r\delta_\phi(\phi)\cos(\phi) \end{aligned} \tag{2.3}$$

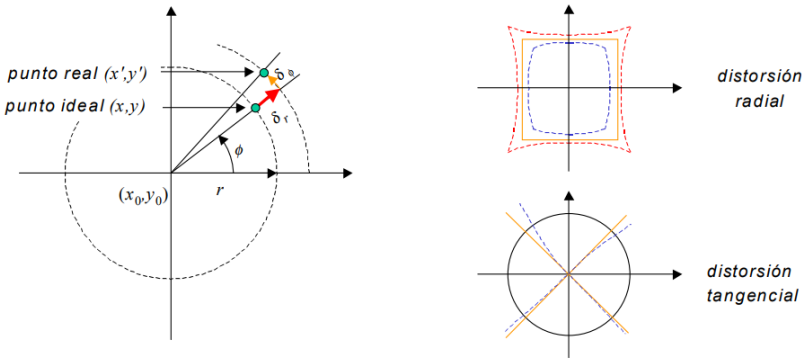


Figura 16: Modelación de distorsión de lente en componente radial y tangencial.

Fuente: [29]

La distorsión radial se basa en la premisa de que un punto ideal (x, y) se proyecta en la imagen real a lo largo de la línea radial que conecta los puntos (x_0, y_0) y (x, y). De manera similar, la distorsión tangencial supone que ocurre una variación en el ángulo δ_ϕ sin afectar el radio. Esto se muestra en la imagen anterior (Figura 16). Mientras que la distorsión radial depende únicamente del radio, la distorsión tangencial está determinada solo por el ángulo. Ambas pueden ser representadas mediante polinomios de grado superior a uno [29].

1.4.5.2 Calibración de un sistema de visión por computador

La calibración es el proceso mediante el cual se determinan lo parámetros intrínsecos y extrínsecos de la cámara, así como los parámetros del manipulador. Además, también permite estimar los parámetros asociados al modelo de distorsión del lente de la cámara [29].

Existen dos métodos ampliamente utilizados para la calibración: la auto calibración y la calibración fotogramétrica. En el caso de la auto calibración, se capturan múltiples imágenes de una misma escena y, a través de la correspondencia entre puntos en distintas imágenes, se pueden estimar los parámetros óptimos del modelo que mejor representen dicha correspondencia. Sin embargo, la reconstrucción 3D obtenida con este método presenta una limitación: está afectada por un factor de escala. Esto se debe a que no es posible conocer el tamaño real de los objetos capturados por la cámara. Por ejemplo, un objeto pequeño ubicado cerca del centro óptico puede generar la misma imagen que un objeto de mayor tamaño situado más cerca del plano de imagen [29].

Para aplicaciones que requieren una reconstrucción 3D precisa, como en el caso de la robótica, se recomienda utilizar la calibración fotogramétrica. Este método emplea un objeto de referencia tridimensional con una geometría perfectamente conocida. Se selecciona N puntos de interés en el objeto de referencia, obteniendo sus coordenadas como: $M_i' = \begin{bmatrix} X_i' & Y_i' & Z_i' & 1 \end{bmatrix}^T$, para $i = 1, \ldots, N$. Posteriormente, el objeto es capturado por la cámara y los puntos de interés se proyectan en una imagen 2D con coordenadas: $w_i = \begin{bmatrix} u_i & v_i & 1 \end{bmatrix}^T$. Con un modelo de proyección adecuado, es posible obtener una estimación teórica de los puntos en el espacio tridimensional lo que permite calcular con mayor precisión la reconstrucción 3D [29]. Los puntos se pueden calcular de la siguiente manera:

$$\hat{w}_i = f(M_i') \tag{2.4}$$

La función de proyección, denotada como f, incorpora los parámetros de la cámara, el lente y el manipulador, según corresponda. Matemáticamente, f es una función no lineal que depende de un vector de parámetros Θ el cual agrupa todos los parámetros del modelo [29]. En el caso de una proyección sin distorsión, se puede utilizar la ecuación (2.13) como función de proyección f. En este escenario, los once parámetros del modelo se organizan en el vector:

$$\Theta = \begin{bmatrix} t_x & , t_y & , t_z & , \omega_x & , \omega_y & , \omega_z & , \alpha_x' & , \alpha_y' & , u_0 & , v_0, s' \end{bmatrix}^T \tag{2.5}$$

La calibración se plantea entonces como un problema de optimización, donde el objetivo es minimizar una función que mide el error entre la proyección estimada $\hat{w}_i$ y la proyección real w_i. De este modo, la tarea consiste en ajustar los parámetros de la función de proyección f para reducir al mínimo la siguiente función:

$$J(\Theta) = \frac{1}{N} \sum_{i=1}^{N} \|\hat{w}_i - w_i\| \rightarrow \text{mín} \tag{2.6}$$

1.4.6. Procesamiento de imágenes

El procesamiento de información juega un papel importante en la ciencia y tecnología, especialmente en el campo de la ingeniería. La planificación, diseño, implementación y operación de procesos dependen en gran medida de la generación y análisis de datos para la toma de decisiones. Actualmente, el estudio de procesos y sistemas se basa cada vez más en el uso de sensores avanzados, capaces de recopilar datos precisos sobre las diversas variables involucradas en estos sistemas complejos [32].

1.4.6.1 Análisis de imágenes

Una imagen es una representación visual de un elemento concreto y abstracto, ya sea real o imaginario. Puede simular una figura, un objeto, una escena o un concepto perceptible a través de la vista [33]. Ejemplos de imágenes incluyen fotografías, ilustraciones, obras de arte, diseños gráficos y reflejos en un espejo. En este apartado es donde mediante la adquisición de una imagen se obtiene una decisión, interpretación o medición.

Una de las aplicaciones en el análisis de imágenes ha sido desarrolladas en procesos de control de inspección visual automática, en la cual a partir de la captura de una imagen se desea conocer si un producto se encuentra en mal estado. Otro de los ejemplos es en la industria de alimentos en donde se realiza un análisis de varias imágenes a color con el objetivo de detectar grados de calidad o alguna anormalidad en los productos alimenticios [34]. El análisis de una imagen consiste en cinco fases, las cuales son:

- **Adquisición de la imagen:** En esta fase se obtiene la imagen del objeto que se desea estudiar y dependiendo de la utilización, la imagen puede ser una termografía, radiografía o fotografía [34].
- **Preprocesamiento:** Se aplican varios filtros para poder eliminar el ruido de la imagen con el propósito de mejorar la calidad o se puede realizar el aumento de contraste [34].
- **Segmentación:** Se realiza la identificación del objeto de estudio dentro de la imagen capturada [34].
- **Medición (extracción de características):** En esta fase se realiza un análisis acerca de los atributos o puntos de interés que se requiere del objeto de estudio [34].
- **Interpretación (clasificación):** Se realiza la interpretación del objeto una vez obtenido los valores en la fase de medición [34].

1.4.6.2 Luz natural en el proceso de visión de una imagen

La luz visible es un fenómeno físico que activa la respuesta del sistema visual humano. Consiste en una forma de energía que se propaga mediante ondas electromagnéticas, cuya longitud de onda es una propiedad fundamental y puede variar desde fracciones de nanómetros hasta varios kilómetros [34].

Las ondas electromagnéticas presentan diferentes características según su longitud de onda, lo que permite clasificarlas en distintas categorías, como rayos gamma, rayos X, radiación ultravioleta, luz visible, radiación infrarroja, microondas y ondas de radio [34].

En este contexto, la luz puede representarse como una distribución espectral de energía $L(\lambda)$, donde (λ) simboliza la longitud de onda. La región del espectro electromagnético que el ojo humano puede percibir se encuentra entre los 350 nm y los 780 nm (Figura 17) [34].

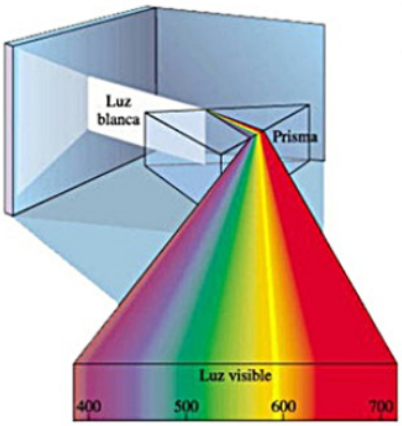


Figura 17: Espectro electromagnético. Longitud de onda en nanómetros. Fuente: [35]

La intensidad luminosa que se recibe de un objeto puede expresarse de la siguiente manera:

$$l(\lambda) = p(\lambda)L(\lambda) \tag{2.7}$$

En donde $p(\lambda)$ representa la capacidad de los objetos para reflejar o transmitir luz y $L(\lambda)$ corresponde a la distribución de la energía de la luz incidente. Al hablar de la percepción de una imagen, es fundamental considerar que esta se ve influenciada por el iluminante $(L(\lambda))$, las características del objeto $(p(\lambda))$ y el observador. Por esta razón, es útil analizar brevemente el sistema de visión humana, ya que sirve como referencia para la comparación con los sistemas de visión artificial [34].

La retina del ojo humano contiene dos tipos de fotorreceptores: los bastones y los conos. Los bastones son responsables de la visión escotópica, que permite la percepción en condiciones de muy baja iluminación. Por otro lado, los conos facilitan la visión fotópica, la cual abarca un rango de iluminación de cinco a seis órdenes de magnitud. En niveles intermedios de luz, ambos

tipos de fotorreceptores actúan en conjunto, dando lugar a la visión mesópica. Dado que las imágenes mostradas en pantallas electrónicas requieren una iluminación adecuada, el análisis suele centrarse en los conos, que además son los encargados de la percepción del color. Estos fotorreceptores, que suman aproximadamente 6.5 millones, se encuentran densamente agrupados en el centro de la retina, en la región conocida como fóvea, con una densidad cercana a 120 conos por grado de arco en el campo visual, lo que equivale a una separación de aproximadamente 2 micras. Fuera del área centrar de un grado de radio desde la fóvea, la densidad de los conos disminuye considerablemente. La pupila del ojo funciona como una apertura variable: en condiciones de alta iluminación, su diámetro es de aproximadamente 2 mm, actuando como un filtro pasa-bajo para los tonos verdes; en situaciones de poca luz, se expande hasta un valor de 8 mm, funcionando como un filtro pasa-banda con una capacidad de aproximadamente 60 ciclos por grado [34]. En la siguiente Figura 18 se ilustra la estructura del ojo humano, facilitando la comprensión de su configuración.

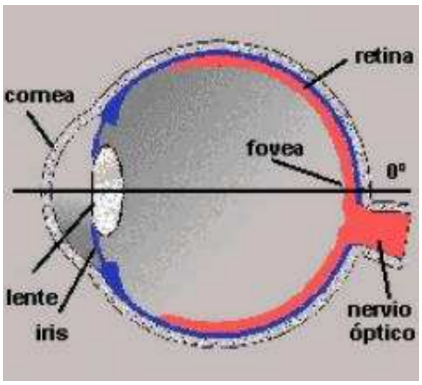


Figura 18: Esquema del ojo humano.
Fuente: [34]

La luminancia o intensidad de un objeto con una distribución espacial de luz $l(x, y, \lambda)$ se expresa de la siguiente manera:

$$f(x, y) = \int l(x, y, \lambda) V(\lambda)\, d\lambda \tag{2.8}$$

La función $V(\lambda)$ representa la eficiencia relativa luminosa del sistema visual. En el caso del ojo humano, esta función tiene una forma similar a una campa de Gauss, como se muestra en la Figura 19, donde sus características varían según se trate de visión escotópica o fotópica. La luminancia de un objeto es una propiedad que no se ve afectada por la luminancia de los objetos circundantes. Sin embargo, el brillo percibido de un objeto esta determinado

por su luminancia en relación con la de su entorno. Como resultado, dos objetos con entornos distintos pueden presentar la misma luminancia, pero ser percibidos con diferente brillo [34].

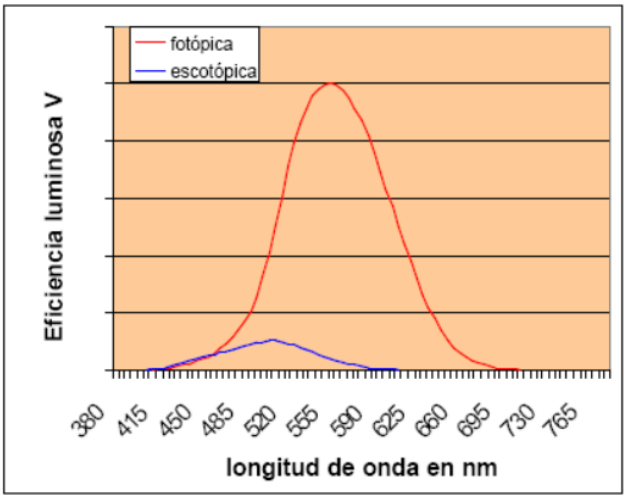


Figura 19: Sensibilidad del ojo humano.
Fuente: [34]

En el apartado de lo que corresponde a imágenes, es fundamental comprender conceptos claves que permiten un mejor entendimiento de los elementos que conforman una imagen. A continuación, se presentan los siguientes conceptos:

1.4.6.3 Píxeles

El píxel, abreviatura de "picture element", es la unidad fundamental de una imagen. Se define como el elemento homogéneo más pequeño en términos de color que la conforma y su nombre surge de la combinación de estas dos palabras en inglés [36].

- **Relaciones entre píxeles**
  Un píxel p ubicado en las coordenadas (x, y) posee cuatro vecinos en las direcciones horizontal y vertical, cuyas coordenadas son: $(x+1, y), (x-1, y), (x, y-1), (x, y+1)$. Este conjunto de píxeles se conoce como vecindad 4 o vecinos 4 de p y se representa como $N_4(p)$, como se muestra en la Figura 20. Cabe destacar que cada uno de estos píxeles se encuentran a una distancia de 1 con respecto a p y que, en los bordes de la imagen algunos de estos vecinos pueden quedar fuera de los límites [36].

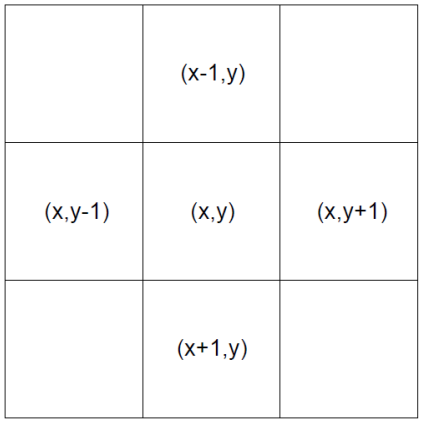


Figura 20: Vecindad $N_4(p)$.
Fuente: [36]

Además, existen cuatro vecinos diagonales de p, cuyas coordenadas son $(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)$ y se representan como $N_D(p)$, según la Figura 21. La combinación de $N_4(p)$ y $N_D(p)$ conforma la vecindad 8 denotada por $N_8(p)$ [36].

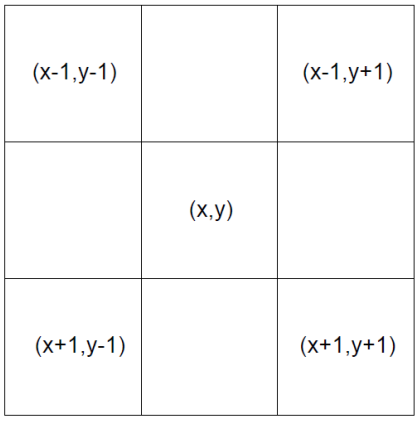


Figura 21: Vecindad $N_D(p)$.
Fuente: [36]

1.4.6.4 Iluminación

Cuando la iluminación en un entorno no está adecuadamente regulada, la calidad de la imagen puede verse afectada, generando un bajo contraste. Esto puede manifestarse en forma de reflejos especulares, sombras o destellos que perjudican la claridad de la imagen. En cambio, una iluminación

bien gestionada proporciona una luz óptima a la escena, permitiendo obtener una imagen con suficiente información para su correcto procesamiento y manipulación [37].

La iluminación juega un papel fundamental en la calidad de una imagen, ya que permite diferenciar, interpretar y reconocer con precisión los detalles relevantes según el propósito de su uso. Un aspecto para considerar es la variación entre la iluminación directa y la iluminación difusa, así como el contraste que se genera cuando una de ellas predomina sobre la otra. Cuando la iluminación es mayormente directa, la forma en que la luz se refleja en los objetos no es homogénea, ya que depende del ángulo entre el punto de vista y la superficie del objeto. Esto provoca que la iluminación no se distribuya de manera uniforme en toda la imagen, como se visualiza en la Figura 22. En contraste, cuando predomina la iluminación difusa, las superficies reflejan la luz en todas las direcciones de manera equitativa, lo que da como resultado una iluminación uniforme en la imagen [38].

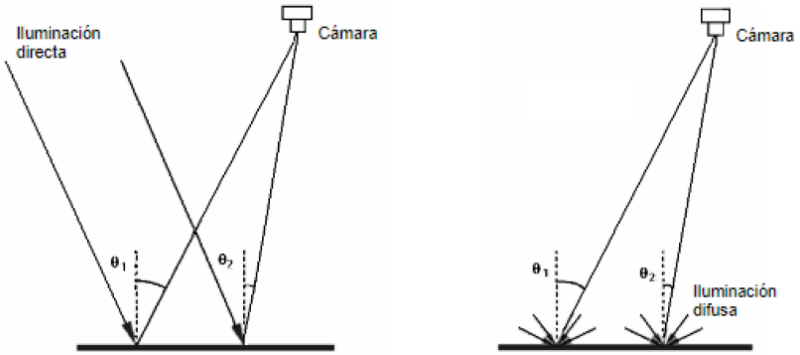


Figura 22: (a) Representación de la escena con luz directa. (b)Representación de la escena con luz difusa.

Fuente: [38]

Las imágenes con una alta incidencia de luz directa carecen de una iluminación uniforme, lo que afecta la claridad en los detalles. En la Figura 22 (a), se puede notar la presencia de sombras y un brillo excesivo en el centro de la imagen, lo que dificulta su apreciación. Debido a esto, es necesario ajustar la fuente de iluminación para lograr una distribución más homogénea o difusa en toda la imagen, como se observa en la Figura 22 (b). La comparación en la Figura 23 permite visualizar la diferencia entre una imagen con iluminación directa y otra con iluminación difusa [38].

(a) (b)

Figura 23: Diferencia de iluminación en la misma imagen. (a) Imagen con iluminación directa. (b) Imagen con iluminación difusa.
Fuente: [38]

En el proceso de la iluminación, existen diversos conceptos clave que influyen en la correcta visualización de una imagen. Estos incluyen ajustes como la intensidad del color, la saturación y el brillo, los cuales permiten mejorar la calidad y claridad de la imagen. Entre los aspectos mas relevantes en este proceso destacan el histograma, la umbralización y el frame, herramientas fundamentales para la optimización y el análisis de las imágenes. A continuación, se detallan de manera general estos conceptos.

- **Histograma:** Un histograma permite visualizar la distribución del brillo en una imagen, proporcionando información detallada sobre sus características luminosas, lo que facilita el análisis y procesamiento. Cabe destacar que, al capturar una imagen con cualquier cámara digital, factores como la cantidad de luz y el ángulo de toma influyen con cualquier cámara digital, factores como la cantidad de luz y el ángulo de toma influyen directamente en su luminosidad y contraste [39].

- **Umbralización:** La umbralización de una imagen es una técnica de segmentación que consiste en determinar un umbral adecuado para diferenciar objetos específicos dentro de la imagen. Este método permite clasificar los píxeles en una escala de grises o en dos categorías (blanco y negro), también conocidas como puntos de objeto y puntos de fondo, utilizando un umbral de intensidad. Además, el umbral puede definirse como el punto donde el histograma de la imagen se divide en dos picos [40].

- **Frame:** También conocido como framerate, es la velocidad a la cual un dispositivo muestra imágenes o fotogramas. La exposición y sucesión de estas imágenes da como resultado el vídeo. El ojo humano es capaz de distinguir aproximadamente 20 imágenes por segundo. De esta forma, al mostrarse más de 20 imágenes por segundo, es posible engañar al ojo y crear ilusión de movimiento. El valor FPS oscila entre 15 y 30: en

Europa se utilizan 25 imágenes por segundo (sistema PAL); en América y Asia 30 fps (sistema NTSC); y en cine por lo general 24 fps [41].

Se puede utilizar cualquier FPS para todos nuestros vídeos, pero hemos de tener en cuenta que, a menor número de fotogramas, notaremos que el vídeo va a 'tirones'. Un elevado número de FPS, supondrá un mayor tamaño y ocupación del archivo correspondiente al vídeo [41].

1.4.7. Modelos de color

El color es un fenómeno óptico en el que intervienen procesos fisiológicos, físicos y químicos [42]. La amplia variedad de colores que percibimos se debe a la combinación de los tres colores primarios: rojo, verde y azul (RGB, por sus siglas en inglés).

Cualquier característica visual de un objeto que pueda ser observada y cuantificada depende tanto del color de la luminosidad. Esto se debe a que la percepción de los límites y formas de los objetos está determinada por la capacidad del ojo humano para diferencias entre distintas zonas de luz y colores presentes en ellas [42].

Un modelo de color, también conocido como espacio de color en diversas fuentes, es una representación matemática que define un conjunto de colores. Estos modelos pueden clasificarse en cuatro categorías principales: Modelos de color básicos (RGB, RGB Normalizado), modelos de color perceptuales (HSV, HSI), modelos de color ortogonales (YCbCr, YUV, YIQ) y modelos de color para impresión (CMY, CMYK) [43]. En este apartado se explicará de manera general los modelos mencionados.

1.4.7.1 RGB

Este concepto se basa en la combinación de tres colores primarios con diferentes niveles de luminancia cromática: rojo, verde y azul (Red, Green, Blue - RGB). Para generar un color específico, es necesario determinar la proporción exacta de cada uno de estos componentes y mezclarlos adecuadamente [44]. Una de las representaciones más habituales en la presentada en la Figura 24.

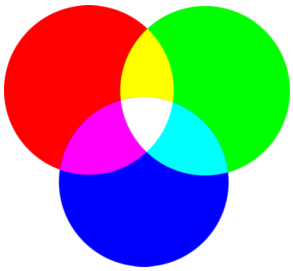


Figura 24: Modelo de color RGB.
Fuente: [45]

Este modelo se fundamenta en el sistema de coordenadas cartesianas. El subespacio de color relevante se representa como un cubo, como se muestra en la Figura 25, donde los colores rojo, verde y azul ocupan tres de sus vértices, mientras que cian, magenta y amarillo están ubicados en otros tres. El color negro se encuentra en el origen del sistema, y el blanco en la esquina opuesta. En este esquema, la escala de grises se extiende desde el negro hasta el blanco a lo largo de la línea que une estos dos puntos [46].

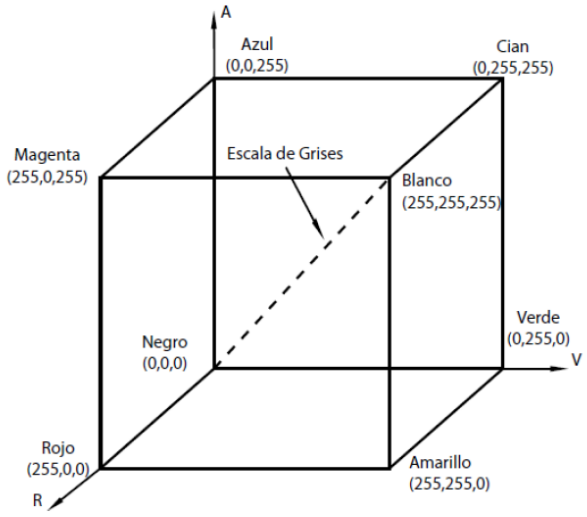


Figura 25: Representación de modelo de color RGB.
Fuente: [45]

Para obtener un color en el modelo RGB, se describe indicando el valor de acuerdo con la componente roja, verde y azul, el cual puede modificarse desde un valor de 0 a un valor máximo según la aplicación que se requiera. En la parte computacional o de visión los valores de estas componentes pueden almacenarse como números enteros en el rango de (0-255) [45].

1.4.7.2 RGB Normalizado

El modelo RGB normalizado es una forma de representación que se obtiene fácilmente a partir de los valores RGB mediante un proceso de normalización, como se indica en la ecuación (2.23) [45]. Este método se utiliza para minimizar los efectos provocados por variaciones en la iluminación de una imagen.

$$r = \frac{R}{R+G+B}, \quad g = \frac{G}{R+G+B}, \quad b = \frac{B}{R+G+B} \tag{2.9}$$

Así, la suma de las tres componentes normalizadas debe ser $r + g + b = 1$ [43].

1.4.7.3 HSV

El modelo HSV representa el color a través de sus componentes esenciales y se considera una forma más intuitiva y cercana a cómo los seres humanos perciben y describen los colores en comparación con el modelo RGB. Su nombre proviene de las siglas en inglés Hue (Tonalidad), Saturation (Saturación) y Value (Valor), lo que refleja su enfoque en la manera en que la luz y la intensidad afectan la percepción del color [45]. Se trata de una transformación no lineal del modelo de color RGB, en la que los colores resultantes son una combinación de tres componentes: tonalidad, saturación y valor. Estos componentes pueden representarse de la forma que se muestra en la Figura 26.

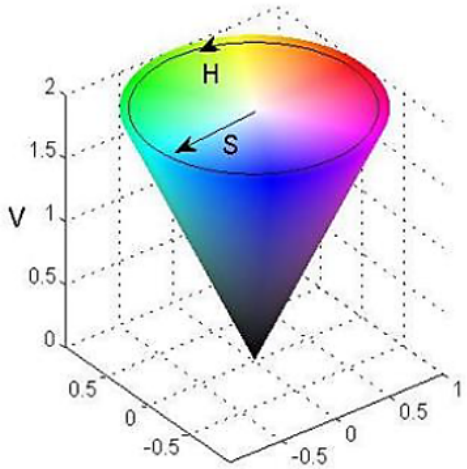


Figura 26: Modelo de color HSV.
Fuente: [11]

Para seleccionar un color en el modelo HSV, primero se debe elegir el tono (H) en una escala circular que abarca valores de 0° y 360°, donde cada valor se le atribuye un color específico. A continuación, se ajusta la saturación (S), determinando la intensidad o pureza del color. Finalmente, se selecciona el valor (V) en el eje vertical, el cual define la luminosidad o brillo del color [11].

- Tonalidad (HUE): El tipo de color (verde, azul, amarillo) que puede representarse mediante un ángulo en un rango de (0°-360°) o alternativamente, como un valor normalizado de 0 % a 100 % [11].
- Saturación (Saturation): La saturación del color se representa con un valor entre 0 y 1, donde 0 indica ausencia de saturación (un tono completamente gris) y 1 corresponde al tono en su máxima intensidad [10].
- Valor (Value): El nivel de brillo se representa con un valor entre 0 y 1, donde 0 indica el negro; y 1 indica el blanco. Alternativamente, también puede expresarse en un rango de 0 a 255 dentro del espectro de colores) [10].

Para realizar una conversión del modelo de color RGB a HSV se utilizan las siguientes expresiones:

$$H = \arccos\left(\frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right) \tag{2.10}$$

$$S = 1 - 3\frac{\text{mín}(R,G,B)}{R+G+B} \tag{2.11}$$

$$V = \frac{1}{3}(R+G+B) \tag{2.12}$$

1.4.7.4 HSI

El espacio de color HSI se basa en la percepción humana de color, diferenciándolo según tres atributos: Tono (Hue), Saturación (Saturation) e Intensidad (Intensity). Estos factores son fundamentales, ya que permiten segmentar imágenes de manera más efectiva en función del tono. Existen variaciones en este espacio de color, entre ellos, el modelo HSV, modelo el cual interesa trabajar en este proyecto. Esto facilita una mejor separación y diferenciación de los colores en la imagen [44].

El modelo de color HSI se define dentro de un sistema de coordenadas cilíndricas y suele representarse como un doble cono, como se observa en la

Figura 27. Cada color se encuentra ubicado en un punto dentro o sobre la superficie de este doble cono. La posición en altura del punto determina la intensidad. Por otro lado, el matiz (H) se expresa en términos de un ángulo en grados, indicando su orientación [47].

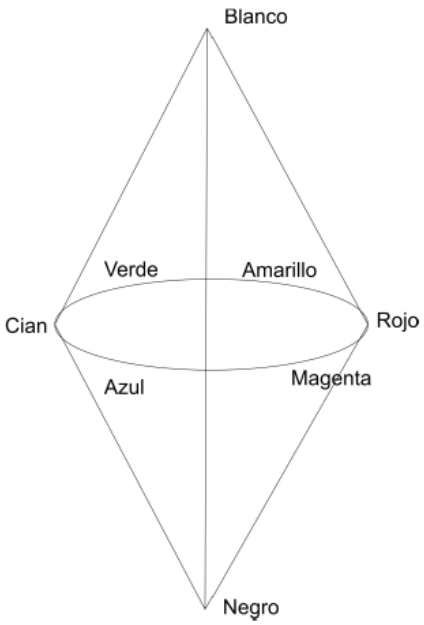


Figura 27: Modelo de color HSI.
Fuente: [45]

Para realizar una conversión del modelo RGB al HSI, se usan las siguientes expresiones:

$$H = \begin{cases} \theta, & \text{si } B \leq G \\ 360^\circ - \theta, & \text{si } B > G \end{cases} \tag{2.13}$$

$$\theta = \cos^{-1}\left(\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{\frac{1}{2}}}\right) \tag{2.14}$$

$$S = 1 - 3 \cdot \frac{\text{mín}(R,G,B)}{R+G+B} \tag{2.15}$$

$$I = \frac{1}{3}(R+G+B) \tag{2.16}$$

1.4.8. Lenguajes de programación para cuadricópteros

Un lenguaje de programación se basa en un grupo de instrucciones que se usan para establecer una comunicación con las computadoras. Se trata del

medio por el cual los desarrolladores escriben código para crear programas y aplicaciones, además los lenguajes pueden variar grandemente en cuanto a los propósitos y complejidad según se requiera [48]. Los tipos de lenguajes de programación se basan de acuerdo con varios criterios:

- Nivel de abstracción (alto o bajo nivel).
- Paradigma (imperativo, declarativo, orientado a objetos, funcional).
- Tipado (estático o dinámico).
- Dominio de aplicación (general o específico).

Los más usados son los siguientes:

1.4.8.1 Matlab

Es una herramienta de programación y cálculo numérico empleada por millones de ingenieros y científicos para el análisis de datos, la elaboración de algoritmos y la construcción de modelos. Matlab integra un entorno de trabajo diseñado para facilitar el análisis iterativo y los procesos de diseño, junto con un lenguaje de programación que permite realizar operaciones matemáticas con matrices y arreglos de manera directa [49].

Figura 28: Software Matlab.
Fuente: [50]

1.4.8.2 Scratch

Es un lenguaje de programación visual desarrollado por el Instituto de Tecnología de Massachusetts, diseñado para simplificar la creación de proyectos interactivos, como juegos, historias y animaciones, de manera intuitiva y entretenida. Su característica más distintiva es una interfaz gráfica basada en bloques de código que se encajan como piezas de un rompecabezas, evitando la necesidad de escribir en lenguajes de programación más complejos. Esto lo hace una opción ideal para principiantes y niños interesados en iniciarse en el mundo de la programación [51].

Figura 29: Software Scratch.
Fuente: [52]

1.4.8.3 Python

Se trata de un lenguaje de programación de alto nivel que sigue el paradigma de la orientación de objetos y cuenta con una semántica dinámica integrada. Ha sido diseñado principalmente para la creación de software y el desarrollo de aplicaciones web. Su simplicidad lo hace una alternativa fácil de aprender, ya que posee una sintaxis clara y centrada en la legibilidad. Gracias a esto, los desarrolladores pueden comprender e interpretar el código en Python con mayor facilidad en comparación con otros lenguajes [53].

Figura 30: Software Python.
Fuente: [54]

Teniendo en cuenta el lenguaje de programación Python, que es el elegido para desarrollar este proyecto, el software comúnmente emplea las siguientes librerías para la elaboración de sistemas de visión por computador.

- **OpenCV (Open source computer vision library):** Se trata de una biblioteca fundamental para la visión artificial ya que ofrece una gran variedad de algoritmos y funciones que simplifican tareas como el procesamiento de imágenes y videos, la detección de objetos y la extracción de características, entre otras. Su interfaz intuitiva, su amplia documentación y su compatibilidad con múltiples plataformas la hacen una elección ideal tanto para principiantes como para profesionales en el área [55].

- **Dlib:** Dlib es una biblioteca flexible y poderosa, especializada en la detección de rostros, el reconocimiento de puntos claves faciales, la alineación de imágenes y otras funciones relacionadas. Proporciona modelos

y herramientas entrenadas para distintas aplicaciones de aprendizaje automático, lo que la hace una opción clave para proyectos de vision por computador que necesitan un análisis facial preciso [55].

- **Pillow:** Es una biblioteca avanzada para el procesamiento de imágenes. Es compatible con múltiples formatos y ofrece diversas funciones, como redimensionar, recortar, aplicar filtros y añadir texto a las imágenes. Ya sea para trabajar con fotografías o crear contenido visual, esta librería proporciona una amplia gama de herramientas para manipular imágenes de manera segura [55].

- **Scikit-image:** Es una biblioteca fácil de usar diseñada para el procesamiento de imágenes y la vision artificial. Proporciona una variedad de algoritmos para tareas como la segmentación, la extracción de características y las transformaciones morfológicas. Gracias a esta librería es posible realizar modificaciones avanzadas en imágenes sin necesidad de un conocimiento profundo de matemáticas complejas [55].

- **TensorFlow and keras:** Son bibliotecas populares en el ámbito del aprendizaje automático que también destacan en aplicaciones de vision artificial. TensorFlow incluye modelos pre-entrenados, como Inception y ResNet, ideales para la clasificación de imágenes, mientras que Keras facilita la construcción, entrenamiento y evaluación de modelos de deep learning de manera intuitiva [55].

- **PyTorch and torchvision:** Al igual que TensorFlow y Keras, estas bibliotecas proporcionan herramientas avanzadas para la vision artificial. PyTorch destaca por su gráfico de cálculo dinámico, mientras que Torchvision ofrece conjunto de datos y modelos entrenados que simplifican la implementación de tareas como la clasificación de imágenes, la detección de objetos y la transferencia de estilos [55].

- **Matplotlib:** Es una excelente alternativa como biblioteca para el procesamiento de imágenes. Resulta particularmente útil como modulo en Python para manipular imágenes, ya que ofrece dos métodos específicos para su lectura y visualización. Esta biblioteca se enfoca en la representación gráfica de datos en 2D, funcionando como una herramienta multiplataforma que trabaja con matrices de Numpy. Aunque su uso principal está orientado a visualizaciones en dos dimensiones, como gráficos de barras, histogramas y diagramas de dispersión, también ha demostrado ser efectiva en el procesamiento de imágenes al permitir la extracción de información de manera eficiente [56].

- **Numpy:** Aunque Numpy es una biblioteca de código abierto en Python diseñada principalmente para el análisis numérico, también puede aplicarse en el procesamiento de imágenes. Permite realizar tareas como recorte de imágenes, manipulación de pixeles, enmascaramiento de valores y otras funciones similares. Su estructura de datos se basa en matrices y arreglos multidimensionales. Además, Numpy facilita procesos como la reducción de color, binarización, recorte con pegado, inversión de colores (positiva o negativa) y muchas otras operaciones útiles en el manejo de imágenes [56].
- **CV simple:** Es un conocido framework de código abierto diseñado para desarrollar aplicaciones de vision por computador mediante el procesamiento de imágenes. Ofrece una interfaz intuitiva que facilita la conexión con cámaras, la conversión de formatos, la manipulación de imágenes y la extracción de características, entre otras funcionalidades [56].

1.4.9. Sistema de comunicación

Un sistema de comunicación se basa en un conjunto de dispositivos que son utilizados con la finalidad de transmitir, emitir y recibir señales de todo tipo, como voz, datos, audio, video, etc. Además, estas señales pueden ser de tipo digital o analógica [57].

Puede describirse fácilmente mediante tres elementos básicos (Figura 31): un transmisor, el cual se encarga de generar la señal que se desea y manipularla de tal forma que pueda viajar a través del canal, mediante procedimientos como modulación, filtrado, codificación, entre otros. Un medio de transmisión, el cual será el canal mediante el cual la señal va a viajar, y puede ser enviada a través de cables o de forma inalámbrica. Y, por último, el receptor, el cual realiza el procedimiento contrario del transmisor con el fin de reconstruir la señal y que esta sea lo más semejante a la original [57].

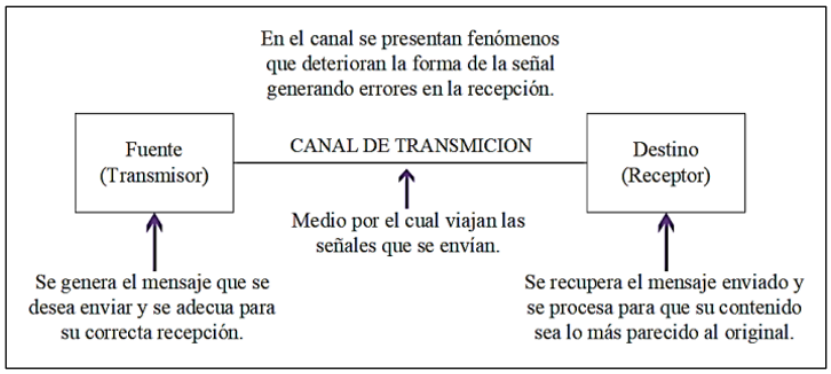


Figura 31: Sistema de comunicación.
Fuente: [58]

A continuación, se presentan los tipos de comunicación más comunes, con una descripción general de cada sistema.

1.4.9.1 Cable USB

La tecnología USB utiliza una arquitectura de comunicación serial y proporciona una interfaz de entrada y salida más veloz en comparación con los puertos seriales convencionales. Los cables USB cuentan con cuatro conductores: dos para la alimentación y dos para la transmisión de datos. Este bus es capaz de operar en tres velocidades de transferencia: HIGH SPEED (480 Mbps), FULL SPEED (12 Mbps) y LOW SPEED (1.5 Mbps). Tanto los controladores USB como los sistemas de recepción de datos y cables están diseñados para minimizar el ruido y reducir errores en la transmisión. Para poder hacer que los dispositivos se comuniquen correctamente, se requieren tres aspectos fundamentales: la detección de la conexión y desconexión del dispositivo, la capacidad del sistema para poder identificar y establecer un protocolo de intercambio de información con nuevos dispositivos conectados, y un mecanismo que permita la interacción entre el software, el hardware del USB y las aplicaciones que necesitan acceder a los periféricos conectados [59].

1.4.9.2 Bluetooth

Bluetooth es una tecnología que facilita la comunicación inalámbrica entre dispositivos y el acceso a internet sin necesidad de cables. Su propósito principal es simplificar la sincronización de datos. Se trata de una tecnología de corto alcance, bajo costo y pequeña escala, que utiliza enlaces de radio para la conexión entre teléfonos móviles y otros dispositivos. Opera en la banda

de 2.4 GHz y tiene la capacidad de atravesar paredes, lo que la convierte en una opción ideal para aplicaciones móviles. Además, permite el intercambio de datos a velocidades de hasta 1Mbit/s. Gracias a su esquema en frecuency hopping, la comunicación entre los dispositivos puede mantenerse así sea en entornos donde haya mucha interferencia electromagnética [60].

1.4.9.3 Wi-Fi

El Wi-Fi es una tecnología de comunicación inalámbrica que permite la conexión de dispositivos electrónicos a una red sin necesidad de cables, utiliza señales de radio. También se la conoce como red de área local inalámbrica (WLAN), la cual facilita el acceso a internet y la interconexión entre dispositivos como teléfonos inteligentes, tabletas y computadoras. Las redes Wi-Fi funcionan mediante la transmisión de ondas de radio en distintas frecuencias, como 2.4 GHz, 5 GHz y 6 GHz, lo que puede permitir operar a diferentes velocidades de conexión. En términos generales, las frecuencias más altas ofrecen mayor velocidad de transmisión de datos, mientras que las frecuencias más bajas, como 2.4 GHz, proporcionan un mayor alcance, aunque con menor velocidad [61].

1.4.10. Protocolo de comunicación

La comunicación entre el ordenador y el cuadricóptero se realizará a través de una conexión Wi-Fi (véase en la Figura 32), lo cual implica lo siguiente:

- **Control Remoto:** Los comandos de vuelo y las instrucciones de navegación se enviarán al dron a través de Python utilizando la conexión por medio de red Wi-Fi, permitiendo el control remoto desde una computadora o dispositivo móvil.
- **Transmisión de datos:** El dron puede transmitir datos de telemetría (como altitud, velocidad y orientación) y videos en tiempo real a través de la conexión Wi-Fi, lo que permite una monitorización continua y el ajuste de las rutas de vuelo.
- **Alcance Limitado:** La operación del dron está limitada por el alcance de la señal Wi-Fi, lo cual es adecuado para pruebas en entornos controlados y aplicaciones de corto alcance.

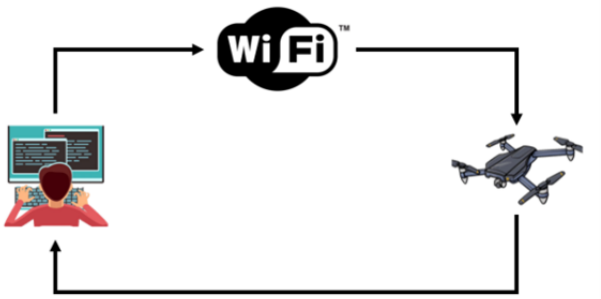


Figura 32: Protocolo de comunicación Wi-Fi.
Fuente: Autoría propia

1.4.11. Normativas técnicas y estándares de los UAV´S

En la siguiente sección se presentan las normativas y estándares, tanto nacionales como internacionales, que regulan el uso de cuadricópteros. Se ofrece una visión general de estos reglamentos con el objetivo de garantizar un pilotaje seguro de aeronaves no tripuladas.

Norma Oficial Mexicana NOM-107 SCT3-2019: Esta norma establece los requerimientos para poder pilotear un sistema de aeronave a distancia (RPAS) en el espacio aéreo mexicano. El objetivo de esta norma es detallar a la ciudadanía acerca del procedimiento que se debe realizar para poder registrar la nave por internet en la Agencia Federal de Aviación Civil (AFAC) [62].

La ley de aviación civil, en su artículo 4, establece que la navegación aérea dentro del territorio nacional se rige tanto por lo dispuesto en dicha ley como por los tratados internacionales suscritos por los Estados Unidos Mexicanos. En este contexto, México es parte del convenio sobre aviación civil internacional, firmado el 7 de diciembre de 1944 en Chicago, Illinois, Estados Unidos. Los sistemas de Aeronaves Pilotadas a Distancia (RPAS) representan un concepto innovador dentro del sector aeronáutico. Tanto la autoridad aeronáutica como la industria aeroespacial deben comprender, definir e integrar estos sistemas para garantizar su correcta operación. Su desarrollo se basa en tecnologías aeroespaciales de vanguardia, ofreciendo nuevas oportunidades para aplicaciones civiles y comerciales, al tiempo que contribuyen a mejorar la seguridad operacional y la eficiencia en la aviación civil [62]. La persona que opere el RPAS o que desee operarlo, debe cumplir con los siguientes requerimientos y limitaciones:

- Para obtener el documento de registro de RPAS, es necesario inscribir los documentos que acrediten la adquisición, transmisión, modificación,

gravamen o extinción de la propiedad, posesión y demás derechos reales sobre aeronaves civiles pilotadas a distancia. Este trámite se realiza a través del sitio web de la SCT/DGAC, con base en lo establecido en el artículo 47, fracción IV, de la ley de aviación civil y el artículo 14 del Reglamento de Registro Aeronáutico Mexicano. La obtención del documento debe llevarse a cabo confirme a lo indicado en el numeral 15.3 de la presente Norma Oficial Mexicana [62].

- Realizar operaciones dentro de clubes de aeromodelismo que cuenten con autorización de la Autoridad Aeronáutica, cumpliendo con los requisitos y restricciones operativas establecidas por dicho club y utilizando únicamente los espacios aéreos designados para su actividad [62].
- No se puede exceder la velocidad de operación proporcionada por el fabricante del RPAS [62].
- El máximo de altura para poder operar el RPAS es de 122 metros (400 ft) [62].
- La distancia máxima horizontal para operar el RPAS es de 457 metros (1500 ft) respecto al piloto [62].
- Se debe operar a una altura máxima de 100 metros (328 ft), en áreas donde alcance círculos de 9.2 Km y 18.5 Km, alrededor de los aeródromos [62].
- El piloto debe operar la aeronave en condiciones meteorológicas libres de nubes [62].
- Se debe mantener una visibilidad mínima de 1.5 Km desde el lugar donde se realiza la estación de control, esto se realiza antes de iniciar la operación [62].
- No se debe operar sobre personas, a menos que estas estén participando en la operación del RPAS, o que se encuentren dentro de lugares donde puedan permanecer cubiertos en caso de que la aeronave se desplome por algún error [62].
- Se debe mantener una distancia horizontal razonable de seguridad con respecto a personas que no se encuentren participando en la operación de la aeronave [62].

Reglamento Europeo de drones RD-2019/945 — RE-2019/947: El nuevo reglamento europeo busca unificar las normativas de los distintos Estados miembros y regular el uso civil de los drones sin importar su tamaño o peso. Su objetivo es establecer un marco común que abarque todos los escenarios operacionales posibles y se adapte a los avances tecnológicos actuales. Como se analizará a continuación, esta nueva normativa introduce cambios significativos en comparación con la legislación vigente en España, especialmente en lo relacionado con los requisitos de formación para pilotos de drones, el proceso de registro de operadores y las especificaciones técnicas necesarias para que las aeronaves obtengan la certificación CE [63].

En la categoría abierta, se establecen los vuelos de bajo riesgo, los cuales no requieren ningún tipo de autorización o permiso, ni tampoco el operador debe realizar alguna declaración [63]. Los requerimientos para cumplir son los siguientes:

- No se permite efectuar el vuelo sobre personas [63].
- No está permitido transportar materiales [63].
- No se permite las operaciones autónomas [63].
- Para que una persona pueda pilotear, es necesario que la edad mínima sea de 16 años [63].
- Siempre se debe mantener el UAS visible [63].
- La aeronave puede volar a una altura máxima de 120 m [63].
- El peso máximo de la aeronave tiene que ser menos de 25 Kg [63].

Para categorías específicas se aplican los siguientes requerimientos:

- Se puede operar a más de 120 m de altura [63].
- Pueden incluir drones de más de 25 Kg [63].
- Se puede realizar vuelos urbanos con drones de más de 4 Kg [63].
- Se puede realizar el transporte de materiales [63].
- Se permite volar sobre aglomeraciones de personas [63].
- Los pilotos deben tener una edad mínima de 16 años [63].
- Se necesita tener un registro del operador del UAS [63].

Reglamento UAS generales para el vuelo de drones en la República Argentina: La ANAC, entidad encargada de la seguridad de los drones en Argentina, ha publicado en internet información detallada sobre el uso de drones tanto recreativo como profesional. A continuación, se presentan los puntos más relevantes. Debido a la popularidad de adquisición de drones en los últimos años, ANAC decidió realizar modificaciones sobre las normas que rigen desde el 2015. Para realizar aquello no solo se decidió regular el vuelo recreativo y con objetivos fotográficos, sino que también se empezó a regular la transportación de cargas usando drones [64]. Las reglas más importantes para tener en consideración para realizar vuelos en Argentina son los siguientes:

- Absolutamente todos los drones deben estar registrados en el Registro Nacional de Aeronaves de ANAC [64].
- Las personas que pueden operar los drones deben tener mínimo 18 años. Sin embargo, personas de 16 y 17 podrán hacerlo siempre y cuando se encuentren en responsabilidad de un adulto [64].
- Está prohibido el vuelo de drones a menos de 5Km (3 millas) de lugares como: aeródromos, helipuertos y aeropuertos [64].
- La altura máxima de vuelo es de 122 m de altura [64].
- Se debe volar durante el día verificando que haya buenas condiciones climáticas [64].
- Está prohibido operar el vuelo en zonas como instalaciones gubernamentales o militares [64].
- Se debe mantener la visibilidad del dron todo el tiempo y no se puede operar el dron a menos de 30 m de edificios [64].

En Argentina los drones se pueden dividir en distintas categorías, las más importantes son de acuerdo con la aplicación y peso. Es decir, por aplicación se puede dividir en: recreativa (recreación, pasatiempo, diversión, etc.) Y comercial (uso con fines de lucro) [64]. Se dividen también de acuerdo con el peso:

- Clase A: hasta 500 gramos.
- Clase B: más de 500 gramos y hasta 5 kilogramos.
- Clase C: más de 5 kilogramos hasta 25 kilogramos.

- Clase D: más de 25 kilogramos hasta 150 kilogramos.
- Clase E: más de 150 kilogramos.

Todos los drones deben poseer registros con excepción de los de la Clase A, para fines recreativos [64].

Reglamento Ecuatoriano de aeronaves pilotadas a distancia (RPAS): La AAC (Autoridad de aviación civil) tiene la facultad de otorgar permisos especiales conforme a las disposiciones de este reglamento para operaciones con RPAS. Estos permisos aplican a aeronaves cuyas características técnicas y propósito de uso justifiquen su realización por parte de entidades públicas o privadas oficialmente reconocidas o autorizadas, siempre que estén destinadas exclusivamente a la investigación científica, la innovación y el desarrollo [65]. A continuación, se establece las reglas de vuelo para poder realizar operaciones de vuelo en aeronaves pilotadas a distancia (RPAS):

- El vuelo de las aeronaves puede efectuarse siempre y cuando no perjudique la seguridad de las operaciones aéreas, de personas que se encuentren circulando por el lugar, de sus bienes o fauna silvestre [65].
- La persona encargada de pilotear la aeronave debe suspender inmediatamente el vuelo siempre y cuando este ponga en peligro la seguridad de las personas [65].
- El operador de la aeronave es el responsable de la seguridad y operación de cada vuelo que se realice [65].
- La persona que se encargue de pilotear la aeronave es el responsable de que el vuelo no perjudique a personas, animales u otras aeronaves en caso de que se pierda el control por cualquier circunstancia [65].
- Cuando se realicen operaciones de vuelo bajo fines recreativos por menores de edad, estos deberán encontrarse bajo el cuidado y supervisión de un adulto responsable [65].
- El operador no podrá efectuar el vuelo si se encuentra fatigado, o si piensa que puede sufrir efectos de fatiga durante el vuelo [65].
- No se podrá ejercer el vuelo si el operador se encuentra bajo los efectos del alcohol o cualquier otra sustancia [65].
- No se podrá operar las aeronaves a una distancia igual o mayor a 9 kilómetros de los límites de cualquier aeródromo o zonas de seguridad del Estado [65].

- Las aeronaves no se pueden operar en zonas prohibidas o restringidas [65].
- Las aeronaves no se podrán volar en un radio de 9 kilómetros cerca de incendios forestales [65].
- Los operadores no podrán volar las aeronaves cerca de propiedades donde comprometa la privacidad de las personas [65].
- El vuelo de aeronaves no podrá efectuarse a menos de 150 metros (500 ft) de centros de privación de libertad o de rehabilitación social. Sin embargo, esta restricción no aplica a los RPAS operados por el Servicio Nacional de Atención Integral a personas adultas privadas de la libertad, ya que su uso está autorizado para labores de vigilancia y seguridad en dichos centros [65].

1.5. Marco Contextual

En el Ecuador, la evolución robótica va formando parte en aplicaciones productivas para la sociedad por medio de inteligencia artificial. En la actualidad se conoce que el uso de drones ha sido relevante en la parte de seguridad, vigilancia y agricultura. Siendo un enfoque a nuevos propósitos para los vehículos aéreos no tripulados.

Este proyecto permite desarrollar un sistema de navegación basado en visión por computador para el vuelo autónomo de drones en la provincia de Santa Elena. Además, el uso de procesamiento de imágenes junto con el modelo HSV es fundamental para ajustar la percepción de los colores de manera similar a la visión humana. Este enfoque permite guiar el vuelo del dron mediante la detección de colores (blanco y negro), facilitando la orientación y posicionamiento para asegurar que se mantenga dentro de una trayectoria definida.

Cabe destacar que el desarrollo del sistema será implementado dentro de un entorno estático y controlado, de tal manera que no afecte al rendimiento de vuelo del dron, es decir que podría realizarse en espacios delimitados, como en una habitación, en una vivienda o en un laboratorio. Esta estrategia permite una observación y análisis detallado de las capacidades del dron, permitiendo operar en un ambiente controlado para facilitar la identificación y resolución de posibles problemas, sin el riesgo asociado a entornos dinámicos impredecibles. Al mantener el dron en un espacio limitado y predecible, también se reducen las variables externas que podrían complicar el proceso de ajuste y calibración del sistema, esto con el fin de evitar que los factores físicos perjudiquen la autonomía de vuelo del dron. Es fundamental tener en consideración que las pruebas de vuelo del dron tienen que realizarse en áreas iluminadas para que el dron pueda posicionarse de mejor manera en el espacio. Ya que si el lugar carece de luminosidad podría afectar el rendimiento de vuelo debido a la mala captación de señal de los sensores.

2. Métodos y diseño experimental

En esta sección se describe detalladamente la metodología empleada para llevar a cabo el proyecto, así como los componentes físicos y lógicos que lo conforman. Se abordan criterios de selección de hardware y software, junto con su integración dentro del sistema propuesto. Además, se detalla el proceso de desarrollo de la propuesta, incluyendo el diseño y la implementación de los algoritmos con el fin de proporcionar una base sólida para su evaluación.

2.1. Métodos

Como punto de inicio fundamental, se deberá realizar un bosquejo de trabajos técnicos con bibliografías y algoritmos empleados en el campo de la visión por computador. Este proceso permitirá identificar estudios relacionados con la temática propuesta, proporcionando una base sólida para el desarrollo del proyecto. Se buscará referencias que contengan problemáticas similares y que ofrezcan soluciones relevantes, sirviendo como base para la formulación de estrategias y toma de decisiones en la implementación del sistema de seguimiento de trayectorias para drones.

Una vez identificadas y analizadas las variedades de aplicaciones utilizadas, se procederá a estudiar el método de seguimiento de línea convencional, empleado en robots equipados con sensores en la parte inferior para detectar y seguir una trayectoria definida [66]. Siguiendo esta lógica, se implementará un enfoque similar en el dron para el seguimiento horizontal, orientando su cámara hacia el suelo. La imagen capturada se dividirá en tres zonas (izquierda, centro y derecha). Esta segmentación permitirá simular sensores virtuales que detectarán la presencia de la línea guía, lo que facilitará la generación de comandos para rotar o trasladar el dron hacia la dirección correcta. En función de la ubicación de la línea en la imagen, el sistema determinará si debe continuar en línea recta o realizar un giro hacia la izquierda o la derecha para reorientarse.

En cuanto al seguimiento vertical, se adoptará un enfoque similar, con la diferencia de que la trayectoria se encontrará trazada sobre una pared. En este caso, la cámara del dron estará situada de frente y se analizarán dos franjas horizontales en la imagen, también dividida en tres regiones. A través del análisis de píxeles en cada región, se determinará si el dron debe moverse de manera lateral o vertical para continuar el trayecto. Estas funciones de control permitirán al dron ajustar su posición tanto en el eje vertical como en el horizontal, facilitando su desplazamiento a lo largo de superficies ver-

ticales. Cabe destacar que, al inicio del programa, el dron deberá situarse a una distancia prudente de la pared para asegurar una visión óptima de la trayectoria y evitar colisiones.

La metodología esencial para este sistema será el procesamiento de imágenes, llevado a cabo mediante la biblioteca OpenCV en el lenguaje de programación Python. La trayectoria estará representada por una línea negra compuesta por varios folios alineados, lo cual permitirá aplicar técnicas de detección de colores. No obstante, dado que el color negro puede presentar dificultades de segmentación en condiciones de iluminación variables, se utilizará el modelo de color HSV (Hue, Saturation, Value) para mejorar la detección. A través de barras deslizantes, se ajustarán los valores de tonalidad, saturación y brillo, optimizando así el filtrado de la línea en tiempo real. Mediante una función específica, se aplicará un rango de color sobre la imagen para generar una máscara binaria, en la que los pixeles que cumplan con los criterios definidos aparecerán en blanco. Esta imagen umbralizada permitirá descartar el fondo y centrarse únicamente en la trayectoria.

En la etapa final del procesamiento, se analizará el contorno más grande de la imagen umbralizada para determinar la ubicación de la trayectoria. El comportamiento de los motores del dron se ajustará en consecuencia: si la región central detecta un valor significativo, el dron avanzará en línea recta; si el patrón de activación corresponde a una curva hacia la izquierda o derecha, el sistema generará un giro en la dirección correspondiente, ajustando la velocidad de los motores para mantener el seguimiento de forma continua y estable.

El sistema de navegación propuesto tendrá como objetivo desarrollar una solución funcional para que un dron sea capaz de seguir de forma autónoma una trayectoria predeterminada en un entorno estático y controlado sin influencia de variables ambientales externas. Este entorno permitirá evaluar de mejor manera el comportamiento del sistema, lo cual es importante para validar el rendimiento del método de seguimiento mediante visión por computador.

Previo a la implementación del sistema, se realizará una etapa investigativa donde se analizarán las características técnicas del dron seleccionado, incluyendo sus modos de vuelo, conectividad inalámbrica, duración de batería y compatibilidad con librerías de programación. Esta etapa también conllevará la ejecución de comandos básicos de vuelo mediante un entorno de desarrollo integrado como Pycharm, con el propósito de validar la comunicación entre el dron y el software, además de facilitar la familiarización con

sus controles y respuestas.

Durante el proceso de desarrollo, se da la necesidad de adaptar el hardware del dron para mejorar la capacidad de detección en el seguimiento horizontal. En este caso, se diseñará un soporte impreso en 3D con un espejo integrado, que será colocado frente a la cámara con una inclinación especifica. Este soporte permitirá redirigir el campo visual de la cámara hacia el suelo, mejorando así la detección de la trayectoria y ampliando el alcance del sistema sin comprometer la estabilidad de vuelo.

2.1.1. Descripción del proyecto

El proyecto consiste en desarrollar un sistema de navegación para el seguimiento de trayectoria de vehículos aéreos no tripulados, utilizando técnicas de visión por computador, se empleará uno de los algoritmos que se adapte de mejor manera al propósito del sistema, permitiendo guiar el vuelo del dron a través de herramientas de procesamiento de imágenes y detección de colores.

El sistema de navegación será diseñado e implementado en el lenguaje de programación Python, en el cual el proceso de guiado se basará en la identificación de una línea mediante la cámara integrada en el dron, que servirá como referencia visual para su trayectoria.

Para la implementación, se realizará la selección de un dron que mejor se adapte a las características y necesidades requeridas para la planificación del proyecto, considerando diversos factores como la capacidad de carga, controlabilidad de vuelo, estabilidad, compatibilidad con softwares de programación etc. Esta selección es importante ya que garantizará que el dron elegido cumpla con los requerimientos y objetivos establecidos para obtener un mejor resultado. A continuación, se muestra en la Figura 33 un esquema, para entender mejor el proceso a seguir para la realización del proyecto.

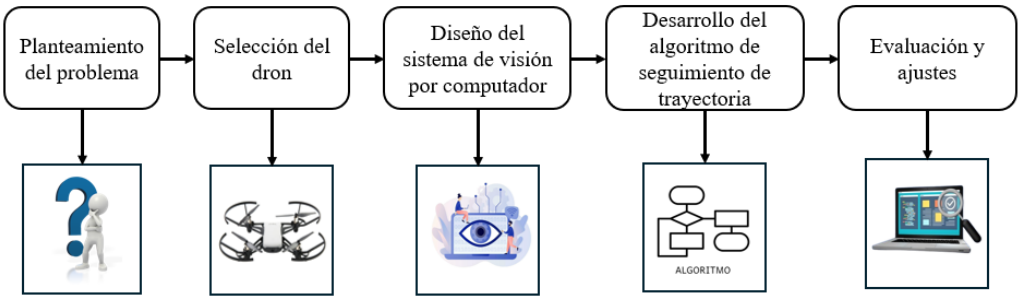


Figura 33: Esquema pictórico de descripción de proyecto.
Fuente: Autoría propia

2.2. Componentes de la propuesta

En el siguiente apartado, se lleva a cabo una descripción de los componentes físicos y lógicos necesarios para la elaboración de este proyecto. Se detalla sus principales características técnicas, junto con una breve explicación de su funcionalidad, destacando la relevancia en el diseño e implementación del sistema.

2.2.1. Componentes físicos

En robótica, el hardware se encuentra conformado por componentes físicos que ayudan a facilitar la comunicación entre los sistemas computacionales y el entorno real. Pueden clasificarse en varias categorías, como sensores y actuadores, cada una de las cuales desempeña una función específica en el funcionamiento del sistema. A continuación, se presentan los componentes físicos utilizados en el desarrollo de la propuesta.

2.2.1.1 Dron DJI Tello

El Dron DJI Tello es un pequeño pero interesante cuadricóptero compacto diseñado para brindar una experiencia de vuelo estable y entretenida (Figura 34). En la parte frontal lleva una cámara y un sistema de posicionamiento visual que, junto con su avanzado controlador de vuelo, permite que el dron pueda mantenerse estable incluso en interiores. Cuenta también con distintos modos de vuelo, captura fotos de 5 megapíxeles, transmite video en 720p y tiene un alcance de hasta 100 metros (328 pies) [67].

Figura 34: Dron DJI Tello.
Fuente: [68]

Una de las ventajas importantes que tiene este modelo de dron, es la capacidad de ser programable, usando SDK (Software Development Kit), en el cual el usuario puede controlar los movimientos del dron. Esta ventaja favorece la realización del proyecto ya que se adapta a las características deseadas. En la Tabla 1 se muestran las especificaciones técnicas del Dron DJI Tello.

Tabla 1: Especificaciones técnicas del dron DJI Tello

Especificaciones Técnicas	
Aeronave	
Modelo	TLW004
Peso (incluidos los protectores para las hélices)	87 gr
Velocidad máxima	28.8 km/h (17.8 mph)
Tiempo de vuelo máximo	13 minutos (sin viento a 15km/h [9 mph] sostenidos)
Intervalo de temperaturas de funcionamiento	De 0 °C a 40 °C (de 32 °F a 104 °F)
Intervalo de frecuencias de funcionamiento	De 2.4 a 2.4835 GHz
Transmisor (PIRE)	20 dBm (FCC), 19 dBm (CE), 19 dBm (SRRC)
Cámara	
Tamaño de imagen máximo	2592 x 1936
Modos de grabación de video	HD: 1280 x 720 30p
Formato de video	MP4
Batería de vuelo	
Capacidad	1100 mAh
Voltaje	3.8 V
Tipo de batería	LiPo
Energía	4.18 Wh
Peso neto	2 gr
Intervalo de temperatura de carga	De 5 ° a 45 °C (de 41 ° a 113 °F)
Potencia de carga máxima	10 W

Fuente: Adaptado de manual de uso de categoría de drones [67]

2.2.1.2 Sistema de posicionamiento visual

Este sistema permite que el dron pueda mantener una posición constante, es decir que el Dron Tello pueda volar en modo estacionario con una mejor precisión volando en interiores o exteriores sin la presencia del viento (Figura 35). Los componentes del sistema en el dron Tello son una cámara y un módulo infrarrojo 3D que se encuentran colocados en la parte inferior del dron [67]. En la siguiente figura se muestra el lugar donde se encuentran situados estos componentes.

Figura 35: Sistema de posicionamiento visual en dron Tello.
Fuente: Autoría propia

2.2.1.3 Motores

El Dron DJI Tello lleva incluidos motores sin escobillas (Figura 36) lo cual ayuda a tener un mejor rendimiento y durabilidad más estable. Los motores del dron son DC (corriente directa) y se encuentran diseñados para generar una rápida respuesta para que la aeronave tenga un vuelo estable [69].

Figura 36: Motores sin escobillas del dron DJI Tello.
Fuente: [70]

Los cuatro motores impulsan las hélices de dos palas para generar la sustentación del dron y pueda mantenerse en el aire, estos motores contienen

un peso liviano permitiendo que el tiempo máximo de vuelo del Dron Tello sea de aproximadamente de 13 minutos. En un dron dos de los motores giran en el sentido de las manecillas del reloj (clockwise, CW), mientras que los otros dos giran en sentido contrario (counter clockwise, CCW). Generalmente, en este tipo de drones, los motores identificados con cables de color azul y negro corresponden a los de tipo CW, es decir, giran en el sentido de las manecillas del reloj. Por otro lado, los motores con cables de color blanco y negro son de tipo CCW, lo que significa que giran en dirección opuesta. Esta configuración es fundamental para mantener la estabilidad y el control del dron durante el vuelo, esto se puede visualizar en la Figura 37.

Las especificaciones de los motores se encuentran en la Tabla 2.

Tabla 2: Especificaciones de los motores

Especificaciones de los motores	
Voltaje	3.7 V DC
Juego final del eje	0.05–0.3 mm
Velocidad sin carga	52000 RPM
Corriente sin carga	93 mA (MÁX)
Corriente de carga	2200 mA (MÁX)
Voltaje de arranque	0.8 V DC (MÁX)
Peso bruto	5.5 g/pc

Fuente: Adaptado de Tello Motors [70]

Figura 37: Sentido de giro de motores.
Fuente: Autoría propia

2.2.1.4 Sistema de propulsión

El sistema de propulsión desempeña un papel fundamental en un dron, ya que permite generar la sustentación necesaria para mantenerlo en el aire y evitar que caiga. Esto se logra mediante el giro a alta velocidad de las hélices, las cuales impulsan el flujo de aire hacia abajo y contrarrestan la fuerza de gravedad. En el caso del Dron Tello, las hélices son de dos palas y están diseñadas con un peso ligero, lo que mejora la eficiencia del vuelo y optimiza el consumo de energía. A primera vista, el movimiento de un dron puede parecer complejo, sin embargo, su funcionamiento es bastante intuitivo, el dron cuenta con cuatro grados de libertad, lo que le permite trasladarse en tres direcciones. Para realizar movimientos de traslación, es necesario que todos los motores giren a la misma velocidad [10]. Cuando la fuerza de sustentación generada supera el peso del dron, este asciende, esto se visualiza en la Figura 38.

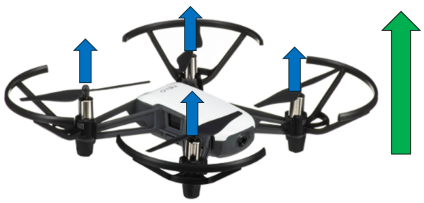


Figura 38: Movimiento ascendente de un dron.
Fuente: Autoría propia

Para lograr un movimiento de descenso, es necesario reducir la velocidad de todos los motores, de modo que la fuerza de sustentación sea inferior al peso del dron como se muestra en la Figura 39 [10].

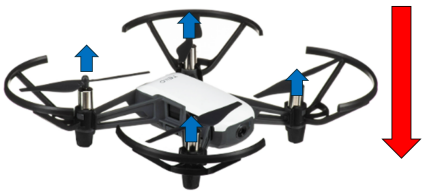


Figura 39: Movimiento descendente de un dron.
Fuente: Autoría propia

Para realizar movimientos laterales, como desplazarse hacia la izquierda (como se muestra en la Figura 40), se disminuye la velocidad de los motores del lado izquierdo y se incrementa la de los motores del lado derecho, lo que permite el desplazamiento en esa dirección [10].

Figura 40: Desplazamiento a la izquierda de un dron.
Fuente: Autoría propia

De manera inversa, para desplazarse hacia la derecha (como se muestra en la Figura 41), se debe reducir la velocidad de los motores del lado derecho y aumentar la de los motores del lado izquierdo, lo que permite alcanzar el movimiento deseado [10].

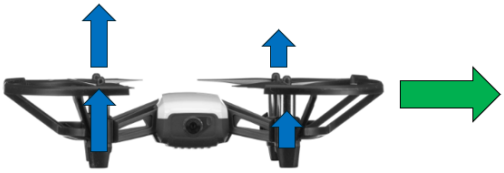


Figura 41: Desplazamiento a la derecha de un dron.
Fuente: Autoría propia

De forma similar, el dron puede desplazarse en la tercera dirección, que corresponde al movimiento hacia adelante y hacia atrás. Para avanzar, se reduce la velocidad de los motores frontales y se incrementa la de los motores traseros, mientras que, para retroceder, se invierte este ajuste (véase en la Figura 42) [10].

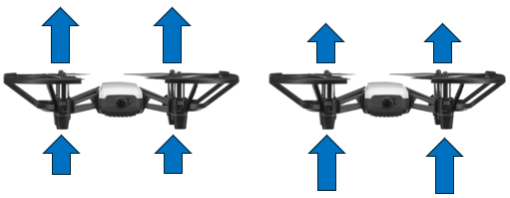


Figura 42: Avance y retroceso de un dron.
Fuente: Autoría propia

Para lograr un desplazamiento hacia atrás, es necesario reducir la velocidad de los motores traseros y aumentar la de los motores delanteros. Esto es posible debido a que dos de los motores giran en el sentido horario, mientras que los otros dos lo hacen en sentido antihorario. Con base en este principio, se puede hacer que el dron gire en dirección de las agujas del reloj disminuyendo la velocidad de los motores que giran en ese mismo sentido y aumentando la de los que giran en dirección opuesta (véase en la Figura 43) [10].

Figura 43: Rotación en sentido horario.
Fuente: Autoría propia

De manera similar, para girar en sentido antihorario, se reduce la velocidad de los motores que giran en esa dirección y se incrementa la de los motores que giran en sentido horario (Figura 44) [10].

Figura 44: Rotación en sentido antihorario.
Fuente: Autoría propia

2.2.1.5 Soporte impreso en 3D

Como se mencionó en el capítulo 1, para lograr un seguimiento de trayectoria de manera horizontal, es fundamental que la cámara del dron pueda capturar la superficie debajo de él. Sin embargo, el Dron Tello tiene su cámara ubicada en la parte frontal y transmite video en orientación horizontal, lo que representa un problema. Para solucionar esto, se utiliza un soporte impreso en 3D (Figura 45) con un espejo inclinado frente a la cámara, lo

que le permite redirigir su enfoque hacia abajo y mejorar la visualización de la superficie horizontal. Por otro lado, para el seguimiento de trayectoria de manera vertical, no es necesario hacer uso de este soporte, ya que la posición frontal de la cámara del dron le permite captar sin dificultad superficies verticales.

Figura 45: Soporte impreso en 3D.
Fuente: Autoría propia

2.2.2. Componentes lógicos

En esta sección se describirán los componentes lógicos necesarios para la programación del Dron Tello, permitiéndole ejecutar las tareas establecidas de manera correcta. Estos componentes son fundamentales para asegurar el buen funcionamiento del sistema de navegación.

2.2.2.1 PyCharm

PyCharm es un entorno de desarrollo integrado (IDE) creado por JetBrains, una empresa reconocida por sus herramientas de desarrollo de alto nivel. Este IDE está especialmente diseñado para desarrolladores que utilizan el lenguaje de programación Python. Existen dos versiones principales de PyCharm: la edición Community, que es gratuita y de código abierto, y la versión Professional, que es de pago e incluye funciones avanzadas [71]. PyCharm ofrece múltiples beneficios. Su editor inteligente permite escribir código de manera eficiente y con alta calidad. La diferenciación de colores para palabras claves, clases y funciones facilita la lectura y comprensión del código, además de hacer más sencilla la detección de errores. También cuenta con una función de autocompletado que agiliza la programación. Las herramientas de navegación dentro del código permiten a los desarrolladores modificar y mejorar su trabajo con facilidad, accediendo rápidamente a funciones, clases o archivos. Además, el modo “Lens” ayuda a inspeccionar y depurar el código fuente de manera efectiva. La función de refactorización

permite modificaciones rápidas en variables locales y globales sin afectar el rendimiento del código, optimizando su estructura interna.

Este entorno de desarrollo también facilita la creación de aplicaciones web en Python, ya que es compatible con tecnologías como HTML, CSS y JavaScript. Además, los cambios pueden visualizarse en tiempo real a través del navegador. También admite frameworks web de Python, como Django y proporciona herramientas avanzadas como autocompletado, sugerencias de parámetros y un depurador integrado. También es compatible con otros frameworks como web2Py y Pyramid.

Por último, este IDE admite bibliotecas científicas de Python, como Matplotlib, Numpy Y Anaconda, lo que lo convierte en una excelente opción para proyectos de ciencia de datos, visión por computador y aprendizaje automático. La visualización de datos mediante gráficos interactivos facilita el análisis, y la integración con herramientas como Django, IPython y Pytest permite desarrollar soluciones innovadoras [72]. En la Figura 46 se muestran las áreas de trabajo del entorno de desarrollo integrado PyCharm.

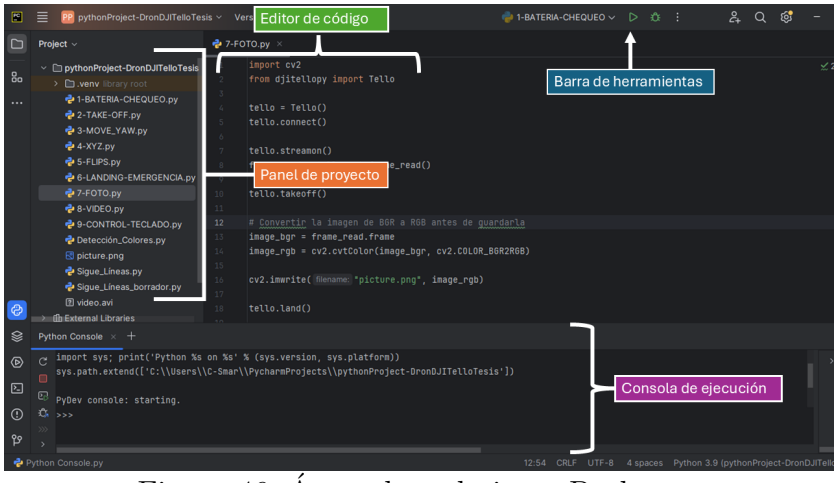


Figura 46: Áreas de trabajo en Pycharm.
Fuente: Autoría propia

A continuación, se presenta una descripción de las principales áreas de trabajo que conforman el entorno de desarrollo integrado (IDE) PyCharm, destacando sus funciones y utilidades para optimizar el flujo de trabajo del desarrollador.

- **Panel de proyecto (Project Panel):** Esta se ubica en la parte izquierda, permite explorar los archivos y directorios del proyecto, facilitando la administración y organización.

- **Editor de código (Editor Window):** Es el espacio donde se escribe y edita el código fuente, se destaca por su resaltado de sintaxis, función de autocompletado y otras herramientas visuales que mejoran la experiencia de programación.
- **Barra de herramientas (Toolbar):** Situada justo debajo del menú principal, proporciona accesos directos a funciones clave, como la ejecución y depuración del codigo, asi como la gestión de versiones.
- **Consola de ejecución (Run/Debug Console):** Generalmente ubicada en la parte inferior, muestra la salida del programa, registros de ejecución y mensajes de error, además de facilitar el proceso de depuración.

2.2.2.2 Librería djitellopy

Es un paquete específico para el Dron DJI Tello que, al importar su librería, permite acceder a un conjunto de comandos diseñados para controlar el dispositivo mediante programación [73]. Esta biblioteca se basa en el SDK oficial de Tello y ofrece diversas funcionalidades, entre las que se incluyen:

- Integración completa de los comandos disponibles para el Dron DJI Tello.
- Captura y recuperación de secuencias de video en tiempo real.
- Recepción y análisis de paquete de estado para monitorear el rendimiento del dron.
- Capacidad de controlar múltiples drones de manera simultánea (enjambre de drones).
- Compatibilidad con Python 3.6 y versiones posteriores.

En la siguiente Tabla 3, se muestran los comandos básicos del Dron DJI Tello, y sus respectivas funciones: El primer paso consiste en asignar un nombre a la variable que se utilizará para controlar el dron, ya que a través de esta variable se ejecutarán todos los comandos posteriores, en este caso se hace uso de la palabra “DRON”.

Tabla 3: Comandos del dron

Comandos del dron	
`DRON = tello.Tello()`	Crea una instancia del objeto Tello que pertenece a la biblioteca djitellopy.
`DRON.connect()`	Establece la conexión entre el programa y el dron.
`DRON.streamon()`	Activa la transmisión de video en tiempo real desde la cámara del dron.
`DRON.takeoff()`	Permite que el dron despegue desde su posición actual.
`DRON.land`	Permite el aterrizaje del dron.
`DRON.move_forward()`	Realiza el movimiento hacia delante.
`DRON.move_back()`	Realiza el movimiento hacia atrás.
`DRON.move_left()`	Realiza el movimiento hacia la izquierda.
`DRON.move_right()`	Realiza el movimiento hacia la derecha.
`DRON.move_up()`	El dron realiza el movimiento ascendente.
`DRON.move_down()`	El dron realiza el movimiento descendente.
`DRON.rotate_clockwise()`	Permite que el dron rote en el sentido de las manecillas del reloj.
`DRON.rotate_counter_clockwise()`	Permite que el dron rote en el sentido contrario a las manecillas del reloj.
`DRON.go_xyz_speed()`	Permite que el dron se mueva en el espacio tridimensional.
`DRON.flip_forward()`	El dron realiza una voltereta hacia delante.
`DRON.flip_back()`	El dron realiza una voltereta hacia atrás.
`DRON.flip_left()`	El dron realiza una voltereta hacia la izquierda.
`DRON.flip_right()`	El dron realiza una voltereta hacia la derecha.
`DRON.send_rc_control(...)`	Permite el control manual de las velocidades de desplazamiento (horizontal y vertical) y la rotación del dron enviando valores directos a sus canales de control remoto.

Fuente: Autoría propia

2.2.2.3 Librería Numpy

Numpy es una biblioteca de Python enfocada en el cálculo numérico y el análisis de datos, especialmente cuando se trabaja con grandes volúmenes de información. Introduce una nueva estructura de datos llamada "arreglo", que permite almacenar y manipular colecciones de elementos del mismo tipo en múltiples dimensiones de manera eficiente. Una de las principales ventajas de Numpy en comparación con las listas convencionales de Python es su alto rendimiento, ya que los "arreglos" pueden procesarse hasta 50 veces más rápido, esto la convierte en una herramienta ideal para el manejo, cálculo de vectores y matrices de gran tamaño [74].

Un arreglo es una estructura de datos compuesta por los elementos del mismo tipo, organizados en una tabla o cuadrícula con múltiples dimensiones como se muestra en la Figura 47, cada una de las dimensiones se denomina eje dentro del arreglo.

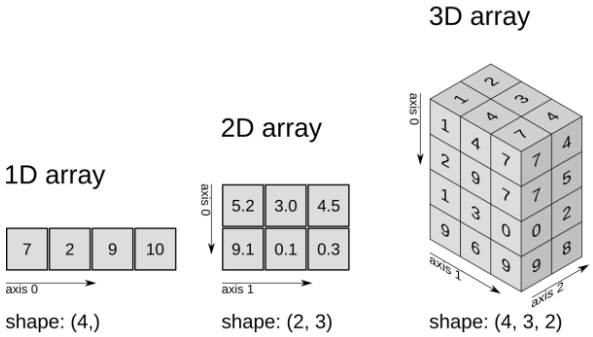


Figura 47: Arreglos en 1D, 2D y 3D.
Fuente: [74]

En Numpy, los arreglos se crean utilizando la función "np.array(lista)", la cual genera un arreglo a partir de una lista o tupla y devuelve una referencia a él [74]. La cantidad de dimensiones del arreglo dependerá de la estructura interna de la lista o tupla proporcionada, por lo cual se debe tomar en consideración lo siguiente:

- Si se proporciona una lista simple de valores, se obtiene un arreglo unidimensional (vector).
- Si se usa una lista que contiene otras listas, se genera un arreglo bidimensional (matriz).
- Si la lista contiene listas anidadas en más niveles, se obtiene un arreglo tridimensional (cubo).

- Este patrón se extiende a dimensiones superiores, sin un límite específico, salvo por la capacidad de memoria disponible en el sistema.

2.2.2.4 Librería OpenCV

OpenCV (Open Source Computer Vision) surgió inicialmente como un proyecto de investigación desarrollado por Intel. En la actualidad es la biblioteca de visión por computador más extensa en cuanto a la cantidad de funcionalidades que ofrece. El módulo cv2 es una parte fundamental de OpenCV, ya que proporciona a los desarrolladores una interfaz sencilla para manejar funciones de procesamiento de imágenes y video de manera eficiente [75].

OpenCV es una biblioteca en visión artificial que proporciona una amplia variedad de herramientas para el procesamiento y análisis de imágenes y videos. Su estructura modular facilita la resolución de múltiples desafíos en el campo de la visión por computador, como la detección de características, el reconocimiento de objetos y la calibración de cámaras. Al ser un proyecto de código abierto, ha sido ampliamente adoptado en sectores de investigación, desarrollo y producción, convirtiéndose en una opción flexible para aplicaciones que van desde dispositivos móviles hasta sistemas de seguridad [76].

Aunque la mayoría de sus funciones están optimizadas en C/C++, OpenCV cuenta con enlaces para varios lenguajes de programación, destacando especialmente su compatibilidad con Python. Esta integración permite un desarrollo ágil, ya que Python ofrece un entorno eficiente para la creación y prueba de algoritmos de visión artificial. Su uso es clave en entornos de aprendizaje automático y en aplicaciones industriales y académicas relacionadas con la visión por computador. La documentación de OpenCV se actualiza constantemente, incorporando nuevas funciones y mejorando las ya existentes. Esto permite que los proyectos basados en esta biblioteca evolucionen de manera fluida, aprovechando los avances más recientes en el área. Por su alto rendimiento y fiabilidad en producción, OpenCV es frecuentemente recomendado en publicaciones y foros especializados como una de las mejores soluciones para el análisis de imágenes [76]. A continuación, en la Tabla 4 se presentan los comandos más utilizados en esta librería acompañados de una breve descripción de su funcionamiento.

Tabla 4: Comandos de OpenCV

Comandos de OpenCV	
`cv2.imread()`	Lee una imagen desde un archivo y la carga en formato de matriz.
`cv2.imshow()`	Muestra una imagen en una ventana emergente.
`cv2.imwrite()`	Guarda una imagen en el disco.
`cv2.IMREAD_COLOR`	Carga la imagen a color.
`cv2.IMREAD_GRAYSCALE`	Carga la imagen en escala de grises.
`cv2.IMREAD_UNCHANGED`	Carga la imagen como está definida.
`cv2.waitKey()`	Espera un evento de teclado.
`cv2.destroyAllWindows()`	Cierra todas las ventanas abiertas.
`cv2.VideoCapture()`	Inicia la captura de video desde una fuente.
`cv2.line()`	Dibuja una línea en una imagen.
`cv2.circle()`	Dibuja un círculo en una imagen.
`cv2.putText()`	Agrega texto a una imagen.
`cv2.createTrackbar()`	Crea un control deslizante en una ventana.
`cv2.getTrackbarPos()`	Obtiene la posición actual de un control deslizante.
`cv2.resize()`	Cambia el tamaño de una imagen.
`cv2.add()`	Realiza una operación de suma de imágenes.
`cv2.bitwise()`	Realiza operaciones bit a bit entre dos imágenes o matrices.
`cv2.cvtColor(img1, cv2.COLOR_BGR2HSV)`	Convierte una imagen de BGR a HSV (espacio de color).
`cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)`	Convierte una imagen de BGR a escala de grises.
`cv2.namedWindow()`	Crea una ventana con un nombre específico.
`cv2.inRange()`	Realiza la segmentación de una imagen basada en un rango de valores de color.
`cv2.countNonZero()`	Cuenta los píxeles en una imagen que tienen un valor distinto de cero.
`cv2.findContours()`	Encuentra los contornos de los objetos en una imagen.
`cv2.RETR_EXTERNAL`	Recupera los contornos externos.
`cv2.CHAIN_APPROX_SIMPLE`	Comprime segmentos horizontales, verticales, diagonales y deja solo puntos finales ahorrando memoria.

Fuente: Autoría propia

2.3. Diseño experimental

A continuación, se presenta el desarrollo del sistema de navegación para seguimiento de trayectorias aplicado a un vehículo aéreo no tripulado, basado en visión por computador y la integración de técnicas esenciales para el análisis de imágenes. El dron seleccionado para llevar a cabo esta propuesta es el modelo DJI Tello, elegido por sus dimensiones compactas, ideales para operar en entornos controlados. La implementación de los algoritmos se realizará en el entorno de desarrollo PyCharm, y posteriormente se llevarán a cabo diversas pruebas experimentales con el fin de validar el funcionamiento del sistema.

2.3.1. Diseño e implementación del hardware

El dron DJI Tello cuenta con un diseño compacto e integrado, por lo que sus componentes internos no son accesibles directamente debido a su estructura cerrada. Sin embargo, es posible identificar y observar sus elementos externos, como se muestra en la Figura 48, donde se aprecian la disposición de los motores, hélices, protectores, sensores, cámara incorporada, entre otros.

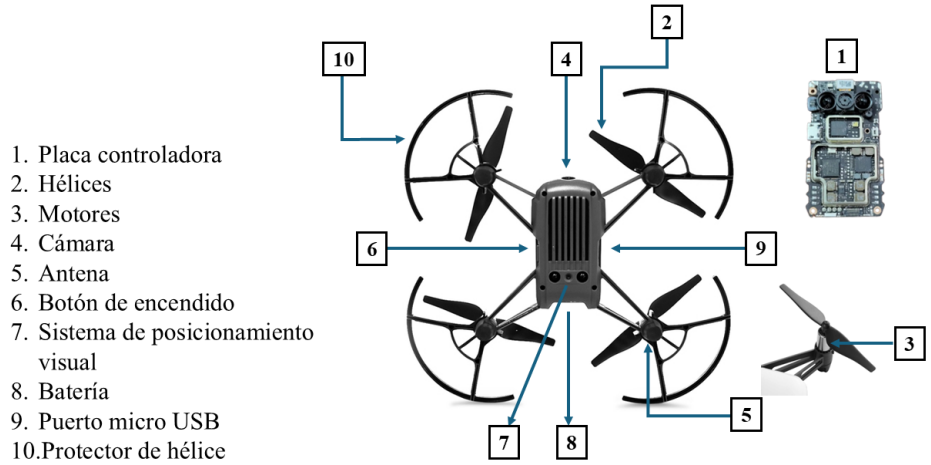


Figura 48: Visualización y reconocimiento de elementos conectados al dron. Fuente: Autoría propia

2.3.1.1 Acoplamiento de soporte impreso en 3D

Como se mencionó previamente en la descripción de los componentes físicos, para el seguimiento de trayectoria horizontal, se utiliza un soporte

adaptado con un espejo. Este se coloca en la parte frontal del dron, como se visualiza en la Figura 49, con el fin de redirigir la visión de la cámara hacia el suelo, lo que permite una mejor visualización del entorno y un enfoque directo de la trayectoria en superficie.

Figura 49: Soporte acoplado al dron.
Fuente: Autoría propia

Es importante destacar que este acople ha sido fabricado con filamento de bajo peso, al igual que el espejo, ya que un soporte demasiado pesado podría afectar la estabilidad del dron durante el vuelo. Por ello, se recomienda que el conjunto sea lo más liviano posible. Además, es fundamental asegurar una correcta colocación del soporte, ya que una mala orientación podría impedir que la cámara capture adecuadamente el suelo, lo que podría generar errores en la detección de la trayectoria.

2.3.1.2 Comunicación del dron Tello con el ordenador

Para establecer comunicación con el dron, lo primero que se debe hacer es presionar el botón de encendido ubicado en el lateral izquierdo del dispositivo. A continuación, se debe esperar a que el led comience a parpadear en color naranja, lo que indica que la red Wi-Fi del dron está activa.

Además, desde el ordenador, se debe buscar la red Wi-Fi identificada con el nombre "Tello", seguida por una serie de números correspondientes al modelo específico del dron. Una vez conectada esta red, el ordenador estará vinculado al dron y listo para enviar instrucciones y comandos, esto se muestra en la Figura 50.

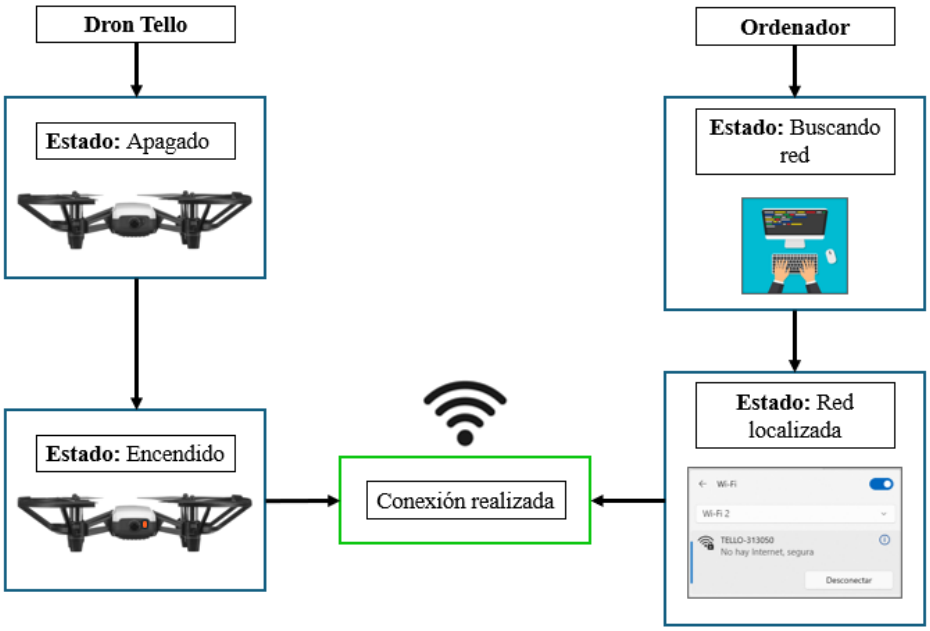


Figura 50: Conexión del dron con el ordenador.
Fuente: Autoría propia

2.3.1.3 Lógica de rotación y traslación para seguimiento horizontal aplicado a dron Tello

En el seguimiento de trayectorias de manera horizontal, se emplea un enfoque inspirado en los robots seguidores de línea convencionales. Estos prototipos suelen tener una estructura alargada y están equipados con múltiples sensores ubicados en su parte inferior, orientados hacia el suelo. Esta disposición permite una mejor detección de la línea y una correcta distinción entre el recorrido y el fondo, facilitando así la navegación a lo largo de la trayectoria.

En el caso del dron Tello, se aplica una lógica similar a la de los robots seguidores de línea convencionales, asumiendo la existencia de tres sensores virtuales distribuidos en las posiciones izquierda, centro y derecha, como se ilustra en la Figura 51. Esta configuración permite simular el comportamiento de los sensores físicos utilizados en los robots seguidores de línea. Cada sensor puede asumir dos posibles estados lógicos: 1 (encendido), cuando detecta la presencia de la línea bajo su posición y 0 (apagado), cuando no la detecta.

Figura 51: Sensores virtuales izquierda, centro y derecha.
Fuente: Autoría propia

Para que el dron avance de manera recta, el sensor central debe estar en estado encendido, mientras que los sensores izquierdo y derecho deben permanecer apagados, lo que indica que la línea se encuentra centrada respecto al dron. Cuando este se aproxima a una curva, ya sea a la izquierda o derecha, el patrón de activación cambia: dos sensores se encenderán, señalando que una mayor parte de la línea está siendo detectada en esa dirección, como se observa en la Figura 52. Este comportamiento permite al dron ajustar su trayectoria según la posición de la línea.

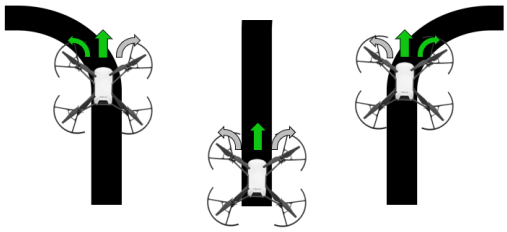


Figura 52: Indicación de giro durante avance del dron.
Fuente: Autoría propia

En la figura anterior se observa que, al tomar una curva hacia la izquierda, los sensores central e izquierdo se encuentran encendidos, mientras que, en una curva hacia la derecha, se activan los sensores central y derecho. Esto ocurre porque la línea negra es visible en esas regiones del campo visual del dron. Con base en esta detección, se puede instruir al dron para que realice un giro suave en la dirección correspondiente, facilitando así la toma de la curva. Sin embargo, si en algún momento solo uno de los sensores laterales (izquierdo o derecho) se encuentran encendidos, como se muestra en la Figura 53, esto puede deberse a que el dron no mantiene una velocidad adecuada o a la presencia de curvas más cerradas en la trayectoria, lo que significaría que

el dron deberá rotar un poco más para tomar la curva con mejor precisión.

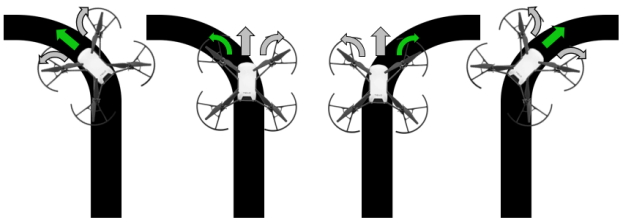


Figura 53: Indicación en giro pronunciado.
Fuente: Autoría propia

Una vez realizada la corrección inicial del giro, durante el avance del dron es posible que, en una curva hacia la izquierda, permanezcan encendidos los sensores central y derecho. Esto indica que el dron se ha desalineado ligeramente y requiere una rotación en sentido horario para corregir su orientación y reposicionarse adecuadamente. De manera similar, si el dron se encuentra en una curva hacia la derecha y se activan los sensores central e izquierdo, será necesario que realice un giro en sentido antihorario para volver a alinearse correctamente con la trayectoria, como se muestra en la Figura 54.

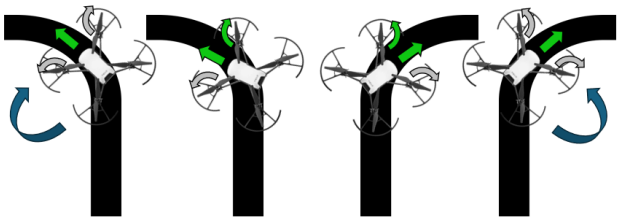


Figura 54: Reorientación mediante rotación leve.
Fuente: Autoría propia

El objetivo es que el dron busque constantemente alinearse con el centro de la línea y se desplace hacia ese punto, o lo más cerca posible de él, como se ilustra en la Figura 55. Esto permite que el dron siga la trayectoria de forma efectiva, realizando movimientos de rotación y traslación siempre que permanezca próximo a la línea. En caso contrario, si el dron se aleja demasiado, se pierde la referencia necesaria para determinar cómo y cuánto debe girar, lo que genera un nuevo desafío: identificar su posición actual respecto a la trayectoria y decidir el movimiento adecuado. En este sentido, en cada instante se intenta localizar el centro de la línea y la posición relativa del dron. Si la línea detectada se encuentra centrada en la imagen, el dron continúa avanzando en línea recta; si hay una desviación, se aplican correcciones en la dirección para retomar el seguimiento.

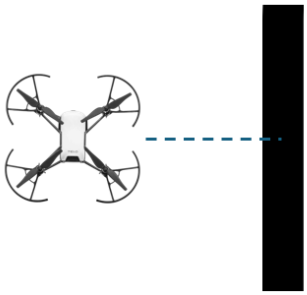


Figura 55: Búsqueda del centro de la línea.
Fuente: Autoría propia

Entonces, el funcionamiento de los sensores virtuales para el dron Tello, sigue la siguiente lógica digital como se muestra en la Figura 56.

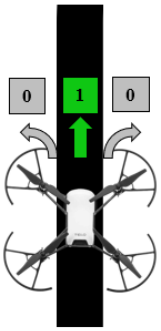


Figura 56: Lógica digital de los sensores virtuales del dron.
Fuente: Autoría propia

Dado que se utilizarán tres sensores virtuales, se puede determinar el número total de combinaciones posibles mediante la fórmula 2^n, donde n representa la cantidad de sensores. En este caso, $2^3 = 8$, lo que indica que existen ocho configuraciones distintas en las que el dron puede detectar la línea. Estas combinaciones se analizan detalladamente en la Tabla 5 a continuación.

Tabla 5: Combinaciones de sensores virtuales y su movimiento asociado

n	Sensor izquierdo	Sensor central	Sensor derecho	Movimiento
0	0	0	0	X
1	0	0	1	Rot. Derecha
2	0	1	0	Avance
3	0	1	1	Rot. Leve der.
4	1	0	0	Rot. Izquierda
5	1	0	1	X
6	1	1	0	Rot. Leve izq.
7	1	1	1	X

Fuente: Autoría propia

En esta Tabla 5 se pueden visualizar tres combinaciones con movimientos con una (X), es decir situaciones un poco probables, o también combinaciones que se podrían dar para otros tipos de trayectorias.

- $n = 0$. Este caso significa que el dron no ha detectado nada de trayectoria en los sensores virtuales.
- $n = 5$. Esta combinación puede darse, debido a una falsa detección de trayectoria, pero es imposible que el dron encontrándose en esa situación pueda realizar giros a la izquierda o derecha al mismo tiempo.
- $n = 7$. Este caso es similar al anterior, ya que el dron no puede realizar varios movimientos a la vez. Además, esta combinación puede darse debido a varios factores, como el grosor considerable de la trayectoria, la proximidad con la que el dron vuela respecto a la línea, o la complejidad del recorrido, que puede incluir cruces y curvas muy cerradas.

2.3.1.4 Lógica de movimiento para seguimiento vertical aplicado a dron Tello

Para el seguimiento vertical de la trayectoria, se adopta un enfoque similar al utilizado en el seguimiento horizontal, con la diferencia de que ahora la trayectoria se encuentra ubicada en la pared. En este caso, el uso de una estructura de soporte en 3D ya no es necesario, ya que el dron Tello cuenta con una cámara frontal que proporciona una visión clara del área directamente frente a él.

En este sistema también se emplean sensores virtuales, aunque no se utiliza la codificación binaria (combinaciones de 1 y 0) aplicada anteriormente. Esto se debe a que, al tratarse de movimientos tanto verticales como laterales, el uso de combinaciones podría generar problemas en la detección, lo que provocaría que el dron se desvíe de la trayectoria, además que en este caso no se aplicaran movimientos de rotación en el dron.

En su lugar, se implementa un enfoque basado en fases, que consiste en analizar dos franjas horizontales dentro de la imagen captada por la cámara del dron: una franja alta y la otra baja, como se muestra en la Figura 57. Cada franja se divide en tres zonas iguales simulando la presencia de sensores virtuales en las posiciones izquierda, centro y derecha.

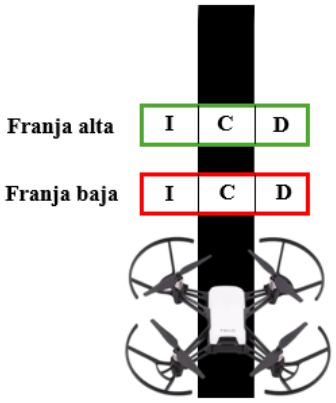


Figura 57: Visualización de franjas alta y baja.
Fuente: Autoría propia

El funcionamiento de los sensores virtuales se basa en la cantidad de pixeles detectados en cada una de las zonas. En el primer caso, cuando en la franja alta el sensor central registra la mayor concentración de píxeles correspondientes a la trayectoria, como se muestra en la Figura 58, se interpreta que el dron debe iniciar la fase de subida. En respuesta, el dron se desplazará verticalmente hacia arriba, siguiendo la línea detectada.

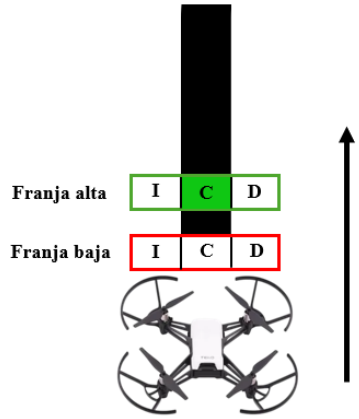


Figura 58: Movimiento vertical hacia arriba en respuesta al sensor central detectado en franja alta.

Fuente: Autoría propia

El sistema guarda la dirección previa del dron, lo cual permite evitar errores en la interpretación de la trayectoria. Por ejemplo, si el dron se encuentra en la fase de subida y, simultáneamente, detecta trayectoria en el sensor centrar de la franja baja, no cambiará de fase ni comenzará a descender por error; en su lugar, continuará subiendo de forma normal.

En el segundo caso, cuando el dron comienza a aproximarse a una curva, ya sea hacia la izquierda o derecha, como se muestra en la Figura 59, el sistema cambia a fase correspondiente de curva. Esta transición ocurre cuando el sensor izquierdo o derecho de la franja alta detecta una mayor cantidad de píxeles, lo que indica una desviación lateral en la trayectoria. En respuesta, el dron se moverá lateralmente en la dirección correspondiente para seguir la curva correctamente.

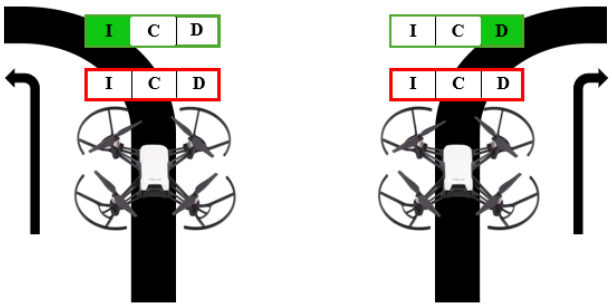


Figura 59: Detección de sensor izquierdo o derecho para realizar cambio de fase a curva.

Fuente: Autoría propia

En el tercer caso, correspondiente a la fase de bajada, el dron debe detectar una mayor concentración de píxeles en la franja baja de la imagen. Esto ocurre típicamente después de completar una curva, como se explicó en el caso anterior. Cuando el dron se encuentra desplazándose lateralmente en una curva y empieza a registrar trayectoria en la franja baja, se interpreta que debe iniciar el descenso para seguir la línea ubicada en la parte inferior.

La activación especifica de los sensores en la franja baja dependerá de la dirección desde la cual el dron proviene. Por ejemplo, si viene de una curva hacia la izquierda, la trayectoria podría ser detectada por el sensor central y posiblemente el derecho; mientras que, si proviene de una curva hacia la derecha, los sensores izquierdo y central serían los que registren mayor cantidad de píxeles. Esta situación se ilustra en la Figura 60, y permite al sistema cambiar a la fase de bajada de forma adecuada.

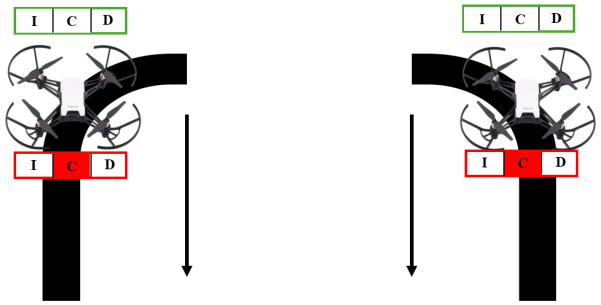


Figura 60: Detección de franja baja para iniciar fase de bajada.
Fuente: Autoría propia

2.3.2. Diseño e implementación del software

En este apartado, se analizarán los algoritmos que se implementarán en el desarrollo del sistema de navegación. El primer paso consiste en desarrollar un programa que permita ajustar los parámetros del modelo de color HSV, con el objetivo de optimizar la visualización de la imagen y facilitar la detección precisa de la trayectoria, la cual, en este caso, es de color negro. Una vez completado el ajuste, los valores mínimos y máximos obtenidos se utilizarán como umbrales en el sistema de seguimiento horizontal y vertical. El procedimiento para implementar el sistema se presenta en el diagrama de bloques de la Figura 61.

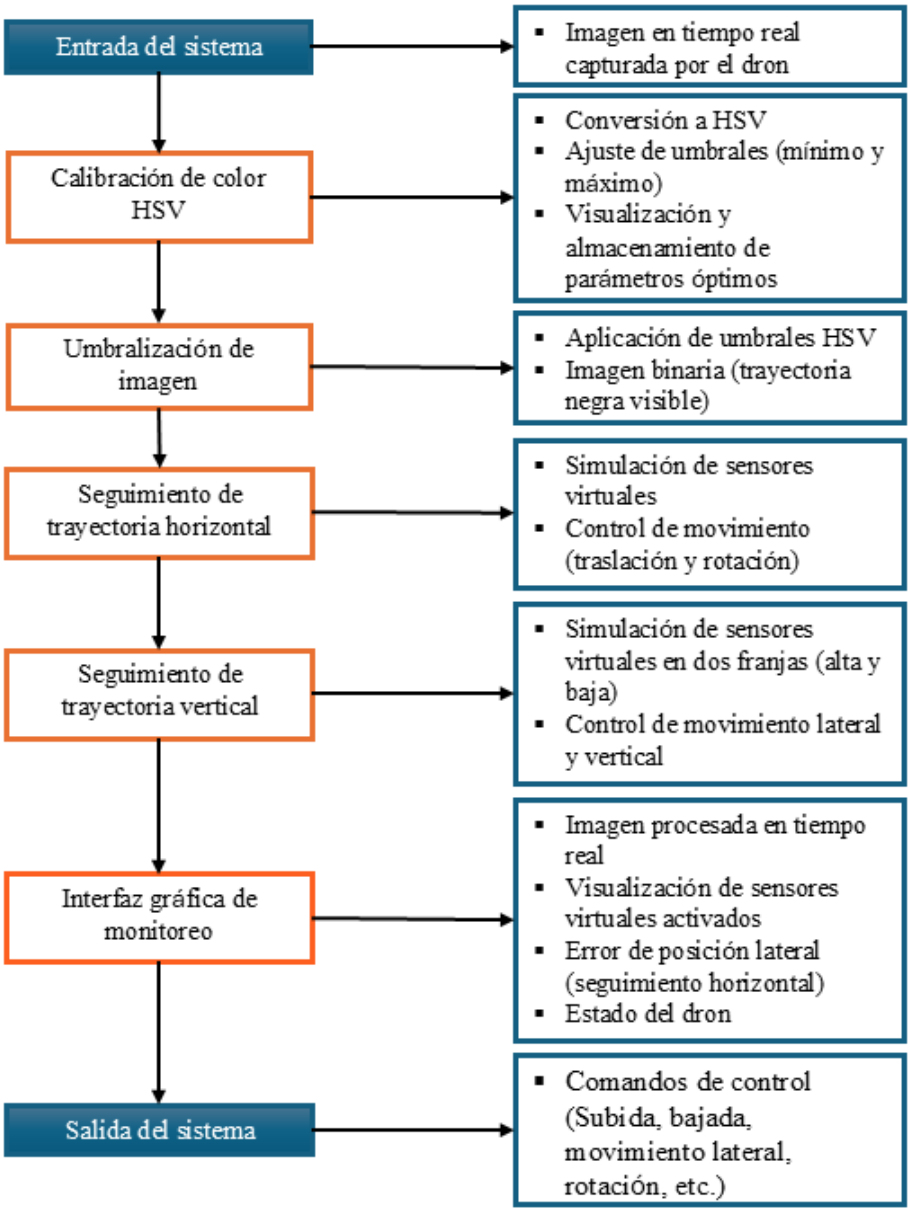


Figura 61: Diagrama de bloques del sistema de navegación.
Fuente: Autoría propia

2.3.2.1 Instalación de librerías del software Python

Para el desarrollo de los programas, es necesario instalar las librerías mostradas en la Figura 62, los comandos y funciones ya han sido verificados previamente en el apartado de componentes lógicos. Cabe destacar, que en la figura también se incluyen algunas librerías que vienen preinstaladas por defecto en el entorno de desarrollo, las cuales no interesan usarse para este proyecto.

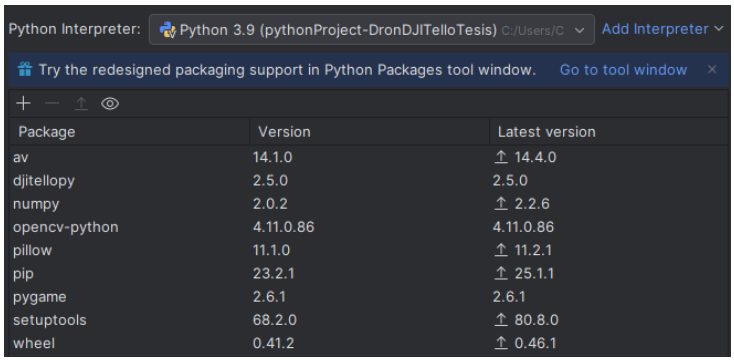


Figura 62: Librerías utilizadas en PyCharm.
Fuente: Autoría propia

2.3.2.2 Algoritmo de detección de colores mediante HSV

El desarrollo de este algoritmo tiene como objetivo permitir la visualización simultánea de tres imágenes durante la ejecución: la imagen original captada por la cámara del dron, la imagen umbralizada y el resultado final de la segmentación. Además, se incorpora una ventana interactiva con barras deslizantes (trackbars) que permiten ajustar en tiempo real los parámetros de tonalidad, saturación y brillo (HSV), definiendo valores mínimos y máximos. A medida que se modifican estos valores, es posible observar cómo se segmenta la imagen umbralizada, resaltando únicamente la trayectoria de color negro y descartando el resto del contorno. En la figura 63, se presenta el diagrama de flujo seguido para la implementación del primer programa.

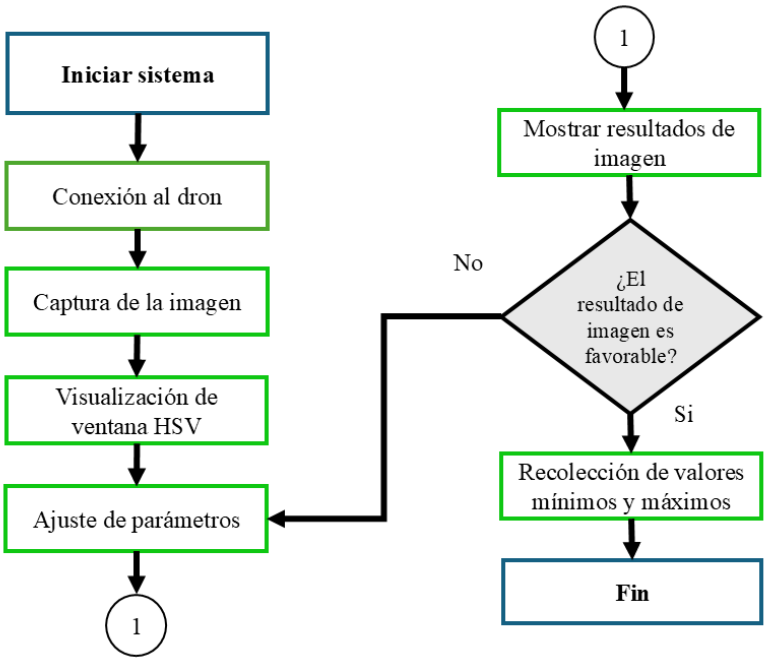


Figura 63: Diagrama de flujo para detección de colores.
Fuente: Autoría propia

Configuración inicial de variables

Para iniciar el sistema y realizar la conexión, lo primero que se realiza es definir el ancho y alto de la imagen que se va a captar por el dron, se recomienda que los valores sean de 480 de ancho y 360 de alto para agilizar el procesamiento. Además, se inicializa el dron y para ello se debe aplicar los siguientes comandos que se muestran en la Figura 64 para realizar la conexión con el ordenador y acceder a la cámara por medio del software. También se verifica el porcentaje de batería, ya que si el dron cuenta con un porcentaje bajo es probable que se apague durante la ejecución.

```
# Dimensiones
Ancho, Alto = 480, 360

# Inicializar dron
DRON = tello.Tello()
DRON.connect()
DRON.streamon()
print("Batería:", DRON.get_battery())
```

Figura 64: Inicialización del dron.
Fuente: Autoría propia

Diseño de ventana de ajuste de parámetros HSV

A continuación, se crea una ventana con el nombre "Controles HSV" con el comando "cv2.nameWindow", la cual se redimensiona a 640 de ancho y 240 de alto con "cv2.resizeWindow".

Para crear las barras deslizantes se hace uso de la función "cv2.createTrackbar", seguido del nombre que se le da a cada barra para posteriormente colocar en donde van a estar situadas, en este caso en la misma ventana creada con el nombre de "Controles HSV", y por último el rango en el que se van a poder desplazar.

Un punto importante para tener en cuenta es que los parámetros del modelo de color se representan de forma diferente en OpenCV respecto a su definición analítica, ya que la tonalidad se expresa en un rango de 0°a 360°, y tanto la saturación como el brillo varían de 0 a 100 %. Sin embargo, dado que OpenCV trabaja con valores de 8 bits (es decir, 0 a 255), se requiere una adaptación de estos rangos. En el caso de la tonalidad, se divide los valores por 2, limitando el rango 0–180°, lo cual es suficiente para las necesidades de percepción visual en aplicaciones informáticas. Por otro lado, la saturación y el brillo se escalan directamente al rango de 0 a 255, lo que permite representarlos adecuadamente en 8 bits, esto se visualiza en la Figura 65.

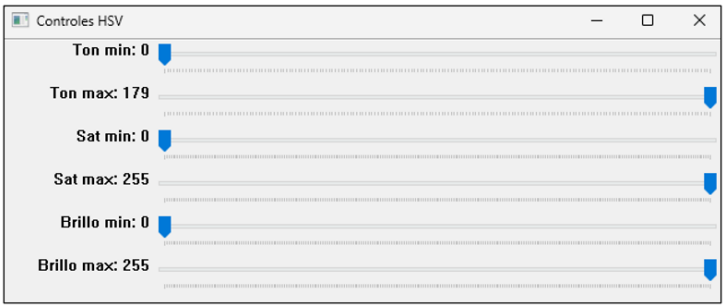


Figura 65: Creación de barras deslizantes.
Fuente: Autoría propia

Para obtener el valor actual de cada barra deslizante, se utiliza la función "cv2.getTrackbarPos", especificando el nombre individual de la barra y el nombre de la ventana a la que pertenece.

Filtrado de imagen mediante máscara HSV

Mediante la función "cv2.inRange", se filtra la imagen en formato HSV utilizando los rangos de valores definidos por las barras deslizantes. A continuación, se crea una máscara sobre la imagen original permitiendo visualizar únicamente los píxeles cuyo color se encuentra dentro del rango especificado. Esto ayuda a eliminar brillos o colores no deseados que puedan interferir con la detección del dron. Este proceso se muestra en la Figura 66. Finalmente, se realiza una nueva conversión de color para visualizar correctamente el resultado de las tres imágenes generadas.

```
# Aplicar máscara
mascara = cv2.inRange(imagen_hsv, menor, mayor)
resultado = cv2.bitwise_and(imagen, imagen, mask=mascara)

# Convertir máscara a BGR para mostrarla a color
mascara_color = cv2.cvtColor(mascara, cv2.COLOR_GRAY2BGR)
```

Figura 66: Segmentación por color HSV utilizando máscaras en OpenCV.
Fuente: Autoría propia

2.3.2.3 Diseño de interfaz gráfica

Para el diseño de la interfaz, se implementó una función específica que permite superponer una etiqueta de texto en la parte superior de una imagen, tal como se muestra en la Figura 67. La función inicia generando una

copia de la imagen original con el fin de evitar modificaciones directas sobre los datos de entrada. Posteriormente, se dibuja un rectángulo de color gris oscuro y relleno sólido en la parte superior de la imagen, abarcando desde la esquina superior izquierda hasta el borde derecho, con una altura fija de 30 píxeles. Este rectángulo actúa como fondo para el texto. Sobre esta base, se aplica la cadena de texto deseada utilizando una fuente legible y color blanco, garantizando una adecuada visibilidad y contraste con el fondo. Además, se definen los títulos que se desean mostrar sobre cada imagen: "Imagen Original", "Mascara HSV" y "Resultado Final, con el objetivo de visualizar estas tres imágenes en una única ventana dispuestas horizontalmente.

Por último, para visualizar en una sola ventana las imágenes anteriormente procesadas, se utiliza el comando "cv2.inshow". Este comando recibe como parámetros el nombre de la ventana y la imagen con el fondo oscuro donde se integraron las tres vistas.

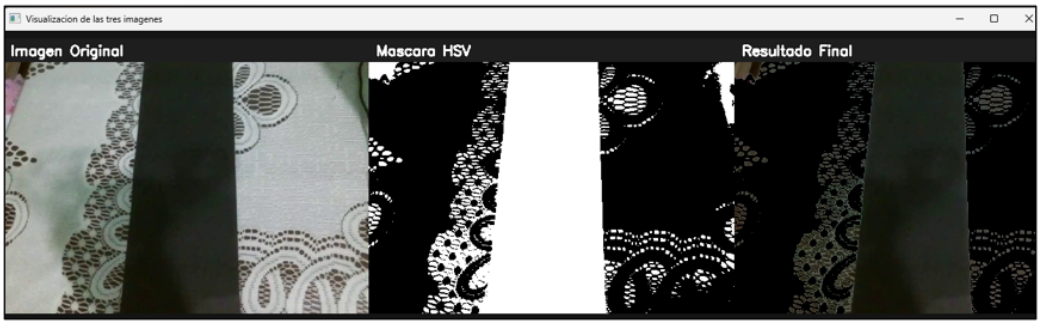


Figura 67: Visualización de las tres imágenes.
Fuente: Autoría propia

El diseño final de la interfaz quedaría como se muestra en la Figura 68. En la parte superior de la ventana, se visualizan las tres imágenes principales: la imagen original, la máscara HSV y el resultado final, lo que permite una comparación clara entre cada etapa del procesamiento. En la esquina inferior derecha se encuentra la ventana denominada "Controles HSV", que contiene seis barras deslizantes para ajustar los valores mínimos y máximos de tonalidad, saturación y brillo. Por otro lado, en la parte inferior izquierda se presenta la salida por consola, donde se imprimen los valores actuales de dichos controles, facilitando su monitoreo en tiempo real.

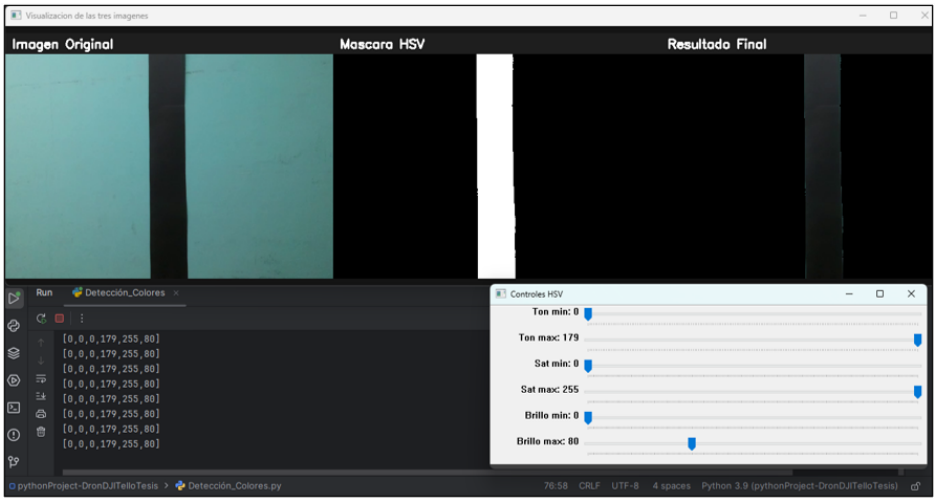


Figura 68: Interfaz para el ajuste de parámetros del modelo de color HSV.
Fuente: Autoría propia

2.3.2.4 Algoritmo de navegación para seguimiento de trayectoria horizontal

El sistema de navegación para seguimiento de trayectoria horizontal se basa en un esquema de seguimiento por sensores virtuales, implementado mediante el procesamiento de la imagen umbralizada captada por su cámara. Esta imagen se divide en tres regiones de igual tamaño, correspondientes a los sensores virtuales izquierdo, central y derecho, en las cuales se asignan valores binarios (1 o 0) según la detección de la línea de referencia. A partir de estas combinaciones binarias, el dron ejecuta movimientos de rotación y traslación conforme a las condiciones establecidas en la Tabla 5.

Adicionalmente, se desarrolla una interfaz gráfica que permite visualizar en tiempo real el estado actual del dron dentro del entorno de navegación. Esta interfaz incluye la representación de parámetros relevantes como la métrica de detección de presencia de línea por división de imagen y el error de posición, que indica la desviación lateral del dron respecto a la trayectoria. La Figura 69 presenta el diagrama de flujo correspondiente al diseño e implementación de este segundo programa.

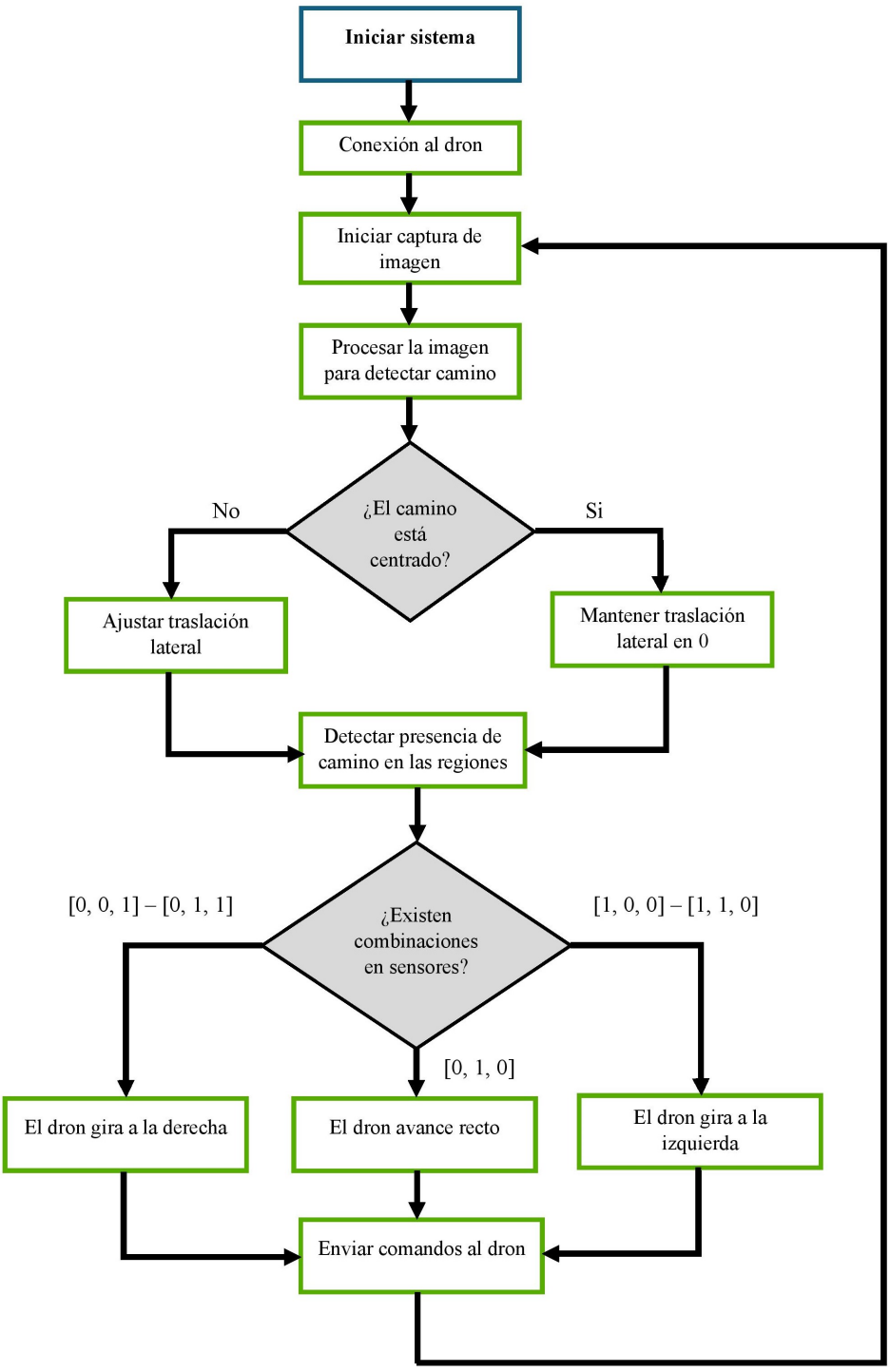


Figura 69: Diagrama de flujo para seguimiento horizontal.
Fuente: Autoría propia

Definición de parámetros de sistema

Para la inicialización del sistema, es necesario definir un conjunto de variables importantes, tal como se muestra en la Figura 70. Entre estas variables se encuentran las dimensiones de la imagen, así como valores mínimos y máximos del modelo de color HSV, los cuales han sido determinados previamente a partir del programa de ajuste de parámetros. Así mismo, se configura el número de sensores virtuales que permitirán segmentar la imagen umbralizada en regiones de interés.

Se establece un umbral que determina la cantidad mínima de píxeles detectados en una región para considerar que un sensor ha sido activado. También se define una lista de pesos asociados a cada sensor, conocida como "peso valor", que representa los grados de rotación que el dron debe aplicar según la posición relativa de la línea detectada.

La velocidad de avance del dron se configura en 10 unidades, valor elegido empíricamente para permitir una respuesta más eficiente al trazar curvas. Finalmente, se incluye una variable de sensibilidad, la cual ajusta el grado de agresividad en el movimiento lateral del dron, permitiendo un control más fino según las condiciones de seguimiento de trayectoria.

```
# Parámetros de imagen
ancho, alto = 480, 360
valores_hsv = [0,0,0,179,255,75]  # HSV para línea negra
sensores = 3
umbral = 0.2
pesoValor = [-30, -20, 0, 20, 30]  # Rotación según combinación de sensores
velAvance = 10
sensibilidad = 4  # Más alto = menos brusco el movimiento lateral
```

Figura 70: Variables establecidas.
Fuente: Autoría propia

Segmentación de la imagen umbralizada en regiones de interés

Con base en los parámetros previamente definidos, se implementa una función denominada "umbralizado", cuya finalidad es almacenar los valores HSV especificados en la Figura 70 y aplicar una máscara sobre la imagen utilizando los rangos establecidos. Esta operación permite aislar la región en interés correspondiente a la trayectoria.

Para obtener la salida de los sensores virtuales, se define una segunda función encargada de segmentar la imagen umbralizada en tres regiones iguales, mediante el uso del comando "np.hsplit", el cual recibe como argumentos

la imagen binaria y la cantidad de sensores definidos. En cada una de las regiones resultantes se realiza un conteo de píxeles activados (valor blanco). Si la cantidad de píxeles en una región supera el umbral preestablecido, se asigna un valor de salida de 1; en caso contrario se asigna un 0. Los resultados binarios obtenidos se imprimen en la consola del software, tal como se muestra en la Figura 71.

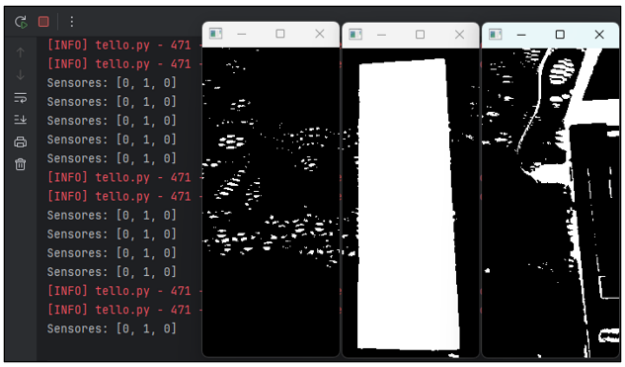


Figura 71: Detección de trayectoria por medio de sensores.
Fuente: Autoría propia

Detección del centro de la trayectoria

En esta etapa del desarrollo del sistema, se requiere identificar el centro de la trayectoria detectada dentro de la imagen procesada. Para este propósito, se implementa una función denominada "detectar centro", la cual emplea el comando "cv2.findContours" para localizar los contornos presentes en la imagen binaria resultante del umbralizado.

Una vez detectados, los contornos se visualizan mediante otra función, que permite su representación gráfica sobre la imagen original o procesada. Posteriormente, para delimitar cada contorno, se utiliza la función "cv2.boundingRect", la cual genera un rectángulo que permite calcular su centro geométrico.

Esta función retorna coordenadas de la esquina superior izquierda del rectángulo (x, y), así como su ancho w. y alto h. Con estos valores, se determina el centro del rectángulo, y, por ende, el centro de la trayectoria aplicando las siguientes expresiones matemáticas:

$$\text{Centro}_x = x + \frac{w}{2} \tag{2.17}$$

$$\text{Centro}_y = y + \frac{h}{2} \tag{2.18}$$

Con el objetivo de visualizar gráficamente el centro de la trayectoria detectada, se incorpora un marcador en forma de círculo de color azul, utilizando la función "cv2.circle". Esta función se aplica sobre la imagen original capturada por el dron, especificando como parámetros las coordenadas previamente calculadas del centro (x, y), el radio del círculo, el color en formato BGR y el grosor del contorno. Esta representación facilita el análisis visual del seguimiento de la trayectoria, como se muestra en la Figura 72.

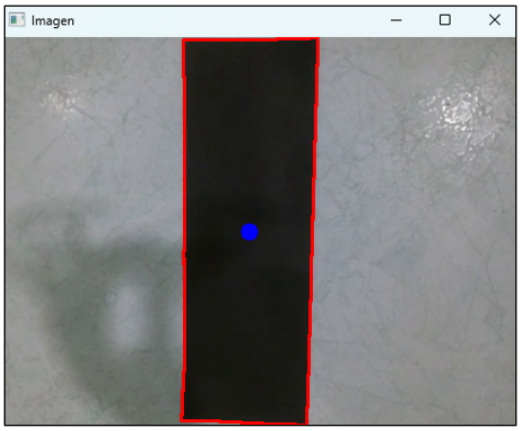


Figura 72: Detección del centro de trayectoria.
Fuente: Autoría propia

Estrategia de control para la corrección de posición lateral

En esta sección se desarrolla la lógica de movimiento del dron, abarcando tanto la traslación como la rotación. En particular, para la traslación, es fundamental calcular el error de posición lateral, el cual representa la desviación del dron respecto al centro de la trayectoria detectada.

Durante el despegue o en condiciones de iluminación deficiente, pueden producirse movimientos abruptos o errores en la detección de la línea, lo que provoca que el dron se desplace fuera de la trayectoria ideal. Para mitigar este efecto y corregir la posición del dron el tiempo real, se emplea una expresión matemática que permite cuantificar dicho error. Esta métrica es esencial para ajustar los comandos de control y mantener el dron alineado con la trayectoria esperada.

El error de posición lateral se calcula mediante la diferencia entre la coordenada horizontal del centro de la trayectoria detectada y el punto medio del ancho de la imagen, como se muestra en la siguiente expresión:

$$\text{error} = \text{Centro}_x - \frac{\text{ancho}}{2} \tag{2.19}$$

Este valor representa que tan alejado se encuentra el dron respecto al centro de la imagen, asumiendo que dicho centro corresponde a la trayectoria ideal.

Para transformar este error en una señal de control interpretable, se aplica el factor de sensibilidad visto en los parámetros del sistema, que regula la magnitud de la corrección. El resultado se limita a un rango comprendido entre -15 y 15 mediante la función "np.clip", obteniendo así el valor de movimiento lateral (positivo hacia la derecha, negativo hacia la izquierda). En el código esta expresión se realizó de la siguiente manera:

$$\text{izq\_derech} = \text{int}\left(\text{clip}\left(\frac{\text{error}}{\text{sensibilidad}},\ -15,\ 15\right)\right) \tag{2.20}$$

Adicionalmente, para evitar oscilaciones menores o movimientos innecesarios cerca del centro, se establece una zona muerta: si el valor de ``izq_derech'' se encuentra dentro del intervalo $(-2, 2)$, se fuerza a cero, desactivando el movimiento lateral:

$$\text{si}\ \ -2 < \text{izq\_derech} < 2, \quad \text{entonces} \quad \text{izq\_derech} = 0 \tag{2.21}$$

Este enfoque permite al sistema realizar correcciones precisas y estables en la posición del dron respecto a la trayectoria deseada.

Implementación de control de rotación en el dron

El control de rotación del dron se basa en la lectura de sensores virtuales, los cuales simulan una segmentación de la imagen umbralizada en tres regiones: izquierda, centro y derecha. A partir del análisis de la cantidad de píxeles detectados en cada región, se genera una lista binaria de tres elementos, donde el valor 1 indica la presencia significativa de línea negra y 0 su ausencia.

Esta lista, denominada "envió" en el código, es evaluada por una estructura condicional que asigna un valor de rotación a partir de una tabla de pesos definida como "Peso valor". Esta tabla contiene valores enteros que representan el ángulo o intensidad del giro necesario para que el dron se reoriente correctamente hacia la trayectoria. La lógica aplicada es la siguiente:

- Si la línea se encuentra principalmente a la derecha $[0, 0, 1]$, se asigna un valor de giro positivo elevado (pesoValor[4]), lo que implica una rotación hacia la derecha.

- Si la línea se encuentra al centro $[0, 1, 0]$, se mantiene la orientación actual (pesoValor[2]), usualmente cero.
- Si la línea se encuentra entre el centro y la derecha $[0, 1, 1]$, se aplica una rotación moderada hacia la derecha (pesoValor[3]).
- Si la línea se encuentra a la izquierda $[1, 0, 0]$, se asigna un giro negativo fuerte (pesoValor[0]), indicando rotación hacia la izquierda.
- Si se encuentra entre la izquierda y el centro $[1, 1, 0]$, se aplica una rotación moderada hacia la izquierda con (pesoValor[1]).
- En cualquier otro caso (por ejemplo, ausencia total de línea), se establece la rotación en cero, deteniendo el giro momentáneamente.

Esta lógica permite al dron corregir su orientación el tiempo real, manteniéndose alineado con la trayectoria en función de la distribución espacial de los píxeles detectados. La estructura es sencilla pero efectiva para entornos controlados, donde la línea a seguir es claramente distinguible en la imagen umbralizada.

2.3.2.5 Diseño de interfaz gráfica para trayectorias horizontales

Para la visualización del seguimiento horizontal de la trayectoria, se desarrolló una interfaz gráfica basada en la generación de un lienzo principal utilizando matrices "Numpy". Este lienzo actúa como contenedor de los distintos elementos visuales, entre ellos el menú de opciones, las imágenes procesadas, los sensores virtuales y los textos informativos.

La altura del lienzo ‘‘`lienzo_height`’’ se define considerando varios componentes: un espacio superior reservado para el título, el alto de la imagen principal, una sección intermedia destinada a la visualización de los sensores y un margen inferior. Por su parte, el ancho del lienzo ‘‘`lienzo_width`’’, permite mostrar simultáneamente dos imágenes separadas por un margen intermedio. En la parte superior del lienzo se incorpora un título descriptivo, centrado horizontalmente. Este se dibuja utilizando una fuente legible y grosor adecuado, complementado con un efecto de sombra negra que mejora la visibilidad sobre el fondo oscuro. Para lograr una alineación precisa, se calcula previamente el tamaño del texto (ancho y altura) mediante la función "cv2.getTextSize", lo que permite posicionarlo correctamente en coordenadas (x, y) dentro del lienzo. El diseño de la interfaz se muestra en la Figura 73.

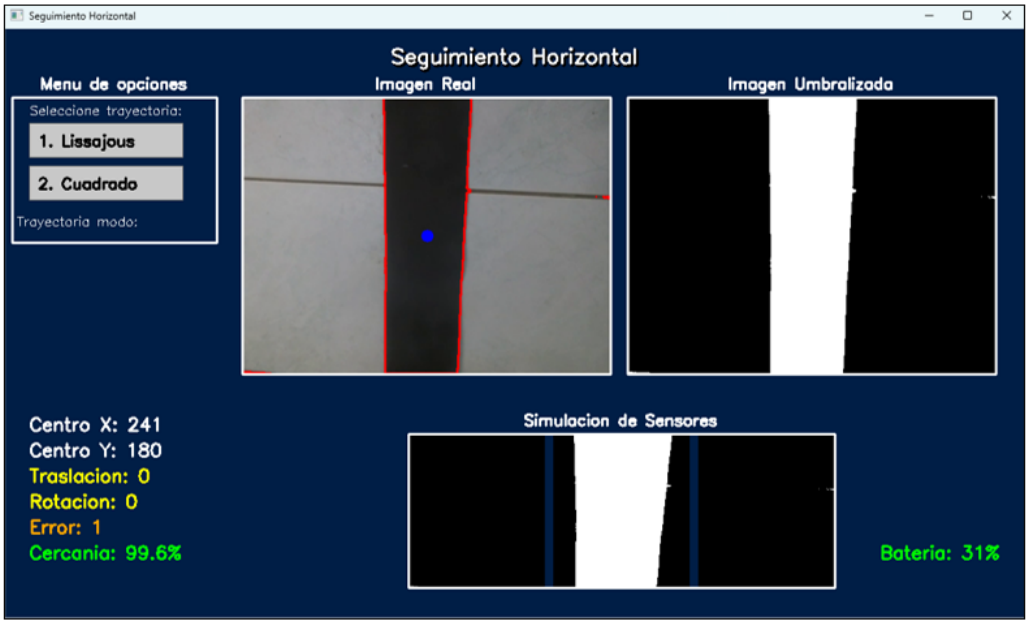


Figura 73: Interfaz gráfica para el seguimiento horizontal.
Fuente: Autoría propia

Diseño de menú de selección de trayectoria

En la parte superior izquierda de la interfaz se muestra el menú de opciones, que permite seleccionar la trayectoria a seguir por el dron. Este menú se genera mediante funciones de OpenCV y consta de un recuadro con fondo oscuro, borde blanco y un título centrado denominado “Menú de opciones”. Debajo se incluye el texto “Seleccione trayectoria”, seguido de una lista numerada con las opciones disponibles: 1. Infinito, 2. Cuadrado y 3. Triángulo.

Cada opción se presenta dentro de un recuadro con su número y nombre. Si una opción esta seleccionada, se resalta con un fondo y borde en color verde; en caso contrario, se mantiene en tono gris. Al final del menú se muestra el texto “Trayectoria modo:”, indicando la opción activa. La posición y espaciado de todos los elementos se calcula dinámicamente para asegurar una presentación ordenada y clara. El diseño del menú se muestra en la Figura 74.

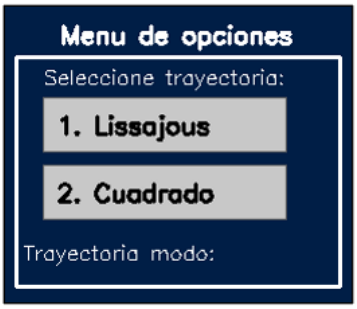


Figura 74: Menú de selección de trayectoria en la interfaz gráfica.
Fuente: Autoría propia

Incorporación de imágenes y etiquetado informativo en la interfaz

Para mejorar la visualización del sistema de seguimiento de trayectoria, se integraron en la interfaz dos imágenes principales: la imagen real capturada por el dron y su versión umbralizada. Ambas se muestran en el lienzo principal con sus respectivos títulos "Imagen Real" e "Imagen Umbralizada", centrados horizontalmente mediante el cálculo del ancho del texto. Estas etiquetas se colocan usando la función "cv2.putText", y permiten al usuario identificar de forma clara el tipo de imagen observada.

Las imágenes se presentan una al lado de la otra y están delimitadas con marcos blancos generados por la función "cv2.rectangle", lo que facilita su distinción visual. Esta disposición permite monitorear simultáneamente tanto la entrada visual original como el resultado del procesamiento de imagen, lo cual es esencial para evaluar el desempeño del sistema de detección en tiempo real, esto se visualiza en la Figura 75.

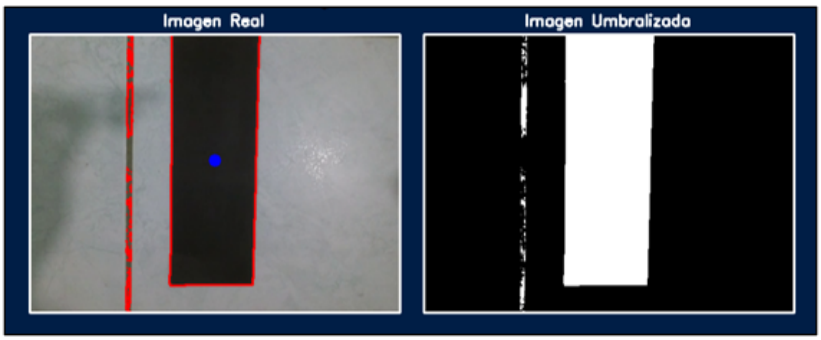


Figura 75: Visualización de las dos imágenes dentro de la interfaz
Fuente: Autoría propia

Visualización gráfica de la segmentación sensorial

Con el fin de representar gráficamente el funcionamiento de los sensores virtuales, se dividió la imagen umbralizada en tres regiones utilizando el método "np.hsplit". A continuación, cada una de estas divisiones se le aplicó un formato de color con "cv2.cvtColor" y se redimensionaron para mejorar la visualización. Posteriormente, las tres regiones fueron organizadas horizontalmente en el lienzo principal, separadas por divisores del mismo color de fondo, simulando así el comportamiento de sensores en tiempo real. Finalmente, se incorporó un marco blanco y un título superior descriptivo con el texto "Simulación de Sensores", como se aprecia en la Figura 76.

Figura 76: División de la imagen umbralizada integrada en la interfaz gráfica.
Fuente: Autoría propia

Monitoreo del error de alineación y acciones del dron

Para brindar información detallada sobre el seguimiento del dron en tiempo real, se implementó una sección en la interfaz que muestra valores claves relacionados con su posición. Primero, se calcula el error de posición horizontal como la diferencia entre el centro de la trayectoria y el centro de la imagen, calculado en la ecuación (2.19). A partir de este error, se obtiene un porcentaje de cercanía, que indica que tan alineado está el dron respecto al centro, expresado en una escala del 0 al 100 %. Estos datos, junto con el valor del centro en (x, y), así como los comandos de traslación y rotación enviados, se visualizan en la Figura 77, mediante la función "cv2.putText". De esta manera, el usuario puede monitorear el comportamiento del dron y su alineación con la trayectoria en todo momento.

Para calcular el porcentaje de cercanía entre el dron y el centro de la trayectoria, se emplea la siguiente expresión:

$$\text{Porcentaje de cercanía} = \text{máx}\left(0, \left(1 - \frac{|error|}{error_{\text{máx}}}\right)\right) \times 100 \qquad (2.22)$$

Esta fórmula garantiza que el resultado no sea negativo y proporciona un valor porcentual que indica que tan cerca está el dron del centro ideal de la trayectoria.

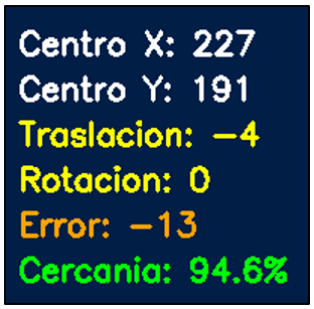


Figura 77: Información visual del seguimiento y desplazamiento del dron.
Fuente: Autoría propia

2.3.2.6 Algoritmo de navegación para el seguimiento de trayectoria vertical

El sistema de navegación diseñado para el seguimiento de trayectoria vertical se fundamenta en la simulación de sensores virtuales. A diferencia del enfoque descrito en el apartado anterior, en esta etapa no se emplean combinaciones binarias, ya que no es necesario que el dron realice rotaciones durante su desplazamiento. Esto se debe a que cualquier giro podría ocasionar la pérdida de visualización de la trayectoria en la pared. En este algoritmo, el dron será capaz de seguir la trayectoria mediante movimientos verticales y horizontales, lo que le permitirá desplazarse adecuadamente a lo largo del recorrido.

Para lograr este comportamiento, la imagen capturada se divide en dos franjas horizontales (alta y baja), y cada una de estas se segmenta en tres zonas de igual tamaño, simulando sensores virtuales, izquierdo, central y derecho. El dron tomará decisiones de movimiento en función del sensor que detecte una mayor concentración de píxeles correspondientes a la trayectoria, permitiendo la navegación autónoma. El diagrama de flujo correspondiente a este algoritmo se presenta en la Figura 78.

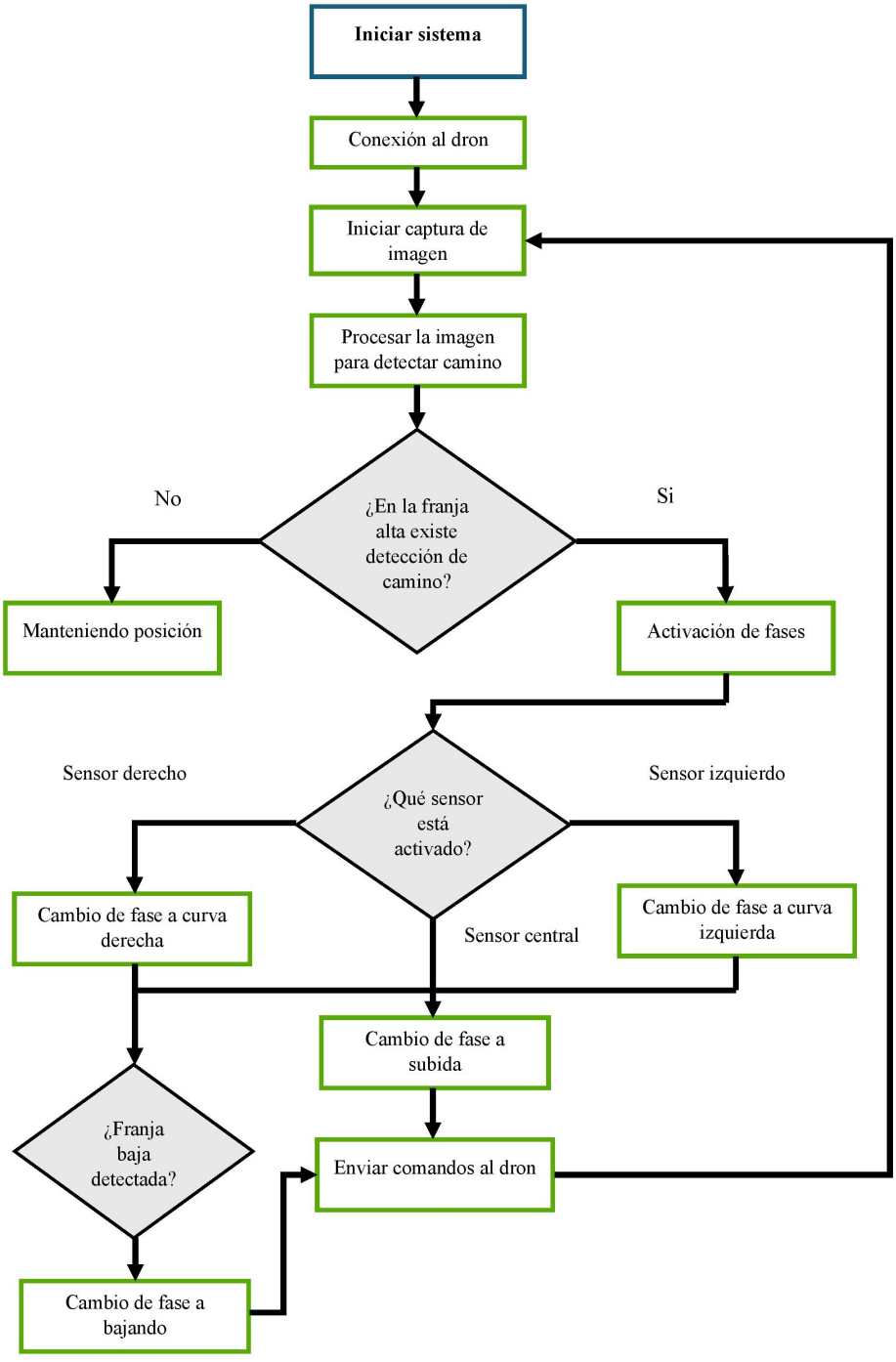


Figura 78: Diagrama de flujo para seguimiento vertical.
Fuente: Autoría propia

Configuración inicial del sistema

Al igual que en el seguimiento horizontal, se establecen los parámetros iniciales del sistema, entre los cuales se incluyen la recolección de los valores HSV, las dimensiones de la imagen, así como las velocidades de desplazamiento vertical y lateral del dron. Este procedimiento se ilustra en la Figura 79.

```
# Parámetros
valores_hsv = [0,0,0,179,255,80]  # HSV para línea negra
ancho, alto = 480, 360
vel_lateral = 30
vel_vertical = 35 # Aumentado para bajar más fuerte
```

Figura 79: Parámetros iniciales del sistema.
Fuente: Autoría propia

Segmentación en HSV y análisis de franjas horizontales

En el procesamiento de la imagen, se implementa una función de umbralizado que convierte la imagen original del formato BGR al modelo de color HSV, con el fin de facilitar la detección de colores específicos. A partir de los valores HSV previamente definidos, se generan los umbrales mínimo y máximo, permitiendo así la segmentación de la imagen mediante la función "cv2.inRange", la cual produce una máscara binaria en la que se resaltan los píxeles que se encuentran dentro del rango especificado.

Posteriormente, se implementan sensores virtuales que operan sobre esta máscara. Para ello, la imagen se divide en dos franjas horizontales: una franja alta (ubicada entre las filas 80 y 139) y una franja inferior (entre las filas 250 y 309) como se muestra en la Figura 80. Cada una de estas franjas se subdivide en tres secciones verticales iguales (izquierda, centro y derecha), lo cual permite evaluar la distribución de píxeles blancos en cada zona, esto se visualiza en la Figura 81.

El sistema cuenta los píxeles blancos presentes en cada una de estas secciones utilizando la función "cv2.countNonZero", y almacena estos valores en un diccionario estructurado, donde se diferencian los conteos de la franja superior y la inferior. Esta información es esencial para la posterior toma de decisiones en el sistema de navegación del dron, ya que permite estimar la posición relativa de los elementos de interés dentro del campo visual.

Figura 80: División de las dos franjas horizontales.
Fuente: Autoría propia

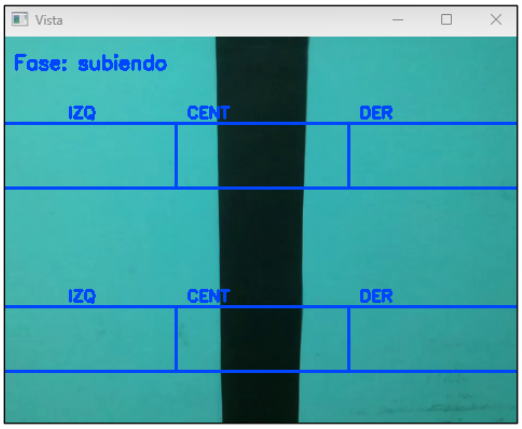


Figura 81: Visualización de las tres divisiones de sensores.
Fuente: Autoría propia

Estrategia de control basada en estados

La lógica de movimiento del dron se gestiona mediante una máquina de estados finita (FSM, por sus siglas en inglés), cuya implementación se encuentra en la función ‘‘`mover_dron`’’. Esta función recibe como entrada el estado actual del sistema (fase) y los valores de los sensores virtuales previamente calculados, y devuelve la siguiente fase, así como los comandos de movimiento vertical y lateral.

El sistema se basa en la lectura de sensores virtuales ubicados en las dos franjas anteriormente descritas. A partir de esta segmentación, se detecta la presencia de la línea de referencia y se infiere la dirección de movimiento necesaria para continuar con el trayecto. Este proceso se muestra en la Figura 82.

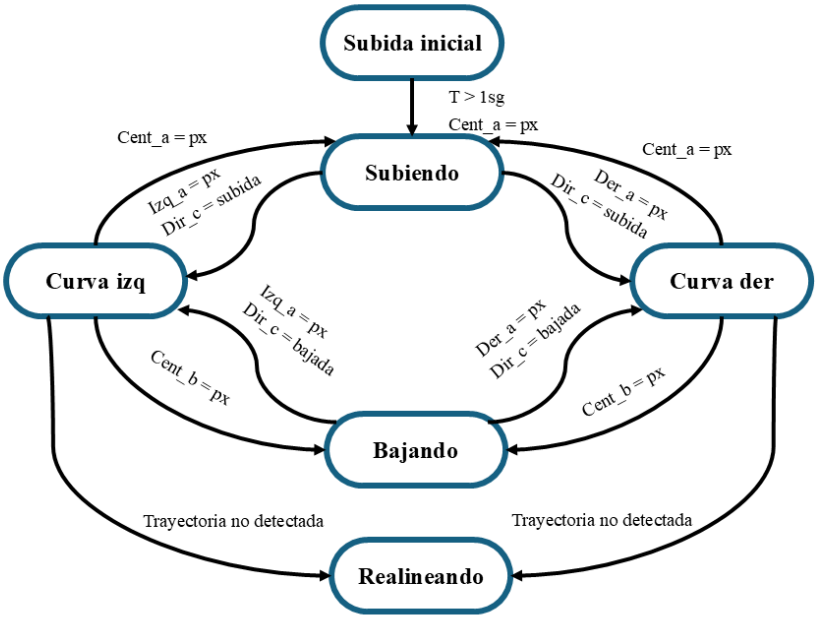


Figura 82: Diagrama de estados.
Fuente: Autoría propia

Las fases que conforman la lógica de navegación son:

- Subida inicial: El dron asciende sin considerar las lecturas de los sensores durante un breve intervalo de tiempo, con el fin de superar la zona de despegue inicial. Una vez transcurrido ese periodo, el sistema transita a la fase “subiendo”.
- Subiendo: Se analiza la franja superior para determinar si la línea se encuentra centrada, lo que indica un ascenso recto. Si se detecta mayor presencia de línea en la sección izquierda o derecha, se interpreta como una curva en esa dirección y se cambia de fase a “curva_izq” o “curva_der”, respectivamente.
- Curva izquierda / Curva derecha: Durante estas fases, el dron se desplaza lateralmente siguiendo la curva. Se evalúa la franja inferior para detectar la finalización de esta. Si se proviene de una fase de bajada, el sistema puede entrar en un estado de espera controlado antes de decidir si continúa subiendo o si necesita realinearse.
- Bajando: El dron desciende mientras se analiza la franja inferior. Si se detecta una línea significativa, el descenso continúa. Al finalizar, el

sistema evalúa nuevamente la franja superior para determinar si se debe entrar en una nueva curva o si se requiere un ajuste de posición.

- Realineado: En esta fase, el dron realiza una subida suave con correcciones laterales, determinadas por el desequilibrio entre las secciones izquierda y derecha de la franja superior. Si se detecta la línea centrada durante un tiempo suficiente, se retorna a la fase subiendo.

Este enfoque basado en fases permite un comportamiento adecuado en el dron frente a entornos con trayectorias de curvas cerradas, cambios de nivel y posibles pérdidas temporales de señal visual. En trayectorias con transiciones mas suaves, como curvas amplias o tramos diagonales, es posible combinar estas fases para generar movimientos continuos en diagonal, ya sea de ascenso o descenso, así como desplazamientos circulares.

2.3.2.7 Diseño de interfaz gráfica para trayectorias verticales

Con el objetivo de facilitar una visualización clara en tiempo real durante el seguimiento vertical de la trayectoria, se diseñó un lienzo gráfico específico para este propósito. Este lienzo posee dimensiones superiores a las del fotograma original capturado por la cámara del dron, lo cual permite incorporar información auxiliar adicional sin comprometer la visualización principal.

El fondo del lienzo se genera utilizando un arreglo de tipo “uint8” de “Numpy”, con un color azul marino como base. En la parte superior se incorpora un titulo descriptivo, centrado horizontalmente, con el texto “Seguimiento Vertical”. Sobre este lienzo se integran las imágenes principales y también el menú de opciones, mientras que, en la sección inferior se presenta el sistema de detección junto con el módulo de simulación de sensores. Este último esta organizado de acuerdo con las distintas franjas de percepción (alta y baja), permitiendo una interpretación clara y estructurada del entorno visualizado por el dron, tal como se muestra en la Figura 83. Cabe señalar que la visualización de la fase actual del dron, así como los rectángulos delimitadores de las franjas, se superpone sobre la imagen original. Esto se debe a que, en las versiones umbralizadas de la imagen, dichos elementos no serían claramente perceptibles.

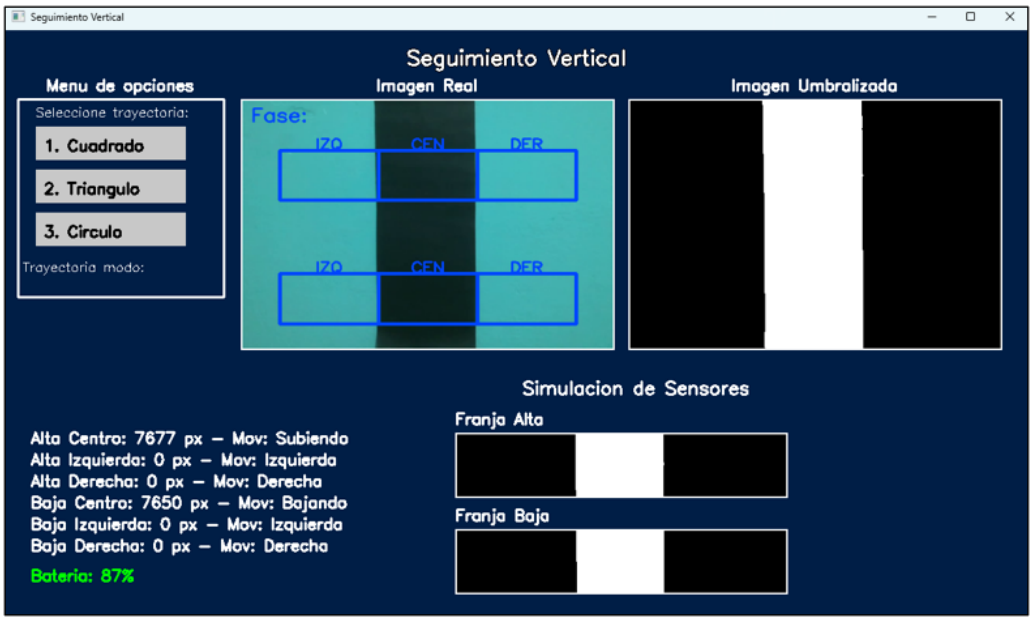


Figura 83: Interfaz gráfica para el seguimiento vertical.
Fuente: Autoría propia

Menú gráfico de selección de trayectoria

Se implementó un menú en la interfaz para facilitar la selección de la trayectoria, tal como se muestra en la Figura 84. Este menú incluye un titulo centrado, un recuadro de fondo gris oscuro con borde blanco y tres opciones disponibles: cuadrado, triangulo y circulo. Cada opción se muestra dentro de un recuadro con su etiqueta correspondiente, y la opción seleccionada se resalta en color verde para indicar el modo activo. Debajo del menú, se presenta un mensaje que indica la trayectoria actual en uso. Esta interfaz permite una selección visual clara e intuitiva durante la operación del dron.

Figura 84: Menú de opciones para selección de trayectoria.
Fuente: Autoría propia

Integración de imágenes en la interfaz gráfica

Con el fin de facilitar la supervisión del comportamiento del sistema de visión por computadora, se integran dos imágenes clave dentro del lienzo principal: la imagen real capturada por el dron y su versión umbralizada. Ambas se colocan en la parte superior del lienzo, alineadas horizontalmente y centradas con respecto al ancho total de la interfaz. Se deja un espacio de 20 píxeles entre ambas imágenes para mantener una separación visual clara.

Para posicionarlas adecuadamente, se calculan las coordenadas necesarias para centrar ambas imágenes en el eje horizontal, y se ubican a una altura fija desde el borde superior del lienzo. La imagen real se coloca a la izquierda y la imagen umbralizada a la derecha. Ambas imágenes se insertan directamente sobre el lienzo utilizando segmentación por regiones del arreglo de píxeles.

Encima de cada imagen, se añade una etiqueta descriptiva “Imagen Real” e “Imagen Umbralizada” utilizando el texto en color blanco. Estas etiquetas se centran horizontalmente con respecto a cada imagen correspondiente y se colocan ligeramente por encima de ellas para mantener una presentación ordenada y legible, tal como se muestra en la Figura 85.

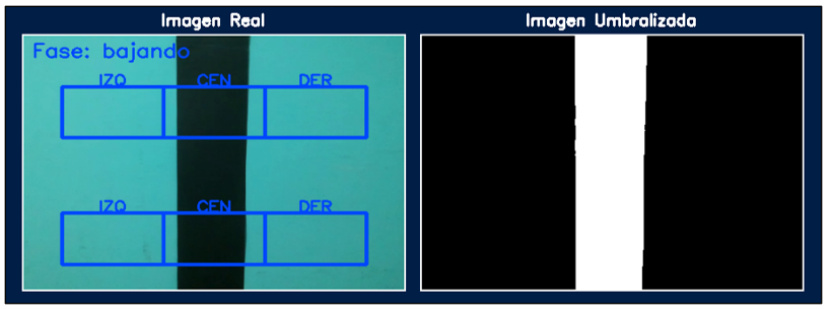


Figura 85: Visualización de las imágenes real y umbralizada en la interfaz gráfica.

Fuente: Autoría propia

Incorporación de franjas de detección en la interfaz gráfica

Para representar gráficamente el sistema de percepción del dron, se integran a la interfaz gráfica las dos franjas horizontales: “Franja Alta” y “Franja Baja”, previamente descritas. Cada franja se delimita mediante líneas que conforman un rectángulo, representando las áreas donde se aplican los sensores virtuales para la detección de la trayectoria, como se ilustra en la Figura 86.

Figura 86: Representación gráfica de las franjas “Alta” y “Baja”.
Fuente: Autoría propia

Supervisión visual del comportamiento del dron a través de sensores

La interfaz gráfica incluye información visual detallada que muestra los valores de píxeles detectados por los sensores virtuales ubicados en las franjas de percepción. Se representan específicamente los valores correspondientes a los sensores izquierdo, central y derecho, tanto en la franja alta como en la franja baja. Además, se visualiza de manera explícita la acción que debe ejecutar el dron en función de la detección realizada junto con el porcentaje actual de batería. Esta representación se muestra en la Figura 87.

Figura 87: Información visual de seguimiento vertical
Fuente: Autoría propia

3. Pruebas y resultados

En este apartado se presentan los resultados obtenidos a partir de las pruebas realizadas para el seguimiento de trayectoria, tanto en el eje horizontal como en el vertical. Estas pruebas fueron ejecutadas utilizando el entorno de desarrollo integrado Pycharm. Así mismo, se incluye la visualización de la interfaz gráfica implementada, la cual permite observar en tiempo real el comportamiento del dron y verificar el seguimiento de trayectoria establecido de manera eficiente.

3.1. Resultados

Se realizaron tres pruebas por cada tipo de trayectoria con el fin de obtener una base de datos que permitiera analizar el comportamiento del dron durante su recorrido. En el seguimiento horizontal, se llevaron a cabo pruebas con trayectorias en forma de curva de Lissajous y cuadrado. Por otro lado, en el seguimiento vertical se evaluaron trayectorias en formas de cuadrado, triángulo y círculo.

3.1.1. Pruebas reales en el seguimiento horizontal

Para la ejecución de las pruebas de seguimiento horizontal, se utilizó un espacio interior dentro de una sala que ofrecía un área moderadamente amplia, adecuada para disponer la trayectoria sobre el suelo. La trayectoria fue diseñada en color negro, con un ancho de 11cm, con el objetivo de facilitar la detección mediante la cámara del dron. Este ancho fue seleccionado para asegurar una correcta identificación y cálculo del centro de la línea, ya que un trazo más delgado dificultaría el proceso de detección.

Con el propósito de evaluar el desempeño del sistema, se implementó una recolección automática de métricas durante cada prueba. Entre las principales métricas registradas se encuentra el tiempo total de ejecución, que representa la duración completa de vuelo desde el inicio hasta el momento en que el dron termina de seguir la trayectoria. Se calculó también el promedio de cercanía, que cuantifica que tan centrado estuvo del dron respecto a la trayectoria detectada, expresado en porcentaje basado en la distancia relativa al centro del encuadre. Otra métrica clave es el promedio de error, el cual refleja en píxeles la desviación promedio entre la posición estimada de la trayectoria y el centro del dron en la imagen; este valor sirve como indicador directo de precisión en el seguimiento. Además, se registró el número de errores de seguimiento, definidos como instantes en los que el error superó

los 100 píxeles, y el número de correcciones laterales, contabilizadas cada vez que el dron ejecutó una acción de ajuste horizontal para mantenerse alineado. Finalmente, se estima la distancia total recorrida en centímetros basada en la cantidad de movimientos de avance del dron durante la prueba.

3.1.1.1 Trayectoria curva de Lissajous

Durante el desarrollo del algoritmo, basado en el seguimiento de línea convencional y procesamiento de imágenes, se observó que las trayectorias con curvas suaves o abiertas, como en el caso de la curva de Lissajous, el dron no requiere realizar giros bruscos. Debido a la naturaleza de estas curvas, es posible mantener una rotación gradual y continua. No obstante, para lograr un seguimiento estable, es fundamental que la velocidad de avance del dron no sea demasiado elevada, ya que un desplazamiento excesivamente rápido podría dificultar la detección precisa de la trayectoria y provocar errores en el seguimiento. La ejecución de esta prueba se observa en la Figura 88.

Figura 88: Seguimiento de trayectoria curva de Lissajous.
Fuente: Autoría propia

Con base en las métricas analizadas previamente, los resultados obtenidos de las tres pruebas correspondientes se presentan en la Tabla 6. Estos resultados permiten evaluar el desempeño del dron al seguir la trayectoria específica bajo condiciones controladas, proporcionando informacion clave sobre su estabilidad y capacidad de adaptación al entorno. El análisis de estos datos resulta fundamental para validar la efectividad del algoritmo implementado y determinar posibles áreas de mejora.

Tabla 6: Resultados obtenidos en la trayectoria curva de Lissajous

Prueba	**Tiempo (s)**	**Cercanía (%)**	**Error (px)**	**Err. > 100 px**	**Corr. lat.**	**Dist. (cm)**
1	82.21	87.91	29.03	59	5113	810.37
2	88.84	92.27	18.55	7	4674	876.77
3	106.67	94.84	12.38	4	500	1055.23

Fuente: Autoría propia

En la primera prueba, el dron completó su recorrido en un tiempo total de 82.21 segundos, logrando un promedio de cercanía del 87.91 % respecto a la trayectoria deseada. El promedio de error fue de 29.03 píxeles, lo que indica una desviación moderada en el seguimiento visual. Sin embargo, se registraron 59 errores mayores a 100 píxeles, junto con un número considerable de 5113 correcciones laterales, lo que sugiere que el sistema de control aún estaba ajustándose para mantener la estabilidad en el trayecto. La distancia estimada recorrida durante esta prueba fue de 810.37 cm.

En la segunda ejecución, se observó una mejora significativa en la precisión del sistema. El tiempo total aumentó ligeramente a 88.84 segundos, mientras que el promedio de cercanía ascendió al 92.27 %, reflejando un mejor alineamiento con la trayectoria. El promedio de error disminuyó a 18.55 píxeles, y los errores críticos mayores a 100 píxeles se redujeron a 7. Así mismo, las correcciones laterales bajaron a 4674, y la distancia estimada fue de 876.77 cm, lo que sugiere un vuelo más estable y eficiente.

La tercera prueba mostró el mejor desempeño general del sistema. El dron permaneció en vuelo por 106.67 segundos, con un promedio de cercanía del 94.84 % y un promedio de error de solo 12.38 píxeles, indicando una alta precisión en el seguimiento de la trayectoria. Solo se detectaron 4 errores críticos, y las correcciones laterales se redujeron a 500, lo cual demuestra un comportamiento mucho más estable. Además, el dron logró recorrer una distancia estimada de 1055.23 cm, reflejando un desplazamiento continuo y controlado durante el trayecto.

3.1.1.2 Trayectoria cuadrada en superficie horizontal

En trayectorias con curvas cerradas, como las de un cuadrado, una rotación moderada no resulta suficiente, ya que el dron tiende a desviarse con facilidad al no lograr ajustar adecuadamente su orientación en las esquinas. Para resolver esta limitación, es necesario implementar un ajuste más preci-

so en los ángulos de rotación, permitiendo que el dron reconozca y maniobre correctamente en cada cambio de dirección. La velocidad de avance utilizada en esta trayectoria se mantiene igual a la empleada en la trayectoria anterior. En la Figura 89 se puede visualizar la realización de esta prueba.

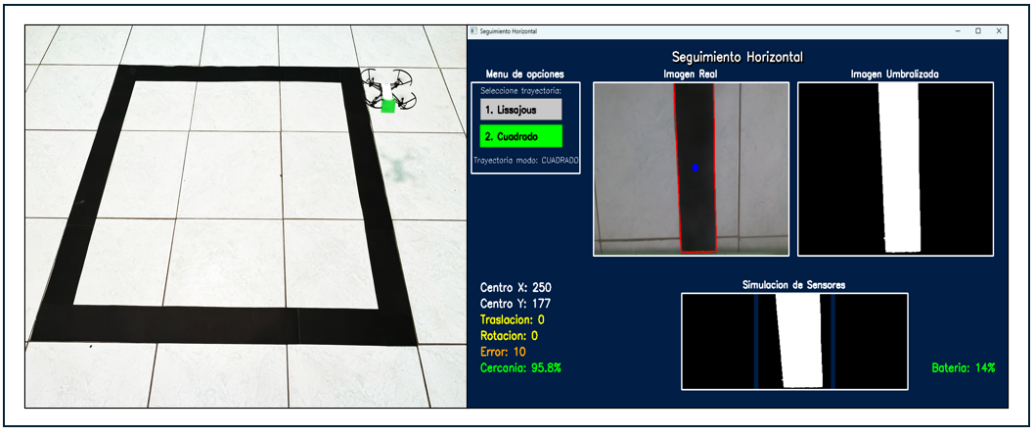


Figura 89: Seguimiento de trayectoria cuadrada en superficie horizontal.
Fuente: Autoría propia

Los resultados obtenidos a partir de las tres pruebas realizadas sobre la trayectoria en forma de cuadrado se presentan en la Tabla 7. Estos datos permiten analizar el rendimiento del dron al enfrentarse a trayectorias con curvas cerradas, evaluando la capacidad de respuesta en los giros.

Tabla 7: Resultados obtenidos en la trayectoria cuadrada

Prueba	**Tiempo (s)**	**Cercanía (%)**	**Error (px)**	**Err. > 100 px**	**Corr. lat.**	**Dist. (cm)**
1	71.42	75.35	59.17	616	4592	712.91
2	70.22	86.51	32.38	22	4392	700.89
3	74.33	89.79	24.51	5	4009	741.82

Fuente: Autoría propia

En la primera ejecución de la trayectoria cuadrada se obtuvo un tiempo total de 71.42 segundos, con un promedio de cercanía del 75.35 % y un promedio de error de 59.17 píxeles. Esta prueba presentó un número elevado de errores mayores a 100 píxeles (616 en total), lo que refleja una inestabilidad considerable en el seguimiento de la trayectoria durante varios tramos. Asi mismo, se registraron 4592 correcciones laterales, indicando que el dron tuvo que ajustar constantemente su posición. La distancia estimada recorrida fue de 712.91 cm.

Durante la segunda prueba se observó una mejora notable en el desempeño del dron. El tiempo total fue ligeramente inferior, con 70.22 segundos, y la cercanía promedio aumentó hasta un 86.51 %. Además, el error promedio disminuyó significativamente a 32.38 píxeles. La cantidad de errores superiores a 100 píxeles tambien se redujo a 22, y las correcciones laterales descendieron a 4392. La distancia estimada fue de 700.89 cm, manteniendose en un rango similar al de la primera prueba.

En la tercera prueba se alcanzaron los mejores resultados entre las tres ejecuciones. El tiempo total fue de 74.33 segundos, con un promedio de cercanía del 89.79 % y un error promedio de tan solo 24.51 píxeles. Unicamente se detectaron 5 errores superiores a 100 píxeles, lo que evidencia un control mucho más preciso del dron. Además, las correcciones laterales disminuyeron a 4009, y la distancia estimada aumentó a 741.82 cm. Estos resultados indican una trayectoria más estable y un rendimiento general optimizado en comparación con las pruebas anteriores.

3.1.2. Pruebas reales en el seguimiento vertical

Para el seguimiento vertical, se utilizó una pared amplia que permitió colocar las distintas trayectorias necesarias para evaluar el desempeño del dron. Durante estas pruebas, se recopilaron datos de forma manual, tales como: el tiempo total en segundos que el dron tarda en completar el recorrido medido con un cronómetro; la distancia total recorrida en centímetros determinada con un flexómetro, marcando los puntos donde el dron cambia de fase; y el número de fases ejecutadas durante el trayecto, con el fin de verificar si el dron logra completar correctamente una trayectoria cerrada. Esta última información se puede observar directamente en la consola de ejecución.

3.1.2.1 Trayectoria cuadrada en superficie vertical

En la trayectoria de forma cuadrada, el algoritmo de seguimiento sobre la pared emplea un sistema basado en máquinas de estados. Este enfoque permite que el seguimiento se realice de manera más fluida y eficiente, ya que la lógica de control de fundamenta en la detección de sensores ubicados en las partes superior e inferior del campo visual. Este sistema incluye movimientos verticales y horizontales que representan adecuadamente el recorrido cuadrado. En particular, el dron ejecuta una secuencia de ascenso, desplazamientos laterales y descenso, como se muestra en la Figura 90. Durante estas pruebas, es fundamental calibrar las velocidades vertical y lateral del dron, de modo que se adapten correctamente a la trayectoria establecida.

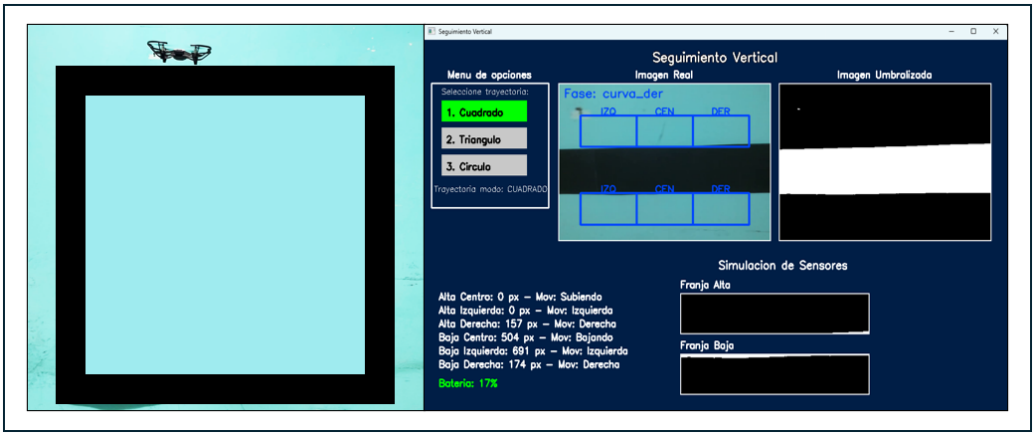


Figura 90: Seguimiento de trayectoria cuadrada en superficie vertical.
Fuente: Autoría propia

A partir de las pruebas realizadas en la trayectoria cuadrada, se recopilaron los datos correspondientes a las tres ejecuciones del dron, cuyos resultados se presentan en la Tabla 8. Estos datos fueron obtenidos mediante observación directa y registro manual durante cada recorrido, permitiendo evaluar el comportamiento del sistema de navegación. La información recopilada es fundamental para analizar la consistencia del algoritmo, así como la precisión del dron al completar una trayectoria cerrada de forma estable y controlada.

Tabla 8: Resultados del seguimiento vertical en trayectoria cuadrada

Prueba	**Tiempo total (s)**	**Cambios de fase**	**Distancia estimada (cm)**
1	22.81	4	480
2	23.98	4	510
3	21.17	4	475

Fuente: Autoría propia

En la primera prueba, el dron completó exitosamente la trayectoria cuadrada vertical en un tiempo total de 22.81 segundos, realizando los 4 cambios de fases esperados, lo que indica que recorrió los cuatro lados del cuadrado de forma lógica. La distancia estimada para esta prueba fue de aproximadamente 480 cm, determinada según el tamaño físico de la trayectoria en la pared.

Durante la segunda prueba, se mantuvo la lógica del seguimiento por fases, también registrando 4 transiciones de fase, con un tiempo total ligeramente mayor de 23.98 segundos, posiblemente por pequeñas variaciones en la

velocidad o la detección visual. La distancia estimada fue de 510 cm, un poco mayor, lo que podría deberse a una leve desviación o a un desplazamiento más amplio durante las curvas.

Finalmente, en la tercera prueba, el dron demostró un desempeño más eficiente en cuanto al tiempo, completando la trayectoria en 21.17 segundos, manteniendo nuevamente los 4 cambios de fase, y recorriendo una distancia estimada de 475 cm.

3.1.2.2 Trayectoria triangular en superficie vertical

En este tipo de trayectoria, es necesario coordinar los movimientos verticales y horizontales, ya que el dron debe desplazarse en dirección diagonal para seguir adecuadamente el contorno de la trayectoria triangular. Posteriormente, debe descender también de forma diagonal y continuar con un desplazamiento lateral para completar el seguimiento de la figura. Este comportamiento se ilustra en la Figura 91.

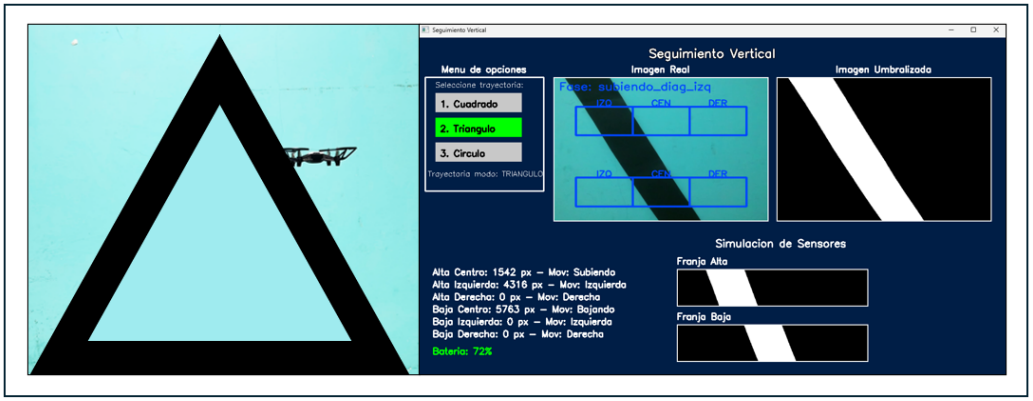


Figura 91: Seguimiento de trayectoria triangular en superficie vertical.
Fuente: Autoría propia

Los resultados obtenidos a partir de las tres pruebas realizadas sobre la trayectoria en forma triangular se presentan en la Tabla 9. Se analiza la comparación entre las tres ejecuciones con el fin de evaluar el comportamiento del dron al completar la trayectoria cerrada.

Tabla 9: Resultados del seguimiento vertical en trayectoria triangular

Prueba	**Tiempo total (s)**	**Cambios de fase**	**Distancia estimada (cm)**
1	10.49	3	350
2	9.50	3	330
3	15.10	3	390

Fuente: Autoría propia

En la primera prueba, el dron completó la trayectoria triangular en un tiempo total de 10.49 segundos, realizando correctamente los 3 cambios de fase esperados. La distancia recorrida fue de aproximadamente 350 cm, lo cual es consistente con una trayectoria bien definida y ejecutada dentro del espacio delimitado.

Durante la segunda prueba, se observó un rendimiento ligeramente más eficiente en cuanto al tiempo, ya que el dron completó el recorrido en 9.50 segundos, manteniendo también los 3 cambios de fase correspondientes. En este caso, la distancia estimada fue de 330 cm, lo cual sugiere una ejecución más precisa o un ligero ajuste en el tamaño de la trayectoria que el dron interpretó visualmente.

En la tercera prueba, el dron tardó 15.10 segundos en completar la figura triangular, lo que representa el mayor tiempo entre las tres pruebas. Aun así, realizó los 3 cambios de fase de manera correcta. La distancia estimada fue de 390 cm, lo que puede deberse a pequeñas variaciones en la posición inicial o a un trazado más amplio durante alguna de las fases. Este comportamiento puede explicarse por la naturaleza de la trayectoria triangular, cuyas formas angulares y cambios bruscos de dirección tienden a desestabilizar el seguimiento del dron, generando desviaciones que incrementan tanto el tiempo de ejecución como la distancia total recorrida.

3.1.2.3 Trayectoria circular en superficie vertical

En esta trayectoria, el algoritmo de seguimiento fue ajustado para integrar las fases empleadas en las trayectorias previas, con el objetivo de permitir que el dron siga un recorrido de forma circular. Para ello, se alternan dinámicamente diferentes fases de movimiento, tales como ascensos, descensos diagonales, así como curvas en las secciones superior e inferior de la trayectoria. El desarrollo del proceso de ejecución correspondiente a esta prueba se presenta en la Figura 92.

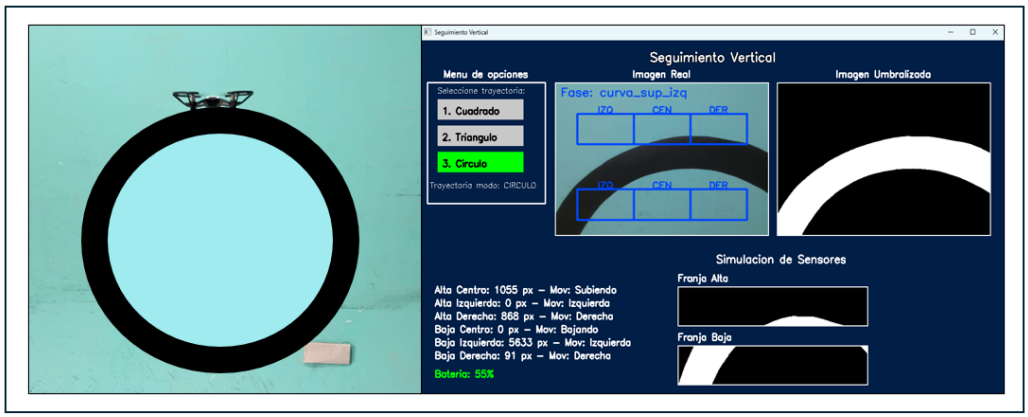


Figura 92: Seguimiento de trayectoria circular en superficie vertical.
Fuente: Autoría propia

En la Tabla 10 se presentan los datos correspondientes a las tres pruebas realizadas para la trayectoria circular. Estos resultados permiten evaluar el desempeño del dron en la ejecución de trayectorias con transiciones continuas y curvas suaves, características propias de este tipo de recorrido. Además, el análisis de los valores registrados facilita la identificación de posibles variaciones en el tiempo de ejecución, la distancia total recorrida y el número de fases completadas.

Tabla 10: Resultados del seguimiento vertical en trayectoria circular

Prueba	**Tiempo total (s)**	**Cambios de fase**	**Distancia estimada (cm)**
1	25.05	8	320
2	21.22	8	250
3	22.32	8	253

Fuente: Autoría propia

En la prueba inicial, el dron completó el recorrido circular en un tiempo total de 25.05 segundos, ejecutando correctamente los 8 cambios de fase que conforman el ciclo de esta trayectoria. La distancia estimada recorrida fue de 320 cm, lo cual siguiere un desplazamiento amplio y constante en cada uno de los segmentos de la figura circular, manteniendo un flujo continuo de movimiento sin interrupciones significativas.

En la segunda ejecución, se observó una ligera mejora en el tiempo de desempeño, reduciéndose a 21.22 segundos, manteniéndose también los 8 cambios de fase, lo que indica que el dron logró completar la trayectoria completa sin errores de transición. La distancia estimada en esta prueba fue

de 250 cm, lo que podría deberse a ligeras variaciones en el ajuste o detección de los puntos de cambio de fase, o a un trazo más compacto del círculo por parte del dron.

Finalmente, en la tercera prueba realizada, el dron registró un tiempo de 22.32 segundos, manteniéndose dentro del rango observado en las pruebas anteriores. Nuevamente se completaron los 8 cambios de fase, asegurando así que se realizó un ciclo completo de la trayectoria circular. La distancia estimada recorrida fue de 253 cm, mostrando una consistencia respecto a la segunda prueba, lo que valida el comportamiento estable del dron en esta trayectoria.

3.2. Discusión

Al comparar los resultados obtenidos en los sistemas de seguimiento horizontal (trayectorias sobre el suelo) frente a los de seguimiento vertical (trayectorias sobre la pared), se evidencian diferencias fundamentales tanto en el enfoque de control como en los indicadores de rendimiento utilizados.

El seguimiento horizontal, basado en el cálculo constante de métricas como porcentaje de cercanía, error promedio en píxeles, y correcciones laterales, permite una evaluación precisa del desempeño del dron a lo largo de trayectorias curvas y poligonales. Trayectorias como la curva de Lissajous demostraron alta estabilidad y precisión, con un error progresivamente menor y una cercanía superior al 90 %, especialmente en las últimas pruebas. Aunque la trayectoria cuadrada presentó mayores desafíos debido a los cambios bruscos de dirección, también se logró una mejora notable con ajustes en el sistema.

Por otro lado, el seguimiento vertical, implementado en trayectorias en forma de cuadrado, triángulo y círculo en pared, opera bajo una lógica distinta, basada en un sistema por fases que guía al dron según la detección de segmentos clave (líneas verticales, diagonales o curvas) sin necesidad de calcular errores en píxeles. Los indicadores utilizados en este caso fueron más estructurales: tiempo total de ejecución, numero de cambios de fase y distancia estimada recorrida. Esta estrategia es menos sensible a pequeñas variaciones visuales, lo que facilita la estabilidad en ambientes con menor contraste o iluminación variable. Sin embargo, requiere una mayor precisión en el diseño físico de la trayectoria en la pared, ya que la detección por fases depende directamente de la geometría.

Al comparar el desempeño del dron en las distintas trayectorias, se eviden-

cia que la figura triangular, aunque con tiempos bajos en dos de las pruebas, presentó mayores desafíos para el sistema de navegación. La geometría de sus ángulos generó confusión en el dron, provocando desfases o cambios de fase anticipados en ciertos casos. Esto afectó tanto la precisión como la estabilidad del seguimiento. Mientras que, la trayectoria cuadrada, con cuatro cambios de fase y un tiempo promedio de ejecución cercano a los 23 segundos, mostró un comportamiento más regular y predecible. Por su parte, la figura circular, que involucró ocho fases y los tiempos más altos, reflejó una ejecución fluida, aunque exigente por la continuidad del contorno. En conjunto, los resultados indican que, más allá del tiempo total, la trayectoria triangular fue la que implicó mayor complejidad operativa.

En resumen, mientras que el seguimiento horizontal ofrece datos cuantitativos finos y detallados, ideal para calibraciones y análisis visuales, el seguimiento vertical destaca por su enfoque lógico, siendo útil en escenarios donde las condiciones visuales puedan dificultar un control basado en píxeles. Ambos enfoques se complementan al demostrar distintas capacidades del dron en entornos y estructuras variadas.

Para complementar el análisis del rendimiento del sistema de seguimiento horizontal, se calcularon los valores de media y desviación estándar para cada una de las métricas registradas en las pruebas experimentales. La media permite representar el comportamiento promedio del dron en aspectos como el tiempo total de ejecución, la precisión del seguimiento (medida mediante la cercanía y error en píxeles), la cantidad de errores críticos, el número de correcciones laterales y la distancia estimada recorrida. Por su parte, la desviación estándar cuantifica la variabilidad o dispersión de los datos respecto al valor medio, lo cual resulta útil para evaluar la consistencia del sistema entre múltiples ejecuciones de la misma trayectoria. Esta información permite comparar de forma objetiva el rendimiento entre diferentes trayectorias y analizar la estabilidad del comportamiento del dron durante su desplazamiento. Los resultados obtenidos se resumen en la Tabla 11.

Tabla 11: Métricas descriptivas del comportamiento del dron en trayectorias horizontales

Métrica	**Lissajous**		**Cuadrado**	
	Media	**Desv. estándar**	**Media**	**Desv. estándar**
Tiempo total (s)	92.57	12.65	71.99	2.11
Prom. cercanía (%)	91.67	3.50	83.88	7.56
Prom. error (px)	19.98	8.41	38.68	18.17
Errores (>100 px)	23.33	30.92	214.33	347.95
Corr. laterales	3429.00	2546.06	4331.00	296.24
Distancia estimada (cm)	914.12	126.63	718.54	21.03

Fuente: Autoría propia

Como se observa en la Tabla 11, la trayectoria de Lissajous presentó un mayor tiempo promedio de ejecución y una mayor distancia recorrida, lo cual es coherente con su complejidad geométrica en comparación con la trayectoria cuadrada. Además, se evidenció un mejor desempeño en términos de precisión, reflejado en un menor promedio de error (19.98 px) y una mayor cercanía al objetivo (91.67 %), en comparación con los valores obtenidos para la trayectoria cuadrada. No obstante, también se registró una mayor variabilidad en métricas como los errores críticos y las correcciones laterales, lo que indica que, aunque el sistema fue más preciso en promedio, requirió un control más activo para mantenerse en la trayectoria. Estos resultados permiten evaluar no solo el rendimiento general del sistema de seguimiento horizontal, sino también su consistencia en diferentes formas geométricas.

En el seguimiento vertical, se realizó un análisis estadístico de los resultados experimentales correspondientes a las trayectorias cuadrada, triangular y circular. Al igual que en el caso del seguimiento horizontal, se calcularon los valores de media y desviación estándar para cada una de las métricas registradas. En este sistema se consideraron parámetros como el tiempo total de ejecución, la cantidad de cambios de fase y la distancia estimada recorrida. Los resultados obtenidos se presentan en la Tabla 12.

Tabla 12: Métricas descriptivas del comportamiento del dron en trayectorias verticales

Métrica	**Cuadrado**		**Triángulo**		**Círculo**	
	Media	**Desv. est.**	**Media**	**Desv. est.**	**Media**	**Desv. est.**
Tiempo (s)	22.65	1.41	11.69	2.98	22.86	1.97
Fases	4.00	0.00	3.00	0.00	8.00	0.00
Distancia (cm)	488.33	18.92	356.66	30.55	274.33	39.57

Fuente: Autoría propia

En cuanto al análisis comparativo de las métricas registradas en cada trayectoria, se observa que la figura triangular presentó el menor tiempo promedio de ejecución (11.69 segundos), aunque con una desviación estándar notablemente mayor (2.98 segundos) en comparación con las demás trayectorias. Esta variabilidad temporal sugiere inestabilidad en el seguimiento, probablemente originada por desfasamientos o anticipaciones erróneas en los cambios de fase, lo cual también se refleja en su distancia estimada promedio de 356.66 cm, con una desviación estándar de 30.55 cm.

Por otro lado, la trayectoria cuadrada evidenció un tiempo promedio de 22.65 segundos con menor variabilidad (1.41 segundos), cumpliendo de forma consistente los cuatro cambios de fase esperados. Su distancia promedio fue de 488.33 cm, lo que refleja un desplazamiento más amplio, acorde con la estructura de la figura.

La trayectoria circular, pese a implicar ocho fases, presentó un tiempo medio de 22.86 segundos y una desviación estándar de 1.97 segundos. Su distancia recorrida fue la menor (274.33 cm), con la desviación más alta (39.57 cm), lo que podría atribuirse a variaciones en la ejecución de curvas suaves y posibles ajustes frecuentes durante el desplazamiento.

Estos resultados permiten evidenciar que, aunque el triángulo fue la figura más rápida en promedio, también fue la menos estable en su ejecución, lo cual sugiere que la lógica de movimiento podría beneficiarse de ajustes específicos para mejorar el rendimiento en trayectorias con ángulos agudos o cambios bruscos de dirección.

4. Conclusiones y recomendaciones

4.1. Conclusiones

- La comprensión de los fundamentos teóricos de visión por computador y el procesamiento de imágenes fue esencial para entender el funcionamiento de los sistemas involucrados. El análisis y tratamiento de imágenes representó un paso clave en el diseño y desarrollo del sistema de navegación del dron.

- El uso del modelo de color HSV resultó conveniente para la detección de imágenes, ya que permitió una segmentación mas precisa de los colores, independientemente de las variaciones de iluminación. Esto facilitó la identificación de la trayectoria y mejoró la fiabilidad del sistema.

- El sistema de navegación desarrollado demostró ser funcional y adaptable ante distintas trayectorias, evidenciando un buen rendimiento general del dron. En el caso de la trayectoria de Lissajous, el dron mostró un desempeño más favorable debido a la suavidad de sus curvas, lo que le permitió mantener un desplazamiento continuo y fluido. Por el contrario, en la trayectoria en forma de cuadrado, el dron presentó dificultades para alinearse rápidamente, debido a la brusquedad de las curvas cerradas. En el seguimiento vertical, el uso de un enfoque basado en máquinas de estado, resultó ser una estrategia efectiva para gestionar los movimientos del dron, permitiendo una transición organizada entre fases y una respuesta más estable ante las trayectorias propuestas.

- En el diseño de las trayectorias se optó por utilizar el color negro, considerando que el área de pruebas contaba con un suelo de color blanco. Esta elección facilitó el proceso de umbralización mediante ajustes en el modelo de color, permitiendo obtener una imagen binaria en la que la trayectoria se visualiza en blanco y el entorno de color negro, lo cual ayudó a reducir el ruido visual y mejorar la detección.

- El uso de métricas cuantitativas como el tiempo de ejecución, el promedio de cercanía, el error en píxeles, el número de errores críticos, las correcciones laterales, los cambios de fase y la distancia estimada resultó fundamental para evaluar de forma objetiva el desempeño del dron en las distintas trayectorias. Estas métricas permitieron identificar fortalezas y debilidades del sistema en condiciones reales, facilitando

el análisis técnico del comportamiento del dron tanto en seguimiento horizontal como vertical.

- La aplicación de herramientas estadísticas como la media y la desviación estándar permitió completar el análisis de las métricas obtenidas, proporcionando una visión mas profunda sobre la consistencia y estabilidad del sistema de navegación implementado. A través de estos valores fue posible comparar la variabilidad entre diferentes ejecuciones de una misma trayectoria y determinar el grado de fiabilidad del comportamiento del dron.

- Los resultados obtenidos muestran que el rendimiento del sistema mejoró progresivamente en trayectorias suaves como la de Lissajous, en el caso del seguimiento horizontal. En esta trayectoria, el dron alcanzó un promedio de error de 19.98 px, una cercanía del 91.67 %, y apenas 23.33 errores críticos. En comparación con la trayectoria cuadrada, donde el sistema presentó mayores dificultades, con un promedio de error de 38.68 px, una cercanía del 83.88 %, y un incremento considerable en los errores críticos de 214.33. Esta diferencia en el desempeño evidencia la influencia de la geometría de la trayectoria sobre la estabilidad y confirma que el sistema puede optimizar su funcionamiento cuando se ajustan adecuadamente los parámetros de control y procesamiento de imagen.

- En el seguimiento vertical, los datos reflejan un comportamiento mucho más consistente, especialmente en la trayectoria cuadrada, que mostró una mayor estabilidad con un tiempo promedio de 22.65 segundos y baja desviación estándar de 1.41 segundos. La trayectoria triangular, por otro lado, presentó mayores dificultades debido a sus vértices agudos, lo que generó una mayor variabilidad en el tiempo, con una media de 11.69 segundos y una desviación estándar de 2.98 segundos. La trayectoria circular mantuvo un rendimiento controlado, con resultados similares a los del cuadrado, con un tiempo promedio de 22.86 segundos y una desviación estándar de 1.97 segundos. Estos datos confirman que el enfoque basado en máquinas de estados resulta eficaz para trayectorias con geometrías bien definidas y que la forma de la trayectoria influye directamente en la precisión del seguimiento.

- A pesar de ciertas limitaciones físicas evidenciadas durante las pruebas, como el sobrecalentamiento del dron en ejecuciones continuas, la limitada duración de la batería, la necesidad de buena iluminación y la afectación a la estabilidad por el peso adicional del soporte en 3D, el

sistema demostró un desempeño confiable. Los algoritmos implementados respondieron adecuadamente frente a estas condiciones, lo que respalda su solidez dentro del entorno experimental propuesto.

4.2. Recomendaciones

- Es importante realizar un ajuste adecuado de los parámetros HSV del algoritmo de detección de colores, con el fin de calibrar correctamente la imagen y generar una máscara adecuada. Una calibración deficiente podría provocar la detección de regiones no deseadas, afectando la fiabilidad del sistema de navegación.

- Al desarrollar las funciones que controlan los movimientos del dron, es fundamental ajustar adecuadamente las variables de acuerdo con la trayectoria a ejecutar. Es necesario modificar parámetros como la velocidad de rotación, avance, desplazamiento lateral y vertical, así como la sensibilidad del sistema. En el seguimiento horizontal, por ejemplo, las curvas varían significativamente entre trayectorias como la de Lissajous y la cuadrada, lo que requiere configuraciones específicas. En el seguimiento vertical, los movimientos pueden ser rectos, diagonales u horizontales, por lo que también se deben adaptar los valores según cada caso.

- Para lograr una mejor comprensión del funcionamiento del dron DJI Tello, se recomienda revisar detalladamente el manual del fabricante. Este documento proporciona una guía completa sobre el ensamblaje y la configuración del dron, incluyendo la conexión correcta de sus componentes, la colocación adecuada de las hélices y el proceso de vinculación por Wi-Fi con un ordenador o dispositivo móvil.

- Se recomienda realizar las calibraciones necesarias del dron antes de iniciar el desarrollo del sistema. Entre ellas, la calibración del IMU (Unidad de medición inercial) y del centro de gravedad, la cual debe efectuarse durante un vuelo estacionario. Una vez completadas estas calibraciones, es conveniente ejecutar pruebas básicas de movimiento para verificar que el hardware funcione correctamente.

- En el caso del seguimiento horizontal, es necesario que las trayectorias tengan un ancho aproximado de 11cm, ya que este valor permite al dron detectar la línea de forma clara y calcular su centro con mayor precisión. Un grosor menor podría dificultar la detección, mientras que un grosor excesivo también podría generar imprecisiones. Mientras que,

para el seguimiento vertical no se requiere ajustar el ancho de la línea, ya que el sistema se basa en la cantidad de píxeles detectados en franjas específicas de la imagen, lo que permite una navegación confiable incluso con trayectorias más delgadas.

- Al momento de ejecutar las pruebas reales con el dron, se recomienda utilizar un espacio amplio y libre de obstáculos que puedan generar colisiones durante el vuelo. Además, es aconsejable seleccionar un entorno con iluminación uniforme y sin regiones oscuras, ya que estas podrían ser interpretadas erróneamente como parte de la trayectoria, afectando la precisión del seguimiento.

- Se recomienda optimizar el diseño del soporte 3D utilizado para mejorar la visualización del suelo. Dado que el dron Tello ya tiene estructura predefinida, disminuir el peso del soporte contribuirá a mejorar la estabilidad del vuelo y a reducir el consumo energético, especialmente durante el seguimiento horizontal, donde se evidenciaron ligeras inestabilidades atribuidas al peso adicional.

- Para fortalecer la evaluación del sistema, se sugiere incluir métricas adicionales como consumo energético, tiempo de respuesta ante cambios de trayectoria o análisis de estabilidad del vuelo. Además, realizar pruebas de mayor duración permitirá analizar el comportamiento del sistema bajo condiciones prolongadas, así como los efectos térmicos o de batería en el hardware del dron.

- Resulta conveniente llevar a cabo una investigación mas profunda sobre los algoritmos de navegación empleados en el dron, con el fin de perfeccionar su lógica de funcionamiento. Esto permitirá mejorar la precisión del seguimiento, especialmente en trayectorias donde se detectaron las desviaciones del centro. Así mismo, el desarrollo de algoritmos más avanzados podría habilitar al sistema para seguir trayectorias más complejas.

Bibliografía

Referencias

[1] "Historia y origen de los DronesDronesEc.club," Aug. 2019.

[2] "Historia de los drones," May 2016. Publication Title: El Drone.

[3] E. Navarro, "Usos y aplicaciones de drones en empresas," June 2017. Publication Title: TAKTIC.

[4] "Procesamiento de imágenes con OpenCV en Python."

[5] "Desarrollo de módulos de visión por computador en Python para la detección de objetos en un entorno de pruebas de conducción autónoma,"

[6] A. Álvarez Tenedor, "Visión por computador aplicada a la navegación y la detección de obstáculos de un dron experimental para la extinción de incendios," Oct. 2019.

[7] E. L. T. Loaiza, "Diseño e implementación de un algoritmo de seguimiento de trayectorias para el minidron Parrot mambo utilizando – Simulink/Stateflow.,"

[8] F. J. González, *POSICIONAMIENTO DE UN UAV MEDIANTE VISIÓN ARTIFICIAL INFRARROJA*. Sept. 2019.

[9] A. Al-Kaff, D. Martín, F. García, A. d. l. Escalera, and J. María Armingol, "Survey of computer vision algorithms and applications for unmanned aerial vehicles," *Expert Systems with Applications*, vol. 92, pp. 447–463, Feb. 2018.

[10] A. Viciana Gálvez, "IMPLEMENTACIÓN DE UN SISTEMA DE NAVEGACIÓN AUTÓNOMO BASADO EN VISIÓN POR COMPUTADOR APLICADO A DRON LO," Sept. 2022.

[11] E. I. R. Juárez, "Seguimiento de ruta para un drone.," July 2023.

[12] A. F. Silva Bohórquez, C. A. Peña Cortés, and L. E. Mendoza, "Sistema de inspección y vigilancia utilizando un robot aéreo guiado mediante visión artificial," *Iteckne*, vol. 10, pp. 190–198, Dec. 2013.

[13] javi, "RPA, RPAS, UAS y UAV: ¿Qué son y en qué se diferencian?," Oct. 2022. Publication Title: UMILES.

[14] “Tello.”

[15] A. Sánchez, “¿Cuáles son las partes de un Dron? Listado completo,” Mar. 2023. Publication Title: UMILES.

[16] “Amazon.com: Midzooparts TBS Source ONE V3 - Kits de marco FPV de fibra de carbono de 8.898 in (8.898 in) para FPV Free-estyle Acrobatic Flying : Juguetes y Juegos.”

[17] Andres, “Diferencia entre motores de AC (Corriente alterna) y DC (Corriente directa),” May 2018.

[18] cmendoza, “Cómo funciona un dron con explicacion de hélices y motores,” Jan. 2020. Publication Title: Guía Drones.

[19] “Controlador de motor electrónico ESC 30A - Teknomovo 2024.” Publication Title: Teknomovo.

[20] T. Robotics, “What is an Electronic Speed Controller & How Does an ESC Work.” Publication Title: Tyto Robotics.

[21] “Robótica, drones y electrónica.”

[22] “Lo que hay que saber para elegir el controlador de vuelo (FC) en un multicóptero – Prometec.”

[23] Natalia, “Tipos de baterías de Drones,” June 2023. Publication Title: IDC.

[24] migueldronero, “ Receptor de un dron ¿Que es? ,” June 2020. Publication Title: De Cero a Dronero.

[25] “Lo que hay que saber para elegir la transmisión de vídeo para un dron FPV – Prometec.”

[26] “Sensores de drones para el vuelo estacionario y las maniobras con precisión en el aire.” Publication Title: TE Connectivity.

[27] Marketing, “Visión por Computador Qué es, Aplicaciones y Objetivos,” Jan. 2022. Publication Title: EDS Robotics.

[28] admin, “¿Qué es la visión por computadora? Sistemas de visión artificial y aplicaciones: ejemplos de soluciones,” May 2021. Publication Title: RecFaces.

[29] M. Domingo, *Vision por Computador*. Santiago de Chile: Departamento de Ciencia de la Computación, Aug. 2004.

[30] J. Dong, "Discussion on Curriculum Reform of Digital Image Processing under the Certification of Engineering Education," *Open Journal of Social Sciences*, vol. 10, no. 01, pp. 147–154, 2022.

[31] J. Weng, P. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 965–980, Oct. 1992.

[32] E. Cuevas, D. Zaldivar, and M. P. Cisneros, "Procesamiento digital de imágenes con MatLAB y SIMULINK,"

[33] "Imagen - Qué es, funciones, características y ejemplos." Publication Title: https://concepto.de/.

[34] R. M. T. Ballestas, "LESSING ARTURO CÁRDENAS DÍAZ,"

[35] "Naturaleza de la luz," Jan. 2011.

[36] M. L. Sánchez, "Fundamentos de procesamiento de imágenes digitales,"

[37] G. Pajares, A. d. l. E. Hueso, and E. A. Gutiérrez, *Conceptos y métodos en visión por computador*. Grupo de Visión del Comité Español de Automática (CEA), 2016.

[38] J. C. M. Ospina, F. A. P. Ortiz, and J. W. B. Bedoya, "Corrección de iluminación para imágenes aéreas de cultivos tomadas a baja altitud.," *Revista Facultad Nacional de Agronomía Medellín*, vol. 60, pp. 4077–4104, July 2007.

[39] A. Vanacloig and V. Luis, "Digitalización de imágenes con ayuda del histograma," Nov. 2011.

[40] C. A. Cattaneo, L. I. Larcher, A. I. Ruggeri, A. C. Herrera, and M. Biasoni, "MÉTODOS DE UMBRALIZACIÓN DE IMÁGENES DIGITALES BASADOS EN ENTROPIA DE SHANNON Y OTROS," 2011.

[41] M. Roncero-Bazarra, "Guías BibUpo: Conceptos clave sobre edición multimedia: el vídeo digital: ¿Qué es un frame?."

[42] *Color*. Universidad de Caldas, 2005.

[43] P. Kakumanu, S. Makrogiannis, and N. Bourbakis, "A survey of skin-color modeling and detection methods," *Pattern Recognition*, vol. 40, pp. 1106–1122, Mar. 2007.

[44] M. A. A. Pérez, "Espacios de Color RGB, HSI y sus Generalizaciones a n-Dimensiones,"

[45] I. M. Basilio and J. Alberto, *DETECCIÓN DE IMÁGENES CON CONTENIDO EXPLÍCITO USANDO LOS MODELOS DE COLOR HSV y YCbCr*. Thesis, Nov. 2011.

[46] R. C. Gonzalez and R. E. Woods, *Digital image processing*. Upper Saddle River, NJ: Prentice-Hall, 2. ed., internat. ed ed., 2002.

[47] K. Jack, *Video demystified: a handbook for the digital engineer*. Demystifying technology series, Amsterdam ; Boston: Elsevier, 4th ed ed., 2005.

[48] E. d. C. d. GoDaddy, "Lenguajes de programación más populares: Una visión general," Nov. 2023. Publication Title: GoDaddy Resources - LATAM.

[49] "MATLAB - El lenguaje del cálculo técnico."

[50] "Matlab \textbar Cuahsi.org."

[51] M. M. Canelo, "Qué es Scratch y para qué sirve," Dec. 2023. Publication Title: Profile Software Services.

[52] ltognoli, "Aprende a crear tu primer videojuego con Scratch - campus-norte.unc.edu.ar," Sept. 2024.

[53] "Python: qué es, para qué sirve y cómo se programa \textbar Informática Industrial," Oct. 2020.

[54] "Introduction To Python \textbar DCNC."

[55] "10 Python Libraries for Computer Vision," June 2019.

[56] A. McFarland, "Las 10 mejores bibliotecas de procesamiento de imágenes en Python," May 2022. Publication Title: Unite.AI.

[57] "SISTEMAS DE COMUNICACIONES."

[58] R. G. Méndez and C. H. Arias, "Metodología Para el Análisis de Redes en Instituciones Públicas Basada en Practicas ITIL v3," 2020.

[59] B. A. P. Anaguano, "INTEGRANTES: LIZARDO ALEXIS LUCERO MOLINA,"

[60] J. C. P. Dominguez, "" TECNOLOGÍA BLUETOOTH " Tesis para obtener el grado de Ingeniero en Comunicaciones y Electrónica. INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA ELÉCTRICA DEPARTAMENTO DE TITULACIÓN,"

[61] "¿Qué es el wifi? - Tipos de conexiones wifi y seguridad \textbar Proofpoint ES," June 2023. Publication Title: Proofpoint.

[62] "SECRETARIA DE COMUNICACIONES Y TRANSPORTES," *14 de noviembre de 2019*, Nov. 2019.

[63] Rexct-29, "ACTUALIZACIÓN \textbar Nuevo Reglamento Europeo sobre RPAS \textbar ¡Consulta las nuevas fechas!," Jan. 2021. Publication Title: One Air.

[64] "Drone Laws in Argentina [Updated January 15, 2025]," Jan. 2025. Publication Title: https://drone-laws.com/.

[65] "Operación de Aeronaves Pilotadas a Distancia (RPAs)," Nov. 2020.

[66] J. L. Chanco Alvear, *Control de un robot seguidor de línea tipo diferencial mediante visión artificial.* bachelorThesis, Aug. 2018.

[67] "Manual de usuario DJI Tello (22 páginas)."

[68] "Drone Dji Tello Boost Combo Dji020 Mini Cámara HD con 3 baterías."

[69] admin, "DJI Tello," June 2020. Publication Title: MUNDO DRONE.

[70] "DJI RYZE Tello Drones Motor engine Repair parts 4Pcs 1set."

[71] A. Casero, "¿Qué es PyCharm y cómo utilizarlo? [2025] | KeepCoding." Section: Blog.

[72] "PyCharm : Todo sobre el IDE de Python más popular," Nov. 2022. Publication Title: Formación en ciencia de datos \textbar DataScientest.com.

[73] "djitellopy: Tello drone library including support for video streaming, swarms, state packets and more."

[74] A. S. Alberca, "La librería Numpy."

[75] F. Zyprian, “CV2: Guía de OpenCV para desarrolladores de Python,” Mar. 2023.

[76] “OpenCV: biblioteca Python para procesamiento de imágenes,” Feb. 2025.

Anexos

Anexo A: Código en PyCharm para detección de colores

```
from djitellopy import tello
import numpy as np
import cv2

# Dimensiones
Ancho, Alto = 480, 360

# Inicializar dron
DRON = tello.Tello()
DRON.connect()
DRON.streamon()
print("Bater a:", DRON.get_battery())

# Funcion vacia para trackbars
def empty(a): pass

# Crear ventana para controles HSV
cv2.namedWindow("Controles HSV")
cv2.resizeWindow("Controles HSV", 640, 240)
cv2.createTrackbar("Ton min", "Controles HSV", 0, 179, empty)
cv2.createTrackbar("Ton max", "Controles HSV", 179, 179, empty)
cv2.createTrackbar("Sat min", "Controles HSV", 0, 255, empty)
cv2.createTrackbar("Sat max", "Controles HSV", 255, 255, empty)
cv2.createTrackbar("Brillo min", "Controles HSV", 0, 255, empty)
cv2.createTrackbar("Brillo max", "Controles HSV", 255, 255, empty)

while True:
    # Leer imagen desde el dron
    imagen = DRON.get_frame_read().frame
    imagen = cv2.resize(imagen, (Ancho, Alto))
    imagen = cv2.cvtColor(imagen, cv2.COLOR_RGB2BGR)

    # Convertir a HSV
    imagen_hsv = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)

    # Leer valores de HSV desde trackbars
    t_min = cv2.getTrackbarPos("Ton min", "Controles HSV")
    t_max = cv2.getTrackbarPos("Ton max", "Controles HSV")
    s_min = cv2.getTrackbarPos("Sat min", "Controles HSV")
    s_max = cv2.getTrackbarPos("Sat max", "Controles HSV")
    b_min = cv2.getTrackbarPos("Brillo min", "Controles HSV")
    b_max = cv2.getTrackbarPos("Brillo max", "Controles HSV")

    print(f'[{t_min},{s_min},{b_min},{t_max},{s_max},{b_max}]')

    menor = np.array([t_min, s_min, b_min])
    mayor = np.array([t_max, s_max, b_max])

    # Aplicar mascara
    mascara = cv2.inRange(imagen_hsv, menor, mayor)
    resultado = cv2.bitwise_and(imagen, imagen, mask=mascara)

    # Convertir mascara a BGR para mostrarla a color
    mascara_color = cv2.cvtColor(mascara, cv2.COLOR_GRAY2BGR)

    # Agregar etiquetas a cada imagen
    def etiquetar(img, texto):
        img = img.copy()
        cv2.rectangle(img, (0, 0), (img.shape[1], 30), (30, 30, 30), -1)  #
            Fondo para texto
        cv2.putText(img, texto, (10, 22), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
            (255, 255, 255), 2)
        return img

    imagen_etiquetada = etiquetar(imagen, "Imagen Original")
    mascara_etiquetada = etiquetar(mascara_color, "Mascara HSV")
    resultado_etiquetado = etiquetar(resultado, "Resultado Final")

    # Unir horizontalmente
```

```
        combinada = np.hstack((imagen_etiquetada, mascara_etiquetada,
            resultado_etiquetado))

        # Crear fondo oscuro
        fondo = np.zeros((combinada.shape[0] + 20, combinada.shape[1], 3), dtype
            =np.uint8)
        fondo[:] = (20, 20, 20)  # Gris oscuro
        fondo[10:10+combinada.shape[0], :] = combinada

        # Mostrar
        cv2.imshow("Visualizacion de las tres imagenes", fondo)

        # Salir con 'q'
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cv2.destroyAllWindows()
```

Anexo B: Código en PyCharm para seguimiento horizontal

```
import cv2
import numpy as np
from djitellopy import tello
import time

# ========== INICIALIZACION ==========
DRON = tello.Tello()
DRON.connect()
DRON.streamon()
DRON.takeoff()
time.sleep(2)
DRON.move_down(30)
print("Bater a:", DRON.get_battery())

# ========== PARAMETROS ==========
ancho, alto = 480, 360
valores_hsv = [0,0,0,179,255,75]
sensores = 3
umbral = 0.2
ultima_direccion = 'derecha'

trayectoria = 0  # Modo inicial por defecto
modo = ""  # Para mostrarlo en la interfaz

# ========== FUNCIONES COMUNES ==========
def umbralizado(imagen):
    hsv = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)
    menor = np.array([valores_hsv[0], valores_hsv[1], valores_hsv[2]])
    mayor = np.array([valores_hsv[3], valores_hsv[4], valores_hsv[5]])
    mascara = cv2.inRange(hsv, menor, mayor)
    return mascara

def obtenerSalidaSensores(imagen_umbraliz, sensores):
    global umbral
    imagenes = np.hsplit(imagen_umbraliz, sensores)
    pixelsTotales = (imagen_umbraliz.shape[1] // sensores) * imagen_umbraliz
        .shape[0]
    envio = []
    sensores_imgs = []              # Se guarda las imagenes de los sensores

    for i, img in enumerate(imagenes):
        contpixels = cv2.countNonZero(img)
        if contpixels > umbral * pixelsTotales:
            envio.append(1)
        else:
            envio.append(0)
        sensores_imgs.append(cv2.cvtColor(img, cv2.COLOR_GRAY2BGR))
        #cv2.imshow(f"Sensor {i}", img)
    print("Lectura sensores:", envio)
    return envio, sensores_imgs

def detectarCentro(imagen_umbraliz, imagen):
    centro_x = 0
    centro_y = 0
    contornos, _ = cv2.findContours(imagen_umbraliz, cv2.RETR_EXTERNAL, cv2.
        CHAIN_APPROX_SIMPLE)
    if contornos:
        cv2.drawContours(imagen, contornos, -1, (0, 0, 255), 2)
        mayor = max(contornos, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(mayor)
        centro_x = x + w // 2
        centro_y = y + h // 2
        cv2.circle(imagen, (centro_x, centro_y), 8, (255, 0, 0), -3)
    return centro_x, centro_y

# ========== FUNCIONES DE COMANDO ==========
def envioComandos_lissajous(envio, centro_x):
    global ancho, pesoValor, velAvance, sensibilidad
    error = centro_x - (ancho // 2)
    izq_derech = int(np.clip(error // sensibilidad, -15, 15))
    izq_derech = 0 if -2 < izq_derech < 2 else izq_derech
```

```
    rotacion = {
        (0, 0, 1): pesoValor[4],
        (0, 1, 0): pesoValor[2],
        (0, 1, 1): pesoValor[3],
        (1, 0, 0): pesoValor[0],
        (1, 1, 0): pesoValor[1],
    }.get(tuple(envio), 0)

    DRON.send_rc_control(izq_derech, velAvance, 0, rotacion)
    return izq_derech, rotacion

def envioComandos_cuadrado(envio, centro_x):
    global ancho, pesoValor, velAvance, sensibilidad, ultima_direccion
    error = centro_x - (ancho // 2)
    izq_derech = int(np.clip(error // sensibilidad, -15, 15))
    izq_derech = 0 if -2 < izq_derech < 2 else izq_derech

    if envio == [1, 0, 0] or envio == [1, 1, 0]:
        rotacion = pesoValor[0] if envio == [1, 0, 0] else pesoValor[1]
        ultima_direccion = 'izquierda'
    elif envio == [0, 1, 1] or envio == [0, 0, 1]:
        rotacion = pesoValor[3] if envio == [0, 1, 1] else pesoValor[4]
        ultima_direccion = 'derecha'
    elif envio == [0, 1, 0]:
        rotacion = pesoValor[2]
    elif envio == [1, 1, 1]:
        rotacion = -20 if ultima_direccion == 'derecha' else 20
        izq_derech = 0
    else:
        rotacion = 0

    DRON.send_rc_control(izq_derech, velAvance, 0, rotacion)
    return izq_derech, rotacion

def envioComandos_triangulo(envio, centro_x):
    global ancho, pesoValor, velAvance, sensibilidad, ultima_direccion
    error = centro_x - (ancho // 2)
    izq_derech = int(np.clip(error // sensibilidad, -15, 15))
    izq_derech = 0 if -2 < izq_derech < 2 else izq_derech

    if envio == [0, 0, 1] or envio == [0, 1, 1]:
        rotacion = pesoValor[4] if envio == [0, 0, 1] else pesoValor[3]
        ultima_direccion = 'derecha'
    elif envio == [1, 0, 0] or envio == [1, 1, 0]:
        rotacion = pesoValor[0] if envio == [1, 0, 0] else pesoValor[1]
        ultima_direccion = 'izquierda'
    elif envio == [0, 1, 0]:
        rotacion = pesoValor[2]
    elif envio == [1, 1, 1]:
        rotacion = 10 if ultima_direccion == 'derecha' else -10
        izq_derech = 0
    else:
        rotacion = 0
        izq_derech = 0

    DRON.send_rc_control(izq_derech, velAvance, 0, rotacion)
    return izq_derech, rotacion

# ========== LOOP PRINCIPAL ==========
print("     Presiona 1 (Infinito), 2 (Cuadrado), .Q para salir.")

# ========== BUCLE PRINCIPAL ==========
#trayectoria = 0  # Ninguna
ultima_direccion = 'derecha'  # Valor por defecto

while True:
    # --- Captura y procesamiento ---
    imagen = DRON.get_frame_read().frame
    imagen = cv2.cvtColor(imagen, cv2.COLOR_RGB2BGR)
    imagen = cv2.resize(imagen, (ancho, alto))

    imagen_umbral = umbralizado(imagen)
    centro_x, centro_y = detectarCentro(imagen_umbral, imagen)
    sensores_virtuales, sensores_imgs = obtenerSalidaSensores(imagen_umbral,
        sensores)

    # --- Detectar tecla presionada ---
    key = cv2.waitKey(1) & 0xFF
```

```
if key == ord('1'):
    trayectoria = 1
    print("     Trayectoria seleccionada: LISSAJOUS")
    pesoValor = [-30, -20, 0, 20, 30]
    velAvance = 10
    sensibilidad = 4

elif key == ord('2'):
    trayectoria = 2
    print("     Trayectoria seleccionada: CUADRADO")
    pesoValor = [-35, -30, 0, 30, 35]
    velAvance = 10
    sensibilidad = 4
    ultima_direccion = 'derecha'

elif key == ord('q'):
    break

# --- Enviar comandos segun trayectoria seleccionada ---
if trayectoria == 1:
   izq_derech, rotacion = envioComandos_lissajous(sensores_virtuales,
       centro_x)
elif trayectoria == 2:
     izq_derech, rotacion = envioComandos_cuadrado(sensores_virtuales,
         centro_x)
else:
     izq_derech, rotacion = 0, 0

# --- Crear lienzo principal ---
lienzo_height = 80 + alto + (alto // 2) + 200
lienzo_width = ancho * 2 + 240
lienzo = np.zeros((lienzo_height, lienzo_width, 3), dtype=np.uint8)
lienzo[:] = (70, 30, 0)

# --- Titulo principal ---
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 0.8
thickness = 2
color_texto = (255, 255, 255)
color_sombra = (0, 0, 0)
font_info = font
font_scale_info = 0.7
thickness_info = 2

titulo = "Seguimiento Horizontal"
(text_width, text_height), baseline = cv2.getTextSize(titulo, font,
    font_scale, thickness)
text_x = (lienzo_width - text_width) // 2
text_y = 40
cv2.putText(lienzo, titulo, (text_x + 2, text_y + 2), font, font_scale,
    color_sombra, thickness + 2)
cv2.putText(lienzo, titulo, (text_x, text_y), font, font_scale,
    color_texto, thickness)

# --- Escalado y titulos de imagenes ---
escala_arriba = 0.9
ancho_reducido = int(ancho * escala_arriba)
alto_reducido = int(alto * escala_arriba)
imagen_real_peq = cv2.resize(imagen, (ancho_reducido, alto_reducido))
imagen_umbral_peq = cv2.resize(cv2.cvtColor(imagen_umbral, cv2.
    COLOR_GRAY2BGR), (ancho_reducido, alto_reducido))

offset_x = 280   # Espacio reservado para el menu visual
lienzo[80:80 + alto_reducido, offset_x:offset_x + ancho_reducido] =
    imagen_real_peq
lienzo[80:80 + alto_reducido, offset_x + ancho_reducido + 20:offset_x +
    ancho_reducido + 20 + ancho_reducido] = imagen_umbral_peq

# Borde para imagen real
cv2.rectangle(lienzo,
              (offset_x, 80),
              (offset_x + ancho_reducido, 80 + alto_reducido),
              (255, 255, 255), 2)

# Borde para imagen umbralizada
umbral_x = offset_x + ancho_reducido + 20
cv2.rectangle(lienzo,
              (umbral_x, 80),
```

```
                (umbral_x + ancho_reducido, 80 + alto_reducido),
                (255, 255, 255), 2)
# --- Titulos de las imagenes ---
# Titulo Imagen Real
titulo_real = "Imagen Real"
(text_w_real, _), _ = cv2.getTextSize(titulo_real, font, 0.6, 2)
pos_x_real = offset_x + (ancho_reducido - text_w_real) // 2
cv2.putText(lienzo, titulo_real, (pos_x_real, 70), font, 0.6, (255, 255,
     255), 2)
# Titulo Imagen Umbralizada
titulo_umbral = "Imagen Umbralizada"
(text_w_umbral, _), _ = cv2.getTextSize(titulo_umbral, font, 0.6, 2)
offset_x_umbral = offset_x + ancho_reducido + 20
pos_x_umbral = offset_x_umbral + (ancho_reducido - text_w_umbral) // 2
cv2.putText(lienzo, titulo_umbral, (pos_x_umbral, 70), font, 0.6, (255,
    255, 255), 2)
# --- Mostrar sensores virtuales ---
sensor_images = []
sensor_width = ancho // sensores
for img in np.hsplit(imagen_umbral, sensores):
    sensor_color = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
    sensor_color = cv2.resize(sensor_color, (sensor_width, 180))
    sensor_images.append(sensor_color)

separador = np.full((180, 10, 3), (70, 30, 0), dtype=np.uint8)
sensores_final = sensor_images[0]
for img_sensor in sensor_images[1:]:
    sensores_final = np.hstack((sensores_final, separador, img_sensor))
inicio_y = alto_reducido + 150
inicio_x = (lienzo_width - sensores_final.shape[1]) // 2 + 125

titulo_sensores = "Simulacion de Sensores"
(text_w_sens, _), _ = cv2.getTextSize(titulo_sensores, font, 0.6, 2)
pos_x_sens = inicio_x + (sensores_final.shape[1] - text_w_sens) // 2
cv2.putText(lienzo, titulo_sensores, (pos_x_sens, inicio_y - 10), font,
    0.6, (255, 255, 255), 2)

lienzo[inicio_y:inicio_y + 180, inicio_x:inicio_x + sensores_final.shape
    [1]] = sensores_final
cv2.rectangle(lienzo, (inicio_x, inicio_y), (inicio_x + sensores_final.
    shape[1], inicio_y + 180), (255, 255, 255), 2)

# --- Mostrar parametros ---
error = centro_x - (ancho // 2)
porcentaje_cercania = round(max(0, (1 - abs(error) / (ancho / 2))) *
    100, 1)
info_inicio_x = 30
info_inicio_y = lienzo_height - 350
espaciado_lineas = 30

cv2.putText(lienzo, f"Centro X: {centro_x}", (info_inicio_x,
    info_inicio_y), font_info, font_scale_info, (255, 255, 255),
    thickness_info)
cv2.putText(lienzo, f"Centro Y: {centro_y}", (info_inicio_x,
    info_inicio_y + espaciado_lineas), font_info, font_scale_info, (255,
     255, 255), thickness_info)
cv2.putText(lienzo, f"Traslacion: {izq_derech}", (info_inicio_x,
    info_inicio_y + 2 * espaciado_lineas), font_info, font_scale_info,
    (0, 255, 255), thickness_info)
cv2.putText(lienzo, f"Rotacion: {rotacion}", (info_inicio_x,
    info_inicio_y + 3 * espaciado_lineas), font_info, font_scale_info,
    (0, 255, 255), thickness_info)
cv2.putText(lienzo, f"Error: {error}", (info_inicio_x, info_inicio_y + 4
     * espaciado_lineas), font_info, font_scale_info, (0, 165, 255),
    thickness_info)
cv2.putText(lienzo, f"Cercania: {porcentaje_cercania}%", (info_inicio_x,
     info_inicio_y + 5 * espaciado_lineas), font_info, font_scale_info,
    (0, 255, 0), thickness_info)
# --- Mostrar bateria en la Imagen Real ---
bateria = DRON.get_battery()
texto_bateria = f"Bateria: {bateria}%"
# Posicion para la parte inferior derecha
```

```
pos_x = lienzo_width - 30 - cv2.getTextSize(texto_bateria, cv2.
    FONT_HERSHEY_SIMPLEX, 0.7, 2)[0][0]
pos_y = lienzo_height - 200  # Un poco arriba del borde inferior

# Dibujar texto en la imagen real
cv2.putText(lienzo, texto_bateria, (pos_x, pos_y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

# --- Menu de trayectoria ---
# === Configuracion general del menu ===
menu_x = 30
menu_y = 160
cuadro_ancho = 180
cuadro_alto = 40
espaciado_vertical = 40
font_menu = cv2.FONT_HERSHEY_SIMPLEX
font_scale_info = 0.6
thickness_info = 2
padding = 6
offset_interno = -50

opciones_trayectoria = {
    1: "Lissajous",
    2: "Cuadrado",
}

# === Calcular altura total del menu para dibujar fondo ===
num_opciones = len(opciones_trayectoria)
alto_opciones = num_opciones * (cuadro_alto + espaciado_vertical) -
    espaciado_vertical
alto_total_menu = 30 + alto_opciones + 0 + 20 # titulo + opciones + "
    Trayectoria modo" + margen inferior

ancho_menu = 240
x_fondo = menu_x - 20
y_fondo = menu_y - 80

# === Fondo del menu (azul marino o gris oscuro) ===
cv2.rectangle(lienzo, (x_fondo, y_fondo),
              (x_fondo + ancho_menu, y_fondo + alto_total_menu),
              (70, 30, 0), -1)

# === Texto centrado arriba del recuadro blanco ===
titulo_menu = "Menu de opciones"
font_titulo = cv2.FONT_HERSHEY_SIMPLEX
escala_titulo = 0.6
grosor_titulo = 2

(t_w, t_h), _ = cv2.getTextSize(titulo_menu, font_titulo, escala_titulo,
     grosor_titulo)
x_titulo = x_fondo + (ancho_menu - t_w) // 2
y_titulo = y_fondo - 10  # Un poco arriba del borde blanco

cv2.putText(lienzo, titulo_menu, (x_titulo, y_titulo), font_titulo,
    escala_titulo, (255, 255, 255), grosor_titulo)

# === Borde blanco del menu ===
cv2.rectangle(lienzo, (x_fondo, y_fondo),
              (x_fondo + ancho_menu, y_fondo + alto_total_menu),
              (255, 255, 255), 2)

# === Titulo del menu ===
cv2.putText(lienzo, "Seleccione trayectoria:",
            (menu_x, menu_y - 10 + offset_interno),
            font_menu, 0.5, (255, 255, 255), 1)

# === Dibujar las opciones ===
for idx, nombre in opciones_trayectoria.items():
    y_actual = menu_y + offset_interno + (idx - 1) * (cuadro_alto +
        espaciado_vertical + offset_interno + 20)

    seleccionado = trayectoria == idx
    color_fondo = (0, 255, 0) if seleccionado else (200, 200, 200)
    color_borde = (0, 200, 0) if seleccionado else (100, 100, 100)
    grosor = 2 if seleccionado else 1

    # Dibujar recuadro
    cv2.rectangle(lienzo, (menu_x, y_actual),
                  (menu_x + cuadro_ancho, y_actual + cuadro_alto),
                  color_borde, grosor)
```

```
            cv2.rectangle(lienzo, (menu_x + 1, y_actual + 1),
                          (menu_x + cuadro_ancho - 1, y_actual + cuadro_alto -
                              1),
                          color_fondo, -1)

            # Texto centrado
            # Texto alineado a la izquierda con numeracion
            nombre_con_numero = f"{idx}. {nombre}"
            text_size, _ = cv2.getTextSize(nombre_con_numero, font_menu, 0.6, 2)
            text_x = menu_x + 10  # Margen izquierdo
            text_y = y_actual + (cuadro_alto + text_size[1]) // 2
            cv2.putText(lienzo, nombre_con_numero, (text_x, text_y), font_menu,
                0.6, (0, 0, 0), 2)

        # === Mostrar texto: Trayectoria modo ===
        # Mostrar siempre "Trayectoria modo:"
        nombre_trayectoria = opciones_trayectoria.get(trayectoria, "")
        texto_modo = f"Trayectoria modo: {nombre_trayectoria.upper()}"
        (w_modo, h_modo), _ = cv2.getTextSize(texto_modo, font_menu, 0.5, 1)

        x_modo = menu_x - 15  # Ajustar m s a la izquierda si es necesario
        y_modo = y_actual + cuadro_alto + 30

        cv2.putText(lienzo, texto_modo, (x_modo, y_modo), font_menu, 0.5, (255,
            255, 255), 1)

        # --- Mostrar ventana ---
        cv2.imshow("Seguimiento Horizontal", lienzo)

# --- Finalizar ---
DRON.land()
time.sleep(1)
DRON.streamoff()
cv2.destroyAllWindows()
```

Anexo C: Código en PyCharm para seguimiento vertical

```
import cv2
import numpy as np
import time
from djitellopy import tello

# ========== INICIALIZACION ==========
DRON = tello.Tello()
DRON.connect()
DRON.streamon()
DRON.takeoff()
time.sleep(1)

print("Bater a:", DRON.get_battery())

# ========== PARAMETROS ==========
valores_hsv = [0,0,0,179,255,75]  # HSV para l nea negra
ancho, alto = 480, 360

# Variables globales compartidas
fase = ""
direccion_curva = None
tiempo_inicio_curva = 0

# Funcion de umbralizado
def umbralizado(imagen):
    hsv = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)
    menor = np.array(valores_hsv[:3])
    mayor = np.array(valores_hsv[3:])
    return cv2.inRange(hsv, menor, mayor)

# Sensores virtuales en dos franjas
def sensores_virtuales(mascara):
    # Dividir la imagen en dos franjas horizontales
    franja_alta = mascara[80:140, :]      # imagen[filas_inicial:filas_final,
        columnas], Significa: toma las filas desde la 80 hasta la 139 (en
        total 60 filas).
    franja_baja = mascara[250:310, :]     # Significa: toma las filas desde
        la 250 hasta la 309 (otras 60 filas).

    tercio = ancho // 3   # Se divide el ancho total de la imagen (480
        p xeles) en tres partes iguales
    sensores = {}         # Aqu  se va a guardar el conteo de p xeles
        blancos en cada parte de cada franja

    for nombre, franja in [("alta", franja_alta), ("baja", franja_baja)]:
        # Bucle for con desempacado, que recorre la lista:
        izq = cv2.countNonZero(franja[:, :tercio])            # Todas las
            filas (:), columnas 0  159
        cent = cv2.countNonZero(franja[:, tercio:2*tercio])   # Todas las
            filas, columnas 160  319
        der = cv2.countNonZero(franja[:, 2*tercio:])          # Todas las
            filas, columnas 320  479

        # |---- IZQ (0  159  ) ----|---- CENT (160  319  ) ----|---- DER (320
            479  ) ----|
        sensores[nombre] = (izq, cent, der)                   # Guarda los
            valores en el diccionario

    return sensores, franja_alta, franja_baja

# Movimiento segun la fase
def mover_dron_cuadrado(fase, sensores):
    global direccion_curva, tiempo_subida_inicial, contador_bajada,
        pasos_en_curva
    global tiempo_espera_curva_post_bajada, tiempo_inicio_realineacion

    # Sensores de l nea en franja alta y baja
    izq_a, cent_a, der_a = sensores["alta"]
    izq_b, cent_b, der_b = sensores["baja"]

    mov_lateral = 0
    mov_vertical = 0

    print(f"\n[INFO] Fase actual: {fase}")
    print(f"     Sensores Alta -> Izq: {izq_a}, Cent: {cent_a}, Der: {der_a}
```

```
    ")
print(f"     Sensores Baja -> Izq: {izq_b}, Cent: {cent_b}, Der: {der_b}
    ")

if fase == "subida_inicial":
    print("[INFO] Fase: SUBIDA INICIAL - Subiendo sin analizar sensores"
        )
    mov_vertical = vel_vertical

    if time.time() - tiempo_subida_inicial > 1.0:
        print("[INFO] Fin de subida inicial -> cambiando a fase:
            SUBIENDO")
        return "subiendo", 0, 0

    return "subida_inicial", 0, mov_vertical

elif fase == "subiendo":
    if cent_a > izq_a and cent_a > der_a and cent_a > 1000:
        print("[INFO] Subiendo recto")
        mov_vertical = vel_vertical
    elif izq_a > cent_a and izq_a > der_a:
        print("[INFO] Detecci n de curva izquierda -> cambiando a fase:
             CURVA_IZQ")
        mov_lateral = -vel_lateral
        direccion_curva = "subida"
        pasos_en_curva = 0
        return "curva_izq", mov_lateral, mov_vertical
    elif der_a > cent_a and der_a > izq_a:
        print("[INFO] Detecci n de curva derecha -> cambiando a fase:
            CURVA_DER")
        mov_lateral = vel_lateral
        direccion_curva = "subida"
        pasos_en_curva = 0
        return "curva_der", mov_lateral, mov_vertical

elif fase == "curva_izq":
    pasos_en_curva += 1
    print(f"[DEBUG] Pasos en curva izquierda: {pasos_en_curva}")
    print(f"[DEBUG] Direcci n de curva: {direccion_curva}")

    if izq_b > cent_b and izq_b > der_b and izq_b > 1500 and
        pasos_en_curva > 60:
        if direccion_curva == "subida":
            print("[INFO] Final de curva izquierda -> cambiando a fase:
                BAJANDO")
            mov_vertical = -vel_vertical
            return "bajando", 0, mov_vertical
        else:
            # NUEVA LOGICA: Esperar en la esquina tras la curva despu s
                 de bajar
            if tiempo_espera_curva_post_bajada is None:
                tiempo_espera_curva_post_bajada = time.time()
                print("[INFO] Esperando en esquina despu s de curva
                    bajada")
                return "curva_izq", 0, 0

            if time.time() - tiempo_espera_curva_post_bajada > 2.5:
                tiempo_espera_curva_post_bajada = None
                if cent_a > 500:
                    print("[INFO] L nea detectada arriba -> cambiando a
                         fase: SUBIENDO")
                    return "subiendo", 0, vel_vertical
                else:
                    print("[INFO] No se detect  l nea arriba ->
                        cambiando a fase: REALINEANDO")
                    return "realineando", 0, 0
            else:
                return "curva_izq", 0, 0  # Quieto esperando
    else:
        print("[INFO] En curva izquierda")
        mov_lateral = -vel_lateral

elif fase == "curva_der":
    pasos_en_curva += 1
    print(f"[DEBUG] Pasos en curva derecha: {pasos_en_curva}")
    print(f"[DEBUG] Direcci n de curva: {direccion_curva}")

    if der_b > cent_b and der_b > izq_b and der_b > 1500 and
        pasos_en_curva > 60:
```

```
            if direccion_curva == "subida":
                print("[INFO] Final de curva derecha -> cambiando a fase:
                    BAJANDO")
                mov_vertical = -vel_vertical
                return "bajando", 0, mov_vertical
            else:
                if tiempo_espera_curva_post_bajada is None:
                    tiempo_espera_curva_post_bajada = time.time()
                    print("[INFO] Esperando en esquina despu s de curva
                        bajada")
                    return "curva_der", 0, 0

                if time.time() - tiempo_espera_curva_post_bajada > 2.5:
                    tiempo_espera_curva_post_bajada = None
                    if cent_a > 500:
                        print("[INFO] L nea detectada arriba -> cambiando a
                             fase: SUBIENDO")
                        return "subiendo", 0, vel_vertical
                    else:
                        print("[INFO] No se detect  l nea arriba ->
                            cambiando a fase: REALINEANDO")
                        return "realineando", 0, 0
                else:
                    return "curva_der", 0, 0  # Quieto esperando
        else:
            print("[INFO] En curva derecha")
            mov_lateral = vel_lateral

    elif fase == "bajando":
        total_linea_baja = izq_b + cent_b + der_b
        print(f"[DEBUG] Total l nea baja: {total_linea_baja}")

        if total_linea_baja > 100:
            mov_vertical = -vel_vertical
            contador_bajada = 0
            print("[INFO] Bajando - l nea detectada")
        elif contador_bajada < 80:
            mov_vertical = -vel_vertical
            contador_bajada += 1
            print(f"[INFO] Forzando bajada adicional - paso {contador_bajada
                }")
        else:
            # Evaluar transicion a curva despues de bajada
            if der_a > cent_a and der_a > izq_a and der_a > 800:
                print("[INFO] Cambio de fase: bajando -> curva_izq (ajustado
                     por reflejo)")
                direccion_curva = "bajada"
                contador_bajada = 0
                return "curva_der", vel_lateral, 0
            elif izq_a > cent_a and izq_a > der_a and izq_a > 800:
                print("[INFO] Cambio de fase: bajando -> curva_der (ajustado
                     por reflejo)")
                direccion_curva = "bajada"
                contador_bajada = 0
                return "curva_izq", -vel_lateral, 0
            else:
                mov_vertical = 0

    elif fase == "realineando":
        if tiempo_inicio_realineacion is None:
            tiempo_inicio_realineacion = time.time()
            print("[INFO] Entrando a fase: REALINEANDO")

        tiempo_actual = time.time()
        duracion_realineacion = tiempo_actual - tiempo_inicio_realineacion

        print(f"[INFO] Tiempo en realineando: {duracion_realineacion:.2f} s"
            )

        # Subida suave constante
        mov_vertical = vel_vertical // 2

        # Ajuste lateral segun balance de sensores izquierdo/derecho
        if izq_a > der_a + 300:
            mov_lateral = -vel_lateral // 2
            print("[INFO] Ajuste lateral: moviendo a la IZQUIERDA para
                centrar")
        elif der_a > izq_a + 300:
```

```
            mov_lateral = vel_lateral // 2
            print("[INFO] Ajuste lateral: moviendo a la DERECHA para centrar
                ")
        else:
            mov_lateral = 0
            print("[INFO] Dron centrado lateralmente")

        # Condicion para considerar que ya puede volver a subir
        if duracion_realineacion > 0.8 and cent_a > 300:
            print("[INFO] L nea centrada detectada -> cambiando a fase:
                SUBIENDO")
            tiempo_inicio_realineacion = None
            return "subiendo", 0, 0

        # Failsafe por exceso de tiempo
        if duracion_realineacion > 5.0:
            print("[WARNING] Tiempo excedido en realineando -> forzando
                cambio a SUBIENDO")
            tiempo_inicio_realineacion = None
            return "subiendo", 0, 0

        return "realineando", mov_lateral, mov_vertical

    return fase, mov_lateral, mov_vertical

# Movimiento seg n la fase
def mover_dron_triangulo(fase, sensores):
    global direccion_curva, tiempo_subida_inicial, contador_bajada,
        pasos_en_curva
    global tiempo_espera_curva_post_bajada, tiempo_inicio_realineacion
    global tiempo_inicio_subida_diag, tiempo_inicio_bajada_diag,
        direccion_previa_bajada
    global tiempo_inicio_curva

    izq_a, cent_a, der_a = sensores["alta"]
    izq_b, cent_b, der_b = sensores["baja"]

    mov_lateral = 0
    mov_vertical = 0

    print(f"\n[INFO] Fase actual: {fase}")
    print(f"     Sensores Alta -> Izq: {izq_a}, Cent: {cent_a}, Der: {der_a}
        ")
    print(f"     Sensores Baja -> Izq: {izq_b}, Cent: {cent_b}, Der: {der_b}
        ")

    # Emergencia: no se detecta linea en ningun lado
    if all(v < 300 for v in [izq_a, cent_a, der_a, izq_b, cent_b, der_b]):
        print("[ALERTA] L nea no detectada -> estabilizando")
        return "esperando_subida", 0, 0

    if fase == "subida_inicial":
        print("[INFO] Fase: SUBIDA INICIAL - Subiendo sin analizar sensores"
            )
        mov_vertical = vel_vertical

        if time.time() - tiempo_subida_inicial > 1.0:
            print("[INFO] Fin de subida inicial -> esperando detecci n para
                subida diagonal")
            return "esperando_subida", 0, 0
        return "subida_inicial", 0, mov_vertical

    elif fase == "esperando_subida":
        if izq_a > 800 and (izq_a >= cent_a and izq_a >= der_a):
            print("[INFO] Detecci n en franja alta izquierda -> SUBIDA
                DIAGONAL IZQ")
            tiempo_inicio_subida_diag = time.time()
            direccion_curva = "izq"
            return "subiendo_diag_izq", -vel_lateral, vel_vertical

        elif der_a > 800 and (der_a >= cent_a and der_a >= izq_a):
            print("[INFO] Detecci n en franja alta derecha -> SUBIDA
                DIAGONAL DER")
            tiempo_inicio_subida_diag = time.time()
            direccion_curva = "der"
            return "subiendo_diag_der", vel_lateral, vel_vertical

        elif cent_a > 800:
            # Nueva condicion para permitir subida si la linea esta centrada
            if direccion_previa_bajada == "izq":
```

```
                print("[INFO] Detecci n en franja alta central -> SUBIDA
                    DIAGONAL IZQ (por direcci n previa)")
                tiempo_inicio_subida_diag = time.time()
                direccion_curva = "izq"
                return "subiendo_diag_izq", -vel_lateral, vel_vertical
            elif direccion_previa_bajada == "der":
                print("[INFO] Detecci n en franja alta central -> SUBIDA
                    DIAGONAL DER (por direcci n previa)")
                tiempo_inicio_subida_diag = time.time()
                direccion_curva = "der"
                return "subiendo_diag_der", vel_lateral, vel_vertical
        return "esperando_subida", 0, 0

    elif fase == "subiendo_diag_izq":
        if time.time() - tiempo_inicio_subida_diag < 1.2:
            print("[INFO] Subiendo diagonal IZQ (tiempo m nimo activo)")
            return "subiendo_diag_izq", -vel_lateral, vel_vertical

        vel_lateral_bajada = int(vel_lateral * 0.3)  # 0.7

        if izq_a < 400 and (izq_b > 200 or cent_b > 200):
            print("[INFO] L nea ya no detectada arriba, detectada abajo ->
                BAJADA DIAGONAL IZQ")
            tiempo_inicio_bajada_diag = time.time()
            direccion_curva = "izq"
            return "bajando_diag_izq", -vel_lateral_bajada, -vel_vertical
        else:
            return "subiendo_diag_izq", -vel_lateral, vel_vertical

    elif fase == "subiendo_diag_der":
        if time.time() - tiempo_inicio_subida_diag < 1.2:
            print("[INFO] Subiendo diagonal DER (tiempo m nimo activo)")
            return "subiendo_diag_der", vel_lateral, vel_vertical

        if der_a < 400 and (der_b > 800 or cent_b > 800):
            print("[INFO] L nea ya no detectada arriba, detectada abajo ->
                BAJADA DIAGONAL DER")
            tiempo_inicio_bajada_diag = time.time()
            direccion_curva = "der"
            return "bajando_diag_der", vel_lateral, -vel_vertical
        else:
            return "subiendo_diag_der", vel_lateral, vel_vertical

    elif fase == "bajando_diag_izq":
        if time.time() - tiempo_inicio_bajada_diag < 2.5:  #3.0
            print("[INFO] Bajando diagonal IZQ (tiempo m nimo activado)")
            return "bajando_diag_izq", -vel_lateral, -vel_vertical

        # Reduccion  de velocidad lateral tras tiempo minimo
        vel_lateral_bajada = int(vel_lateral * 0.3) # 0.7

        if izq_b < 100 and (der_a > 100 or cent_a > 100):
            print("[INFO] Bajada terminada -> cambiando a CURVA_DER")
            pasos_en_curva = 0
            direccion_previa_bajada = "izq"  ###
            tiempo_inicio_curva = time.time()
            return "curva_der", vel_lateral, 0
        else:
            return "bajando_diag_izq", -vel_lateral_bajada, -vel_vertical

    elif fase == "bajando_diag_der":
        if time.time() - tiempo_inicio_bajada_diag < 2.5:   #3.0
            print("[INFO] Bajando diagonal DER (tiempo m nimo activo)")
            return "bajando_diag_der", vel_lateral, -vel_vertical

        # Reduccion de velocidad lateral tras tiempo minimo
        vel_lateral_bajada = int(vel_lateral * 0.3)

        if der_b < 100 and (izq_a > 100 or cent_a > 100):
            print("[INFO] Bajada terminada -> cambiando a CURVA_IZQ")
            pasos_en_curva = 0
            direccion_previa_bajada = "der"  ###
            tiempo_inicio_curva = time.time()
            return "curva_izq", -vel_lateral, 0
        else:
            return "bajando_diag_der", vel_lateral_bajada, -vel_vertical

    elif fase == "curva_izq":
```

```
            pasos_en_curva += 1
            print(f"[DEBUG] Pasos en curva izquierda: {pasos_en_curva}")

            if time.time() - tiempo_inicio_curva < 4.0:
                return "curva_izq", -vel_lateral, 0

            if pasos_en_curva > 60 and cent_a > 800:
                if direccion_previa_bajada == "der":
                    print("[INFO] L nea detectada al centro -> SUBIDA DIAGONAL
                        DER")  #IZQ
                    tiempo_inicio_subida_diag = time.time()
                    direccion_curva = "der"   # izq
                    return "subiendo_diag_der", vel_lateral, vel_vertical
            return "curva_izq", -vel_lateral, 0

        elif fase == "curva_der":
            pasos_en_curva += 1
            print(f"[DEBUG] Pasos en curva derecha: {pasos_en_curva}")

            if time.time() - tiempo_inicio_curva < 4.0:
                return "curva_der", vel_lateral, 0

            if pasos_en_curva > 60 and cent_a > 800:
                if direccion_previa_bajada == "izq":
                    print("[INFO] L nea detectada al centro -> SUBIDA DIAGONAL
                        IZQ")  #DER
                    tiempo_inicio_subida_diag = time.time()
                    direccion_curva = "izq"
                    return "subiendo_diag_izq", -vel_lateral, -vel_vertical
            return "curva_der", vel_lateral, 0

        return fase, mov_lateral, mov_vertical

    def mover_dron_circulo(fase, sensores):
        global direccion_curva, tiempo_inicio_curva, tiempo_inicio_bajada

        izq_a, cent_a, der_a = sensores["alta"]
        izq_b, cent_b, der_b = sensores["baja"]

        umbral_sensor = 500
        print(f"\n[INFO] Fase actual: {fase}")
        print(f"      Sensores Alta -> Izq: {izq_a}, Cent: {cent_a}, Der: {der_a}
            ")
        print(f"      Sensores Baja -> Izq: {izq_b}, Cent: {cent_b}, Der: {der_b}
            ")

        # Fase: Subida Diagonal hacia la izquierda
        if fase == "subiendo_diag_izq":
            tiempo_inicio_bajada = None  # reinicio inmediato
            print("[INFO] Fase: SUBIDA DIAGONAL IZQ")
            if cent_a < umbral_sensor and izq_a < umbral_sensor:
                print("[INFO] Fin de subida -> curva superior izquierda")
                direccion_curva = "izq"
                tiempo_inicio_curva = time.time()
                return "curva_sup_izq", -vel_lateral, 0
            return "subiendo_diag_izq", -vel_lateral, vel_vertical

        # Fase: Subida Diagonal hacia la derecha
        elif fase == "subiendo_diag_der":
            tiempo_inicio_bajada = None  # reinicio inmediato
            print("[INFO] Fase: SUBIDA DIAGONAL DER")
            if cent_a < umbral_sensor and der_a < umbral_sensor:
                print("[INFO] Fin de subida -> Subida diagonal izquierda")
                #direccion_curva = "der"
                tiempo_inicio_curva = time.time()
                return "subiendo_diag_izq", -vel_lateral, vel_vertical
            return "subiendo_diag_der", vel_lateral, vel_vertical

        # Fase: Curva superior izquierda
        elif fase == "curva_sup_izq":
            tiempo_inicio_bajada = None  # reinicio inmediato
            print("[INFO] Fase: CURVA SUPERIOR IZQ")
            if time.time() - tiempo_inicio_curva > duracion_curva:
                print("[INFO] Fin curva sup izq -> bajando diagonal izq")
                return "bajando_diag_izq", -vel_lateral, -vel_vertical
            return "curva_sup_izq", -vel_lateral, 0

        # Fase: Curva superior derecha
        elif fase == "curva_sup_der":
```

```
        tiempo_inicio_bajada = None  # reinicio inmediato
        print("[INFO] Fase: CURVA SUPERIOR DER")
        if time.time() - tiempo_inicio_curva > duracion_curva:
            print("[INFO] Fin curva sup der -> bajando diagonal der")
            return "bajando_diag_der", vel_lateral, -vel_vertical
        return "curva_sup_der", vel_lateral, 0

    # Fase: Bajada diagonal izquierda
    elif fase == "bajando_diag_izq":
        print("[INFO] Fase: BAJANDO DIAGONAL IZQ")

        if tiempo_inicio_bajada is None:
            tiempo_inicio_bajada = time.time()

        if (cent_b < umbral_sensor and izq_b < umbral_sensor) or (time.time
            () - tiempo_inicio_bajada > duracion_max_bajada):
            print("[INFO] Fin de bajada izq -> bajando diagonal der (suaviza
                giro)")
            tiempo_inicio_bajada = None
            tiempo_inicio_curva = time.time()
            return "bajando_diag_der_pre_curva", vel_lateral, -vel_vertical
        return "bajando_diag_izq", -vel_lateral, -vel_vertical

    # NUEVA FASE: transicion bajando hacia la derecha antes de curva der
    elif fase == "bajando_diag_der_pre_curva":
        tiempo_inicio_bajada = None  # reinicio inmediato
        print("[INFO] Fase: TRANSICI N BAJANDO DIAGONAL DER (pre curva
            derecha)")
        if time.time() - tiempo_inicio_curva > 2.0:  # duraci n breve (~1s)
            print("[INFO] Fin transici n -> curva inferior derecha")
            direccion_curva = "der"
            tiempo_inicio_curva = time.time()
            return "curva_inf_der", vel_lateral, 0
        return "bajando_diag_der_pre_curva", vel_lateral, -vel_vertical

    # Fase: Bajada diagonal derecha
    elif fase == "bajando_diag_der":
        print("[INFO] Fase: BAJANDO DIAGONAL DER")

        if tiempo_inicio_bajada is None:
            tiempo_inicio_bajada = time.time()

        if (cent_b < umbral_sensor and der_b < umbral_sensor) or (time.time
            () - tiempo_inicio_bajada > duracion_max_bajada):
            print("[INFO] Fin de bajada der -> bajando diagonal izq (suaviza
                giro)")
            tiempo_inicio_bajada = None
            tiempo_inicio_curva = time.time()
            return "bajando_diag_izq_pre_curva", -vel_lateral, -vel_vertical
        return "bajando_diag_der", vel_lateral, -vel_vertical

    # NUEVA FASE: transicion bajando hacia la izquierda antes de curva izq
    elif fase == "bajando_diag_izq_pre_curva":
        tiempo_inicio_bajada = None  # reinicio inmediato
        print("[INFO] Fase: TRANSICI N BAJANDO DIAGONAL IZQ (pre curva
            izquierda)")
        if time.time() - tiempo_inicio_curva > 2.0:
            print("[INFO] Fin transici n -> curva inferior izquierda")
            direccion_curva = "izq"
            tiempo_inicio_curva = time.time()
            return "curva_inf_izq", -vel_lateral, 0
        return "bajando_diag_izq_pre_curva", -vel_lateral, -vel_vertical

    # Fase: Curva inferior izquierda
    elif fase == "curva_inf_izq":
        tiempo_inicio_bajada = None  # reinicio inmediato
        print("[INFO] Fase: CURVA INFERIOR IZQ")
        if time.time() - tiempo_inicio_curva > duracion_curva:
            print("[INFO] Fin curva inf izq -> subiendo diagonal izq")
            return "subiendo_diag_izq", -vel_lateral, vel_vertical
        return "curva_inf_izq", -vel_lateral, 0

    # Fase: Curva inferior derecha
    elif fase == "curva_inf_der":
        tiempo_inicio_bajada = None  # reinicio inmediato
        print("[INFO] Fase: CURVA INFERIOR DER")
```

```
            if time.time() - tiempo_inicio_curva > duracion_curva:
                print("[INFO] Fin curva inf der -> subiendo diagonal der")
                return "subiendo_diag_der", vel_lateral, vel_vertical
            return "curva_inf_der", vel_lateral, 0

        # Fase inicial: detectar hacia donde empezar
        print("[WARN] Fase desconocida o inicial, intentando detectar inicio..."
            )
        if izq_a > umbral_sensor:
            print("[INFO] Detecci n izquierda alta -> subiendo diagonal izq")
            return "subiendo_diag_izq", -vel_lateral, vel_vertical
        elif der_a > umbral_sensor:
            print("[INFO] Detecci n derecha alta -> subiendo diagonal der")
            return "subiendo_diag_der", vel_lateral, vel_vertical

        return "esperando_inicio", 0, 0

    # Trayectorias
    def iniciar_cuadrado():
        global vel_lateral, vel_vertical, fase, tiempo_subida_inicial
        global direccion_curva, contador_bajada, tiempo_inicio_realineacion
        global pasos_en_curva, tiempo_espera_curva_post_bajada

        vel_lateral = 25  #30
        vel_vertical = 32 #45
        fase = "subida_inicial"
        tiempo_subida_inicial = time.time()
        direccion_curva = None
        contador_bajada = 0
        tiempo_inicio_realineacion = None
        pasos_en_curva = 0
        tiempo_espera_curva_post_bajada = None

    def iniciar_triangulo():
        global vel_lateral, vel_vertical, fase, tiempo_subida_inicial
        global direccion_curva, contador_bajada, tiempo_inicio_realineacion
        global pasos_en_curva, tiempo_espera_curva_post_bajada
        global tiempo_inicio_subida_diag, tiempo_inicio_bajada_diag,
            direccion_previa_bajada

        vel_lateral = 25   # 25
        vel_vertical = 32
        fase = "subida_inicial"
        tiempo_subida_inicial = time.time()
        direccion_curva = None
        contador_bajada = 0
        tiempo_inicio_realineacion = None
        pasos_en_curva = 0
        tiempo_espera_curva_post_bajada = None
        tiempo_inicio_subida_diag = 0
        tiempo_inicio_bajada_diag = 0
        direccion_previa_bajada = None

    def iniciar_circulo():
        global vel_lateral, vel_vertical, fase, direccion_curva, duracion_curva,
             tiempo_inicio_curva
        global tiempo_inicio_bajada, duracion_max_bajada

        vel_lateral = 20
        vel_vertical = 35
        fase = "esperando_inicio"
        direccion_curva = None
        duracion_curva = 1.0
        tiempo_inicio_curva = time.time()
        tiempo_inicio_bajada = None
        duracion_max_bajada = 1.5

    # ========== LOOP PRINCIPAL ==========
    print("     Presiona 1 (Cuadrado), 2 (Tri ngulo), 3 (C rculo). Q para
        salir.")

    # Crear el lienzo vacio (antes del while)
    margen_izquierdo = 200
    lienzo_ancho = ancho + margen_izquierdo + 515
    lienzo_alto = alto + 350

    # Variables de bateria
    porcentaje_bateria = 0
    tiempo_bateria = time.time()
```

```
trayectoria_actual = ""    # Puede ser "cuadrado", "triangulo", "circulo"

# ========== LOOP ==========
while True:
    # 1. Capturar imagen y procesar
    frame = DRON.get_frame_read().frame
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
    frame = cv2.resize(frame, (ancho, alto))

    mascara = umbralizado(frame)
    sensores, franja_alta, franja_baja = sensores_virtuales(mascara)

    key = cv2.waitKey(1) & 0xFF

    # Dentro del bucle principal:
    if key == ord('1'):
        iniciar_cuadrado()
        trayectoria_actual = "cuadrado"
    elif key == ord('2'):
        iniciar_triangulo()
        trayectoria_actual = "triangulo"
    elif key == ord('3'):
        iniciar_circulo()
        trayectoria_actual = "circulo"

        # Subida inicial solo para la trayectoria del c rculo
        print("[INFO] Subida inicial controlada")
        t_subida_inicial = time.time()
        while time.time() - t_subida_inicial < 3:
            DRON.send_rc_control(0, 0, 20, 0)
            time.sleep(0.1)
        DRON.send_rc_control(0, 0, 0, 0)
        print("[INFO] Subida inicial finalizada")

    elif key == ord('q'):
        break

    # Luego, dentro del bucle, despues de procesar teclas:
    if trayectoria_actual == "cuadrado":
        fase, mov_lateral, mov_vertical = mover_dron_cuadrado(fase, sensores
            )
        DRON.send_rc_control(mov_lateral, 0, mov_vertical, 0)
    elif trayectoria_actual == "triangulo":
        fase, mov_lateral, mov_vertical = mover_dron_triangulo(fase,
            sensores)
        DRON.send_rc_control(mov_lateral, 0, mov_vertical, 0)
    elif trayectoria_actual == "circulo":
        fase, mov_lateral, mov_vertical = mover_dron_circulo(fase, sensores)
        DRON.send_rc_control(mov_lateral, 0, mov_vertical, 0)
    else:
        # Si no hay trayectoria activa, detener el dron (opcional)
        DRON.send_rc_control(0, 0, 0, 0)

# Crear lienzo
 # === Dibujar cuadros que indican las franjas en el frame original ===
    alto_frame, ancho_frame, _ = frame.shape

    # Coordenadas horizontales (ancho)
    x_inicio = int(ancho_frame * 0.1) # 0.1
    x_final = int(ancho_frame * 0.9)  # 0.9

    # Coordenadas verticales para franja alta
    y_franja_alta_inicio = int(alto_frame * 0.20) #0.4
    y_franja_alta_final = int(alto_frame * 0.40)  #0.6

    # Coordenadas verticales para franja baja
    y_franja_baja_inicio = int(alto_frame * 0.7)
    y_franja_baja_final = int(alto_frame * 0.9)

    # Color de las lineas
    color_lineas = (255, 70, 0)

    # === Dibujar cuadros para Franja Alta ===
    cv2.line(frame, (x_inicio, y_franja_alta_inicio), (x_final,
        y_franja_alta_inicio), color_lineas, 3)  # arriba
    cv2.line(frame, (x_inicio, y_franja_alta_final), (x_final,
        y_franja_alta_final), color_lineas, 3)  # abajo
```

```
cv2.line(frame, (x_inicio, y_franja_alta_inicio), (x_inicio,
    y_franja_alta_final), color_lineas, 3)  # izquierda
cv2.line(frame, (x_final, y_franja_alta_inicio), (x_final,
    y_franja_alta_final), color_lineas, 3)  # derecha

# === Dibujar cuadros para Franja Baja ===
cv2.line(frame, (x_inicio, y_franja_baja_inicio), (x_final,
    y_franja_baja_inicio), color_lineas, 3)  # arriba
cv2.line(frame, (x_inicio, y_franja_baja_final), (x_final,
    y_franja_baja_final), color_lineas, 3)  # abajo
cv2.line(frame, (x_inicio, y_franja_baja_inicio), (x_inicio,
    y_franja_baja_final), color_lineas, 3)  # izquierda
cv2.line(frame, (x_final, y_franja_baja_inicio), (x_final,
    y_franja_baja_final), color_lineas, 3)  # derecha

# === Dibujar divisiones verticales en las franjas (para sensores Izq -
    Centro - Der) ===

# Divisiones en 3 partes
ancho_franja = x_final - x_inicio
x_izq = x_inicio + ancho_franja // 3
x_der = x_inicio + 2 * ancho_franja // 3

# Para Franja Alta
cv2.line(frame, (x_izq, y_franja_alta_inicio), (x_izq,
    y_franja_alta_final), color_lineas, 3)
cv2.line(frame, (x_der, y_franja_alta_inicio), (x_der,
    y_franja_alta_final), color_lineas, 3)

# Para Franja Baja
cv2.line(frame, (x_izq, y_franja_baja_inicio), (x_izq,
    y_franja_baja_final), color_lineas, 3)
cv2.line(frame, (x_der, y_franja_baja_inicio), (x_der,
    y_franja_baja_final), color_lineas, 3)


# === Etiquetas de sensores: Izquierda (I), Centro (C), Derecha (D) ===

# Funcion para dibujar la letra centrada
def dibujar_letra(texto, x_centro, y_pos, img, color=(255, 70, 0)):
    (w, h), _ = cv2.getTextSize(texto, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)
    x_text = x_centro - w // 2
    y_text = y_pos + h // 2
    cv2.putText(img, texto, (x_text, y_text), cv2.FONT_HERSHEY_SIMPLEX,
        0.7, color, 2)

# Coordenadas de centros para cada zona
x_centro_izq = (x_inicio + x_izq) // 2
x_centro_cen = (x_izq + x_der) // 2
x_centro_der = (x_der + x_final) // 2

# Para franja alta
y_texto_franja_alta = y_franja_alta_inicio - 10  # un poquito arriba
dibujar_letra("IZQ", x_centro_izq, y_texto_franja_alta, frame)
dibujar_letra("CEN", x_centro_cen, y_texto_franja_alta, frame)
dibujar_letra("DER", x_centro_der, y_texto_franja_alta, frame)

# Para franja baja
y_texto_franja_baja = y_franja_baja_inicio - 10
dibujar_letra("IZQ", x_centro_izq, y_texto_franja_baja, frame)
dibujar_letra("CEN", x_centro_cen, y_texto_franja_baja, frame)
dibujar_letra("DER", x_centro_der, y_texto_franja_baja, frame)

# 2. Texto de fase
cv2.putText(frame, f"Fase: {fase}", (10, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 70, 0), 2)

# 3. Convertir franjas a BGR
franja_alta = cv2.cvtColor(franja_alta, cv2.COLOR_GRAY2BGR)
franja_baja = cv2.cvtColor(franja_baja, cv2.COLOR_GRAY2BGR)

# 4. Ajustar tama os para imagenes (m s grandes)
ancho_img = int(ancho * 0.9)   # antes 0.8     ahora 0.9
alto_img  = int(alto  * 0.8)   # antes 0.75     ahora 0.8

# 5. Tama o de franjas (mismo ancho que antes)
ancho_franja = int(ancho * 0.8)
```

```
alto_franja  = int(alto  * 0.2)

# 6. Redimensionar
frame_redim       = cv2.resize(frame,               (ancho_img,
    alto_img   ))
mascara_redim     = cv2.resize(mascara,             (ancho_img,
    alto_img   ))
franja_alta_redim = cv2.resize(franja_alta,         (ancho_franja,
    alto_franja))
franja_baja_redim = cv2.resize(franja_baja,         (ancho_franja,
    alto_franja))

# 7. A adir borde blanco
def add_border(img):
    return cv2.copyMakeBorder(img, 2,2,2,2, cv2.BORDER_CONSTANT, value
        =(255,255,255))
frame_redim       = add_border(frame_redim)
mascara_redim     = add_border(cv2.cvtColor(mascara_redim, cv2.
    COLOR_GRAY2BGR))
franja_alta_redim = add_border(franja_alta_redim)
franja_baja_redim = add_border(franja_baja_redim)

# 8. Crear fondo azul marino
lienzo = np.full((lienzo_alto, lienzo_ancho, 3),
                 (70, 30, 0), dtype=np.uint8)

# 9. Titulo principal (con sombra mas peque o)
texto = "Seguimiento Vertical"
sz, _ = cv2.getTextSize(texto, cv2.FONT_HERSHEY_SIMPLEX, 0.8, 2)
x_txt = (lienzo_ancho - sz[0]) // 2
cv2.putText(lienzo, texto, (x_txt, 40),
            cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,0), 4)
cv2.putText(lienzo, texto, (x_txt, 40),
            cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255,255,255),2)

# 10. Calcular posiciones centradas
x_c_img = (lienzo_ancho - (ancho_img*2 + 20)) // 2 + 120
y_c_img = 80

x_c_fr  = (lienzo_ancho - ancho_franja) // 2 + 120
y_f_al  = y_c_img + alto_img + 100   # desplazado 20px mas abajo
y_f_ba  = y_f_al + alto_franja + 40  # separacion aumentada

# 11. Dibujar imagenes
lienzo[y_c_img:y_c_img+alto_img+4,
       x_c_img:x_c_img+ancho_img+4] = frame_redim
lienzo[y_c_img:y_c_img+alto_img+4,
       x_c_img+ancho_img+20 : x_c_img+2*ancho_img+24] = mascara_redim

# 12. Etiquetas sobre las imagenes, ahora centradas
text_real = "Imagen Real"
(w_real, _), _ = cv2.getTextSize(text_real, cv2.FONT_HERSHEY_SIMPLEX,
    0.6, 2)
x_text_real = x_c_img + ((ancho_img + 4) - w_real) // 2
y_text_img = y_c_img - 10
cv2.putText(lienzo, text_real,
            (x_text_real, y_text_img),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

text_umb = "Imagen Umbralizada"
(w_umb, _), _ = cv2.getTextSize(text_umb, cv2.FONT_HERSHEY_SIMPLEX, 0.6,
     2)
x_text_umb = x_c_img + ancho_img + 20 + ((ancho_img + 4) - w_umb) // 2
cv2.putText(lienzo, text_umb,
            (x_text_umb, y_text_img),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

# Texto "Simulacion de Sensores" centrado debajo de la imagen
    umbralizada
# ========================
sim_text = "Simulacion de Sensores"

# Obtener ancho del texto
(w_sim, _), _ = cv2.getTextSize(sim_text, cv2.FONT_HERSHEY_SIMPLEX, 0.6,
     2)

# Calcular centro de la imagen umbralizada
```

```
centro_ambas_imgs = x_c_img + (ancho_img + 20 + ancho_img) // 2

# Coordenadas para centrar el texto debajo de la imagen
x_sim = centro_ambas_imgs - (w_sim // 2)
y_sim = y_f_al - 45  # Puedes moverlo m s arriba o abajo cambiando este
    valor

# Dibujar el texto
cv2.putText(lienzo, sim_text,
            (x_sim, y_sim),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

# 14. Dibujar franjas
lienzo[y_f_al:y_f_al + alto_franja + 4,
       x_c_fr:x_c_fr + ancho_franja + 4] = franja_alta_redim
lienzo[y_f_ba:y_f_ba + alto_franja + 4,
       x_c_fr:x_c_fr + ancho_franja + 4] = franja_baja_redim

# 15. Etiquetas de franjas (arriba, a la izquierda)
cv2.putText(lienzo, "Franja Alta",
            (x_c_fr, y_f_al - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
cv2.putText(lienzo, "Franja Baja",
            (x_c_fr, y_f_ba - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

# === Mostrar informacion de sensores y movimientos ===

# Texto base
info_inicio_x = 30
info_inicio_y = lienzo_alto - 230  # Parte inferior izquierda
line_spacing = 25  # Menor espacio entre l neas para que quede m s
    compacto

# Datos de sensores
alta_izq, alta_cen, alta_der = sensores['alta']
baja_izq, baja_cen, baja_der = sensores['baja']

# Texto que quieres mostrar
informacion = [
    f"Alta Centro: {alta_cen} px - Mov: Subiendo",
    f"Alta Izquierda: {alta_izq} px - Mov: Izquierda",
    f"Alta Derecha: {alta_der} px - Mov: Derecha",
    f"Baja Centro: {baja_cen} px - Mov: Bajando",
    f"Baja Izquierda: {baja_izq} px - Mov: Izquierda",
    f"Baja Derecha: {baja_der} px - Mov: Derecha"
]

# Dibujar cada l nea
for i, texto in enumerate(informacion):
    y = info_inicio_y + i * line_spacing
    cv2.putText(lienzo, texto, (info_inicio_x, y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)  #
                    Letra m s peque a

# === Mostrar bateria debajo de la  ltima  l nea ===

# Actualizar bateria cada 1 segundo aprox
if time.time() - tiempo_bateria >= 1:
    porcentaje_bateria = DRON.get_battery()
    tiempo_bateria = time.time()

# Posicion bateria (debajo de todo)
y_bateria = info_inicio_y + len(informacion) * line_spacing + 10

texto_bateria = f"Bateria: {porcentaje_bateria}%"
cv2.putText(lienzo, texto_bateria,
            (info_inicio_x, y_bateria),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)  # Letra
                igual peque a, color amarillo para destacar

# === Dibujo del menu de seleccion de trayectoria con recuadros ===
menu_x = 35
menu_y = 140
espacio_entre = 50  # mas espacio vertical para los recuadros
ancho_recuadro = 180
alto_recuadro = 35
tama o_fuente = 0.6
```

```
        grosor = 2
        offset_recuadros = 5

        opciones = [
            ("1. Cuadrado", "cuadrado"),
            ("2. Triangulo", "triangulo"),
            ("3. Circulo", "circulo")
        ]
        # === Fondo del menu ===
        # Calcula el alto total del menu: "Seleccione trayectoria" + 3 opciones
            + espacio extra
        alto_menu = 30 + len(opciones) * espacio_entre + 40
        ancho_menu = 240  # ajustable segun necesidad
        x_fondo = menu_x - 20
        y_fondo = menu_y - 60

        # === Titulo centrado arriba del cuadro blanco ===
        titulo_menu = "Menu de opciones"
        (titulo_ancho, _), _ = cv2.getTextSize(titulo_menu, cv2.
            FONT_HERSHEY_SIMPLEX, 0.6, 1)
        centro_x = x_fondo + (ancho_menu - titulo_ancho) // 2
        cv2.putText(lienzo, titulo_menu, (centro_x, y_fondo - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

        # === Fondo del menu (relleno gris oscuro) ===
        cv2.rectangle(lienzo, (x_fondo, y_fondo),
                      (x_fondo + ancho_menu, y_fondo + alto_menu),
                      (70, 30, 0 ), -1)

        # === Borde blanco del menu ===
        cv2.rectangle(lienzo, (x_fondo, y_fondo),
                      (x_fondo + ancho_menu, y_fondo + alto_menu),
                      (255, 255, 255), 2)  # Grosor 2 para el borde blanco

        # Titulo del menu
        cv2.putText(lienzo, "Seleccione trayectoria:",
                    (menu_x, menu_y - 40), cv2.FONT_HERSHEY_SIMPLEX,
                    0.5, (255, 255, 255), 1)

        # Opciones dentro de recuadros
        for i, (texto, nombre_tray) in enumerate(opciones):
            y_pos = menu_y + i * espacio_entre
            color_texto = (0, 0, 0) if trayectoria_actual == nombre_tray else
                (0, 0, 0)
            color_fondo = (200, 200, 200) if trayectoria_actual != nombre_tray
                else (0, 255, 0)

            # Dibujar rectangulo de fondo
            cv2.rectangle(lienzo, (menu_x + offset_recuadros - 4, y_pos - 28),
                          (menu_x + offset_recuadros + 170, y_pos + 10),
                              color_fondo, -1)

            # Dibujar texto encima del rectangulo
            cv2.putText(lienzo, texto, (menu_x + offset_recuadros + 5, y_pos),
                        cv2.FONT_HERSHEY_SIMPLEX, tama o_fuente, color_texto,
                            grosor)

        # Mostrar trayectoria activa debajo del menu
        texto_estado = f"Trayectoria modo: {trayectoria_actual.upper()}"
        cv2.putText(lienzo, texto_estado,
                    (menu_x - 15, menu_y + len(opciones) * espacio_entre - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

        # 16. Mostrar resultado
        cv2.imshow("Seguimiento Vertical", lienzo)

    # Finalizar
    DRON.land()
    cv2.destroyAllWindows()
```

Trabajo_Integración_Curricular

2% Textos sospechosos

- < 1% **Similitudes (ignorado)**
  - < 1% similitudes entre comillas
  - < 1% entre las fuentes mencionadas
- 2% **Idiomas no reconocidos**
- 30% **Textos potencialmente generados por la IA (ignorado)**

Nombre del documento: Trabajo_Integración_Curricular.pdf
ID del documento: ff9a0e4fadc9b2b19df38a8dc1dcce20c1879be5
Tamaño del documento original: 7,73 MB

Depositante: Junior Rafael Figueroa Olmedo
Fecha de depósito: 13/6/2025
Tipo de carga: interface
fecha de fin de análisis: 13/6/2025

Número de palabras: 35.558
Número de caracteres: 234.250

Ubicación de las similitudes en el documento:

Fuentes principales detectadas

N°	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	**Trabajo_Integración_Curricular_Auto.pdf** | Trabajo_Integración_Curricu... #10c8e3 El documento proviene de mi grupo 7 fuentes similares	45%		Palabras idénticas: 45% (16.665 palabras)
2	**TESIS_MONITOREO_FINAL_4.docx** | TESIS_MONITOREO_FINAL_4 #7ff3e4 El documento proviene de mi grupo	< 1%		Palabras idénticas: < 1% (23 palabras)

Fuentes con similitudes fortuitas

N°	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	**hdl.handle.net** | Visión por computador aplicada a la navegación y la detección d... http://hdl.handle.net/10016/30155	< 1%		Palabras idénticas: < 1% (35 palabras)
2	**Taller 3. 2P.pdf** | Taller N° 3 2p (asincrónico)(813434)_Taller 3. 2P.pdf #dc32e2 El documento proviene de mi grupo	< 1%		Palabras idénticas: < 1% (18 palabras)
3	**www.academia.edu** | (PDF) " TECNOLOGÍA BLUETOOTH " Tesis para obtener el g... https://www.academia.edu/31776999/_TECNOLOGÍA_BLUETOOTH_Tesis_p	< 1%		Palabras idénticas: < 1% (16 palabras)
4	**ri-ng.uaq.mx** https://ri-ng.uaq.mx/bitstream/123456789/8996/1/IGLIN-244014.pdf	< 1%		Palabras idénticas: < 1% (20 palabras)
5	**hdl.handle.net** | Diseño de un sistema de control de dirección de un vehículo http://hdl.handle.net/10016/16015	< 1%		Palabras idénticas: < 1% (17 palabras)

Fuentes mencionadas (sin similitudes detectadas)

Estas fuentes han sido citadas en el documento sin encontrar similitudes.

1 https://dronesec.club/historia-y-origen-de-los-drones/
2 http://eldrone.es/historia-de-los-drones/
3 https://taktic.es/usos-y-aplicaciones-de-drones-en
4 https://imaginaformacion.com/tutoriales/opencv-en-python
5 https://oa.upm.es/56813/1/TFG_FEDERICO_JOSE_GONZALEZ_VICO_PUERTAS.pdf