SISTEMA AUTÓNOMO DE ATERRIZAJE PARA MICRO VEHÍCULOS AÉREOS BASADO EN YOLOV8

YOLOV8 BASED AUTONOMOUS LANDING SYSTEM FOR MICRO AERIAL VEHICLES

Alejandro Daniel Matías Pacheco
Universidad Autónoma de Querétaro, México
danielma65@gmail.com

Juan Manuel Ramos Arreguin
Universidad Autónoma de Querétaro, México
jsistdig@yahoo.com.mx

José Martínez Carranza
Instituto Nacional de Astrofísica, Óptica y Electrónica, México
carranza@inaoep.mx

Jesús Carlos Pedraza Ortega
Universidad Autónoma de Querétaro, México
caryoko@yahoo.com

Saúl Tovar Arriaga
Universidad Autónoma de Querétaro, México
saul.tovar@uaq.mx

Recepción: 14/noviembre/2024 **Aceptación:** 16/abril/2025

Resumen

La navegación autónoma de micro vehículos aéreos (MAVs) con sensores visuales ha ganado relevancia, particularmente en la etapa de aterrizaje, donde los sensores de ubicación tradicionales pueden fallar. El uso de aprendizaje profundo y visión computacional permite superar estas limitaciones, pues las cámaras brindan más información del entorno, especialmente al aterrizar en plataformas en movimiento. En este trabajo, se utiliza el detector You Only Look Once versión 8 (YOLOv8) para identificar un marcador de aterrizaje tipo Quick Response (QR). Se entrena un Perceptrón Multicapa (MLP) para estimar la altura relativa entre el MAV-plataforma móvil. Un controlador Proporcional-Integral (PI) calcula y envía los comandos de control al MAV para aterrizar, utilizando Robot Operating System (ROS) como interfaz. Se utilizó un dron Tello para probar el sistema, logrando una

tasa de aterrizajes exitosos del 90% con una precisión de detección del 97%, demostrando que YOLOv8 es un componente clave.

Palabras Clave: Autónomo, Aterrizaje, MAV, MLP, YOLOv8.

Abstract

Autonomous navigation of micro aerial vehicles (MAVs) using visual sensors has gained relevance, particularly during the landing phase, where traditional positioning sensors may fail. The use of deep learning and computer vision helps overcome these limitations, as cameras provide more environmental information, especially when landing on moving platforms. In this work, the You Only Look Once version 8 (YOLOv8) detector is used to identify a Quick Response (QR) landing marker. A Multilayer Perceptron (MLP) is trained to estimate the relative height between the MAV and the moving platform. A Proportional-Integral (PI) controller calculates and sends control commands to the MAV for landing, using Robot Operating System (ROS) as an interface. A Tello drone was used to test the system, achieving a 90% *successful landing rate with a detection accuracy of* 97%*, demonstrating that YOLOv8 is a key component.*

Keywords: *Autonomous, Landing, MAV, MLP, YOLOv8.*

1. Introducción

En los últimos años, los vehículos aéreos no tripulados (UAVs), comúnmente conocidos como drones, han visto un aumento significativo en diversas aplicaciones, que van desde la agricultura y entrega de paquetería hasta el apoyo en misiones de rescate. Los micro vehículos aéreos (MAVs), una subdivisión de los UAVs, han emergido como una alternativa eficaz para realizar tareas en espacios confinados o como una opción de bajo costo frente a drones de mayor tamaño. Uno de los mayores desafíos durante el vuelo de un dron es la etapa de aterrizaje (80% de los accidentes de UAVs ocurren en esta fase [Yu, 2018]), especialmente en entornos dinámicos donde los sensores convencionales, como el Sistema de Posicionamiento Global (GPS), son inestables o están ausentes debido a restricciones como la capacidad de carga. Actualmente, la navegación y aterrizaje

autónomo de drones de aborda empleando sensores tradicionales como GPS, giroscopio, acelerómetro y sistemas de navegación inercial (INS), sin embargo, tienen restricciones que pueden afectar la navegación, tales como el número de satélites disponibles (GPS) o la pérdida de precisión debido al error de deriva (INS) [Lu, 2018]. Adicionalmente y enfocándose en los MAVs, éstos tienen una capacidad de carga útil muy limitada y es necesario reducir al mínimo los sensores a bordo.

Dadas las limitaciones anteriores, los sistemas de navegación visuales se han convertido en un foco de investigación, ya que los sensores visuales pueden aportar información más detallada del entorno [Lu, 2018] y no dependen de la disponibilidad de dispositivos externos, y en concreto, las cámaras monoculares son ideales para aplicaciones en las que se requiere un peso mínimo. Uno de estos enfoques ha sido el uso de marcadores visuales de aterrizaje combinados con técnicas de visión por computadora basadas en aprendizaje profundo. La idea principal es detectar el marcador de aterrizaje y obtener información sobre su posición utilizando aprendizaje profundo. Simultáneamente, estos datos se envían a un algoritmo de control para realizar el aterrizaje de forma autónoma.

En [Yu, 2018] se utilizó una red neuronal convolucional (CNN) basada en YOLO y SqueezeNet para inferir las coordenadas del centro de un marcador de aterrizaje. La CNN fue entrenada en una estación de trabajo equipada con una unidad de procesamiento de gráficos (GPU) dedicada, utilizando imágenes de varios marcadores de aterrizaje bajo diferentes condiciones de iluminación. El hardware de prueba fue un MAV DJI NAZA F450 con una computadora a bordo basada en un procesador ARM Cortex-A15. El sistema integrado usa la cámara a bordo para adquirir imágenes RGB y alimentar la CNN. Al detectar un marcador de aterrizaje de $50\,x\,50\,cm$, sus coordenadas se infieren y se convierten en desplazamientos angulares en los ejes x e y, empleando los parámetros de la cámara. Estos desplazamientos se envían al autopiloto PIXHAWK, y con la ayuda de un sonar a bordo (para obtener la altura del MAV), se calculan las posiciones en los ejes x e y del centro del marcador de aterrizaje respecto al MAV. Finalmente, el autopiloto ejecuta la rutina de aterrizaje. El modelo detector alcanzó una exactitud del 83.70% con una eficiencia de 21 cuadros por segundo (FPS), mientras que el error de

aterrizaje entre el centro del MAV y el centro del marcador fue de $8.20\ cm$ en el eje x y de $9.11\ cm$ en el eje y.

En [Nguyen, 2018] se propuso un módulo compuesto por la CNN lightDenseNet como extractor de características y YOLOv2 como detector de marcadores. YOLO retorna cuadros delimitadores que potencialmente contienen marcadores de aterrizaje, luego se selecciona aquel con el mayor puntaje de confianza. Si el cuadro delimitador seleccionado es igual o menor a $150\ pixeles$, las coordenadas de su centro se seleccionan como centro del marcador. Si el cuadro es mayor, dicha sección de la imagen se envía al algoritmo Profile Checker v2 para refinar la ubicación del centro del marcador. Profile Checker v2 emplea un umbral adaptativo y técnicas morfológicas para eliminar el ruido, luego se obtiene el centro geométrico. En seguida, se dibuja un círculo alrededor del centro predicho y se segmenta en subperfiles basados en un umbral, dividiendo el perfil en blanco y negro. Finalmente, se detectan dos puntos llamados "P" y "Q", y la dirección del marcador se estima mediante el punto medio "K" del arco entre estos puntos y el centro detectado. El rendimiento del módulo de detección fue evaluado utilizando un UAV DJI Phantom 4 con cámara a bordo y una PC con GPU NVIDIA GeForce 1070, obteniendo una precisión y recall promedio de 99%. Así mismo, la eficiencia fue medida empleando la misma PC y un kit con GPU Snapdragon 835 a bordo, consiguiendo tasas de 40 y $20\ FPS$, respectivamente.

En [Lin, 2021] se empleó un enfoque jerárquico para la detección de un marcador de aterrizaje en forma de “H” de $100x100\ cm$ en condiciones nocturnas. En la primera etapa, el detector utiliza un filtro gaussiano para reducir el ruido y mejorar la imagen, seguido de la aplicación de un umbral adaptativo para binarizar la imagen y capturar los bordes del marcador. Posteriormente, se realiza un análisis de componentes conectados para seleccionar las regiones de interés (ROIs) tanto en la imagen mejorada como en la binarizada. En la segunda etapa, ambas imágenes se procesan en un árbol de decisión que valida las ROIs utilizando cuatro nodos basados en la forma, proporciones de píxeles, componentes gráficos y su disposición. En la tercera etapa, las ROIs se envían a una red neuronal convolucional (CNN) para confirmar si contienen un marcador de aterrizaje. Si es

positivo, se extraen las coordenadas de los vértices y el centro de la "H". El rendimiento del detector fue evaluado utilizando un UAV DJI M100 conectado a una PC con GPU Nvidia Geforce 1070, a través del framework Robot Operating System (ROS), obteniendo una precisión, recall y F-measure de 97, 94 y 96%, respectivamente. Además, la eficiencia fue medida utilizando dicha PC y una tarjeta Nvidia TX2 a bordo, alcanzando tasas de hasta 40 y 12 FPS, respectivamente.

En [Cabrera-Ponce, 2020] se diseñó un sistema basado en el detector Single Shot Detector (SSD) de siete capas para detectar un marcador en forma de "H" y obtener un cuadro delimitador cuyas coordenadas son utilizadas por el controlador de aterrizaje. El SSD fue entrenado con 5000 imágenes RGB de $320\ x\ 240\ píxeles$ de la plataforma de aterrizaje, capturadas desde múltiples vistas, con rotaciones, escalas y variaciones de iluminación. Para el entrenamiento, se utilizó una PC equipada con una GPU NVIDIA GTX 960M. El sistema se evaluó tanto de manera externa (usando la misma PC) como con una computadora a bordo (Intel Stick M3 sin GPU). Las pruebas se realizaron con un MAV Bebop 2, utilizando el framework ROS para establecer la comunicación entre el MAV y la computadora, así como para calcular y enviar las señales de control. En las pruebas con la PC externa, el sistema alcanzó una confianza de detección promedio del 98.43% y una eficiencia de $90\ FPS$, mientras que, con la computadora a bordo, obtuvo una confianza promedio del 98.26% con una eficiencia de $13\ FPS$.

En [Wu, 2022] se abordó el aterrizaje autónomo en una plataforma móvil mediante el diseño de un controlador Proporcional Integral Derivativo (PID) asistido por aprendizaje profundo por refuerzo (Deep Deterministic Policy Gradient - DDPG), con retroalimentación correctiva basada en heurísticas para ajuste fino. Basándose en experiencias previas de aterrizaje manual de drones, se proporcionaron heurísticas al controlador para acelerar el proceso de ajuste realizado por el algoritmo de aprendizaje por refuerzo (RL). Para estimar la posición relativa del MAV respecto a la plataforma de aterrizaje, utilizaron un círculo como marcador de aterrizaje. Para detectar dicha marca, desarrollaron un sistema que convierte las imágenes capturadas por la cámara al modelo de color Matiz, Saturación y Valor (HSV), eliminando todos los colores excepto el azul. Luego, la máscara HSV transforma la

imagen a escala de grises, y a partir de esta imagen se obtiene una versión binarizada mediante segmentación por umbral. De esta manera, se identifica el círculo y se estiman las coordenadas del centro y el diámetro, y combinando este diámetro con la distancia focal de la cámara, se calcula la altitud del UAV. Para validar su enfoque, se utilizó el entorno virtual Gazebo y ROS. La velocidad de la plataforma de aterrizaje se estableció en $0.1\,m/s$ y la del MAV en 0.1 y $0.2\,m/s$, obteniendo una tasa de aterrizajes exitosos del 97 y 93%, respectivamente.

Dadas las limitantes de carga en un MAV y la tendencia a emplear sensores visuales apoyados por aprendizaje profundo en esta área, esta investigación presenta un sistema de aterrizaje autónomo basado en el detector de objetos YOLOv8 para identificar un marcador de aterrizaje, específicamente un código QR. Las salidas del detector se utilizan para determinar la posición de un MAV Tello en relación con el marcador y alimentar un Perceptrón Multicapa (MLP) para estimar la altura, también en relación con el marcador.

Con la posición determinada en todos los ejes, se emplea un controlador PI para aterrizar de manera segura el MAV sobre el marcador, ubicado en una plataforma en movimiento. Si bien otros equipos han empleado versiones anteriores de YOLO para obtener las coordenadas del marcador, en este trabajo se explotan las características nativas de la versión 8 del detector, evitando realizar posprocesamientos. De igual forma, al usar un MLP para estimar la altura, el sistema puede adaptarse rápidamente a otras cámaras a bordo, pues no se necesita conocer los parámetros intrínsecos de éstas. Finalmente, los módulos se han programado empleando ROS, permitiendo así adaptarlos a otros drones con modificaciones mínimas en las señales de control.

2. Métodos

Con el objetivo de implementar un sistema de aterrizaje autónomo para un MAV en una plataforma en movimiento, en este trabajo se desarrolla un método que aprovecha las características nativas de YOLOv8 y la modularidad de ROS. Se presentan en orden los submódulos a implementar y su función:

- Módulo 1 – driver Tello: Establece la conexión MAV-PC(ROS).

- Módulo 2 - interfaz de imágenes: Adquiere las imágenes del MAV y asegura la compatibilidad de éstas entre ROS y OpenCV (Python). Se integra en los demás módulos.
- Módulo 3 - control manual: Empleado para posicionar el MAV sobre el marcador y controlarlo en caso de emergencia, usando el teclado.
- Módulo 4 - detección de marcador y estimación de altura: Emplea YOLOv8 para identificar el marcador de aterrizaje QR y un MLP para estimar la altura del MAV respecto a este.
- Módulo 5 - aterrizaje: Toma las salidas del módulo 4 para calcular la posición del MAV respecto al marcador y realizar el aterrizaje usando un controlador Proporcional-Integral (PI).

En seguida, se describe el hardware, software y otros recursos empleados, junto con una breve explicación de la razón de su elección, complementada en las subsecciones correspondientes:

- MAV DJI Tello: MAV disponible para pruebas en la institución.
- Laptop Lenovo con procesador Ryzen 7, GPU integrada Radeon Graphics, Ubuntu 20.04.5: Equipo compatible con el MAV Tello.
- Código QR de $10x10\ cm$ como marcador de aterrizaje, con la palabra ‘10x10’: Se estableció este tamaño como adecuado, considerando la navegación en espacios reducidos.
- YOLOv8 preentrenado como detector de códigos QR: Elegido por sus características nativas de salida (coordenadas de esquinas y centro del cuadro delimitador, así como segmentación del código QR).
- MLP como estimador de altura: Elegido por su capacidad de obtener un modelo adaptable e independiente de las características de la cámara a bordo.
- Controlador PI para la rutina de aterrizaje: Seleccionado por su simplicidad y robustez.
- Controlador Tello como interfaz entre el MAV y ROS: Proporciona funciones que simplifican el control del MAV.

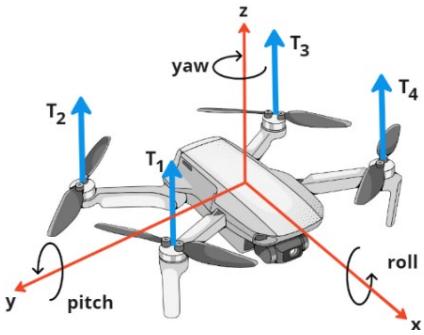
- ROS Noetic como framework para la implementación de los algoritmos y control del MAV: Su enfoque modular facilita la integración de los diferentes algoritmos desarrollados.

Micro Vehículos Aéreos y configuración del MAV Tello

Los vehículos aéreos no tripulados (UAVs) son una división de aeronaves capaces de volar sin un piloto humano a bordo. El UAV típico consta de la estructura de la aeronave, la carga útil de sensores y una estación de control en tierra [Narayanan, 2015]. Uno de los tipos más populares de UAVs son los micro vehículos aéreos (MAVs), que se definen como aeronaves cuyas dimensiones son iguales o menores a $15\ cm$ [Hall, 2013]. Debido a su tamaño y limitaciones de carga útil, los MAVs suelen ser controlados desde estaciones de control en tierra, y a menudo están equipados con una cámara, lo que los hace ideales para tareas de monitoreo en espacios reducidos. Uno de los tipos más utilizados de MAVs es el cuadricóptero, que consta de cuatro actuadores controlados individualmente para generar empuje relativo [Idrissi, 2022].

En este trabajo se emplea un MAV Tello para evaluar el sistema desarrollado. Este MAV es un cuadricóptero con dimensiones de $9.8\ x\ 9.25\ x\ 4.1\ cm$. Incluye una cámara de $5\ Megapixeles$ con un campo de visión de $82.6°$ y una resolución de video de $720\ p$. Es capaz de volar hasta una altura de $30\ m$, con una velocidad máxima de $8\ m/s$ y un tiempo de vuelo máximo de $13\ min$ [Bhujbal, 2022]. En la Figura 1a, se muestra el sistema de coordenadas local del MAV (el marco de referencia del Tello), donde el eje x apunta hacia adelante en la dirección de la nariz del dron, el eje y apunta hacia la derecha, y el eje z hacia arriba.

El MAV Tello incluye una cámara orientada hacia adelante; sin embargo, la cámara debe estar orientada hacia abajo para detectar el marcador y realizar el aterrizaje. Para lograr esto, se colocó un espejo de $2\ x\ 2.5\ cm$ en un ángulo de $45°$ al frente del dron, como se muestra en la Figura 1b. Como resultado de añadir el espejo, la imagen se invierte verticalmente (inversión a lo largo del eje y del sistema de coordenadas de la cámara). Esta inversión se tendrá en cuenta en el desarrollo del algoritmo de control de posición.



a) Marco de referencia del MAV Tello.



b) MAV Tello con espejo orientado hacia abajo.

Fuente: elaboración propia

Figura 1 Marco de referencia y modificación al MAV Tello.

Framework ROS y configuración del ambiente de desarrollo

Robot Operating System (ROS) es un conjunto de frameworks y bibliotecas de código abierto utilizados para desarrollar software para la programación de robots. Uno de sus objetivos es facilitar este proceso mediante una arquitectura modular, la reutilización de software y la interoperabilidad entre todos los elementos del entorno ROS [Bernardeschi, 2018]. ROS se basa en nodos que representan los elementos del robot y pueden establecer comunicación empleando los métodos de suscripción y publicación. Los nodos se dividen en publicadores y suscriptores, cuyas funciones son enviar y recibir información, respectivamente. Pueden enviar mensajes (información) a través de los tópicos (ruta o bus) [Joseph, 2018].

Con el fin de aprovechar las ventajas que ofrece ROS y previo a programar los algoritmos de aterrizaje, se configuró el entorno de desarrollo considerando los requerimientos de la versión Noetic de ROS, instalando Ubuntu 20.04, Python 3.8, y finalmente ROS y sus dependencias, junto con el driver Tello-Python proporcionado por [Balbuena-Palma, 2024]. La guía de configuración del ambiente, así como los scripts (módulos) desarrollados y los conjuntos de datos generados se encuentran disponibles en GitHub [Matías-Pacheco, 2024].

Módulo 1 – Driver Tello

Conformado por el driver Tello, permite establecer la conexión entre el MAV y los módulos de ROS ejecutados en la laptop. Al ejecutarlo se identificó la principal limitante en el desarrollo de los algoritmos de detección y control. El desarrollo del

sistema se inició en una laptop Acer Nitro 5 con GPU NVIDIA 3060. Sin embargo, debido a problemas de compatibilidad entre los módulos Wi-Fi de la laptop y el MAV Tello, y la falta de acceso a otro equipo con GPU dedicada, se optó por utilizar una laptop Lenovo sin GPU dedicada, instalando en este equipo el ambiente de desarrollo. Después de establecer la conexión Tello-ROS, se ejecutó el comando rqt_image_view (de la biblioteca rqt, ya integrada en el conjunto de herramientas de ROS) para verificar que la transmisión de video funcionara correctamente.

Módulo 2 - interfaz de imágenes

Para alimentar YOLO v8, es necesario trabajar directamente con los fotogramas recibidos por ROS. ROS maneja imágenes en su propio formato de mensaje (sensor_msgs/Image), por lo tanto, para procesar los fotogramas, es necesario crear una interfaz entre ROS y OpenCV, convirtiendo las imágenes de ROS en imágenes de OpenCV utilizando cv_bridge, una biblioteca incluida en el framework de ROS. Para implementar esta función, el script crea el nodo image_converter y se suscribe al tópico /tello/image_raw. Luego, cv_bridge se usa para convertir la imagen, y finalmente, la secuencia de video se muestra en una ventana. Esta sección de código se integrará más adelante en los scripts del detector de marcadores y del aterrizador.

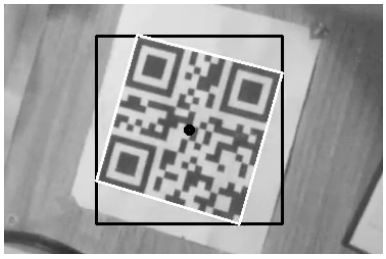
Módulo 3 – control manual

Se creó un script para el control manual del MAV utilizando el teclado. Este permite el despegue, aterrizaje, movimiento y cambio entre los modos autónomo y manual. En las pruebas del sistema, este script se utilizará para volar el MAV sobre el marcador de aterrizaje y comenzar la rutina de aterrizaje autónomo. El script publica mensajes en los tópicos /tello/takeoff, /tello/land, /tello/cmd_vel (controla el movimiento del MAV a través de sus componentes lineales x, y, z y el angular z) y /keyboard/override (este tópico permite cambiar del modo manual al modo autónomo). Después de escribir el script de control por teclado, se probó en conjunto con el módulo de interfaz de imágenes, con el fin de verificar el correcto control integrado de movimiento y la adquisición de imágenes del MAV.

Módulo 4 – detección de marcador (4a) y estimación de altura (4b)

Con el fin de detectar el código QR utilizado como marcador de aterrizaje, se empleó QRDet, un modelo de detector de códigos QR preentrenado basado en YOLOv8 y que puede obtenerse en [Cañas, 2023]. YOLO (*You Only Look Once*) es un sistema de detección de objetos de una sola etapa propuesto en 2015 por J. Redmon [Xiao, 2020]. Este algoritmo divide la imagen de entrada en una cuadrícula y, en una sola evaluación, predice los cuadros delimitadores y las probabilidades de clase para cada celda [Adarsh, 2020].

En esta investigación, se utiliza YOLOv8. Esta variante fue introducida por Ultralytics en 2023, manteniendo una arquitectura base similar a las versiones anteriores, pero con mejoras en el módulo *Cross-Stage Partial* (CSP) Bottleneck con 2 convoluciones (C2f). Este módulo modificado combina características de alto nivel con información contextual, mejorando la precisión de detección y centrándose en optimizar la velocidad de ejecución sin sacrificar el rendimiento. YOLOv8 también incluye un módulo de segmentación semántica llamado YOLOv8-seg, que consiste en un extractor de características CSPDarknet53 seguido de un módulo C2f [Hussain, 2024]. El modelo QRDet basado en YOLOv8 forma un submódulo (4a) que toma una imagen de entrada (empleando los módulos 1 y 2) y devuelve la confianza de detección, las coordenadas en píxeles de las esquinas del cuadro delimitador, las coordenadas en píxeles del centro del cuadro delimitador y las coordenadas en píxeles del polígono de cuatro esquinas que segmenta el QR. En la Figura 2 se puede observar el código QR empleado como marcador, siendo segmentado y rodeado por un cuadro delimitador. También se muestra el centro del código QR. Para estimar la altura entre el dron y el marcador de aterrizaje, se implementó un submódulo (4b) integrado por un Perceptrón Multicapa (MLP). Este enfoque permite obtener una estimación sin necesidad de calibración de la cámara o conocimiento de parámetros como la distancia focal. Un MLP está compuesto por las siguientes características: una capa de entrada o vector de características x, un número arbitrario de capas ocultas, una capa de salida, y un conjunto de pesos y bias entre cada capa, w y θ respectivamente, además de una función de activación para cada capa oculta [Loy, 2019].



Fuente: elaboración propia

Figura 2 Marcador QR detectado y segmentado con YOLOv8 QRDet.

El valor de activación para una neurona está dado por la Ecuación 1. Donde x_i es el valor de cada neurona de entrada, w_i es el valor de la conexión entre la neurona i y la salida. Finalmente b es el bias.

$$a(x) = \sum_i w_i\, x_i + b \tag{1}$$

Para entrenar el MLP, se generó un conjunto de datos volando el MAV Tello sobre el marcador QR a diferentes alturas (5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110 y $120\ cm$) y se ejecutó el submódulo detector 4b. El submódulo retorna entre otras, las salidas bb_area, xmid y y_mid. De este modo se obtuvieron los atributos de entrenamiento para para el MLP, siendo vectores de la forma (qr_size, bb_area, xmid, ymid, height), correspondientes al tamaño del QR (en cm), área del QR (en píxeles, calculada con las coordenadas de las esquinas del cuadro delimitador, también en píxeles), las coordenadas x, y (en píxeles) del centro del cuadro delimitador, así como la altura de vuelo asociada (en cm). Para cada altura se generó un archivo txt con los atributos mencionados. Posteriormente se realizó el análisis y limpieza de datos, detectando y eliminando valores atípicos. Luego, las 13 alturas diferentes se transformaron en clases (asignando etiquetas de 0 a 12) y se realizó una codificación one-hot. Además, los atributos fueron estandarizados. Utilizando Keras, se generó la siguiente arquitectura del MLP:

- Capa de entrada con 64 neuronas y función de activación ReLU.
- Segunda capa con 32 neuronas y función de activación ReLU.
- Capa de salida con 13 neuronas, utilizando la función de activación Softmax.
- Optimizador Adam.

- Función de pérdida Categorical Crossentropy.
- 1300 épocas.
- Tamaño del lote = 32.
- Tamaño de conjunto de entrenamiento = 0.8, tamaño de prueba = 0.2.

Una vez entrenado el MLP, los submódulos de detección del marcador y estimación de altura se integraron en un solo script. Este script se suscribe al tópico /tello/image_raw para obtener los fotogramas de video y genera el nodo image_converter, publicando los siguientes tópicos (Tabla 1), que serán utilizados por el módulo de aterrizaje.

Tabla 1 Tópicos publicados por el módulo 4.

Tópico	Tipo de Mensaje	Función
/qr_detection/center	Int32MultiArray	Publica las coordenadas en pixeles del centro (x, y) del cuadro delimitador del marcador QR detectado.
/qr_detection/quad_points	Int32MultiArray	Publica las coordenadas en pixeles de los puntos que forman el borde inferior del cuadrilátero segmentado del marcador QR.
/qr_detection/predicted_height	Int32	Publica la altura estimada (en cm) del MAV en relación con el marcador QR.

Fuente: elaboración propia

Módulo 5 - aterrizaje

Este módulo recibe los datos publicados por el módulo 4 y los utiliza para calcular los errores de posición en los ejes x y y (medidos en píxeles), el error de posición en el eje z (medido en centímetros), y el error de orientación en el eje yaw (medido en grados). Luego, el controlador PI calcula las señales de control para el dron y las publica en los tópicos correspondientes de ROS.

Para obtener la señal de control para los ejes x, y, z y yaw, se sigue la Ecuación 2 y su versión en su forma discreta, la Ecuación 3 [Wang, 2000].

$$U_i = k_{pi} * e_i + k_{ii} * \int e_i * dt \tag{2}$$

$$U_i = k_{pi} * e_i + k_{ii} * \sum_{i=1}^{n_i} e_i \tag{3}$$

Donde U_i es la señal de control para x, y, z y yaw, k_{pi} son las ganancias proporcionales, k_{ii} las ganancias integrales, $\int e_i * dt$ es el error acumulado (integral) para el eje i. El error integral se aproxima sumando el error en cada paso de tiempo discreto: $\sum_{i=1}^{n_i} e_i$. Para calcular los errores de posición en los ejes x, y y z, el módulo 5 se suscribe a los tópicos /qr_detection/center y /qr_detection/predicted_height (Tabla 1), Ecuaciones 4, 5 y 6, respectivamente.

$$e_x = x_q - x_c \tag{4}$$

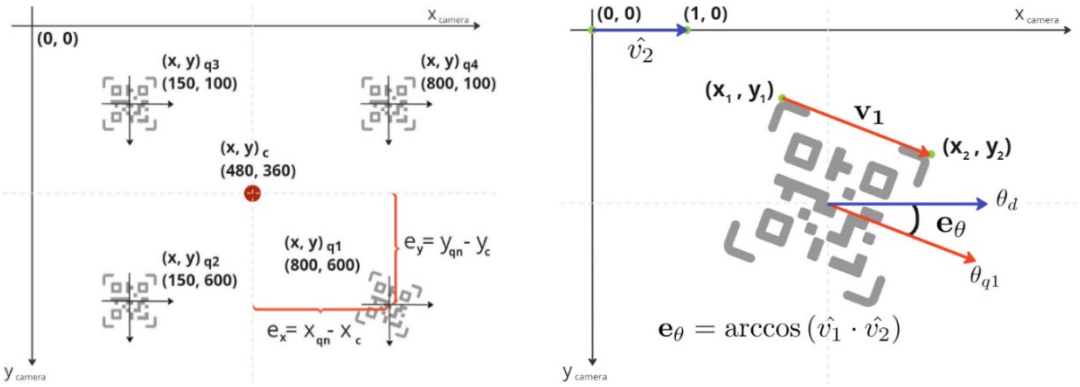
$$e_y = y_q - y_c \tag{5}$$

$$e_z = 0 - height \tag{6}$$

Donde x_q y y_q son las coordenadas en píxeles del centro del marcador de aterrizaje, x_c y y_c son las coordenadas en píxeles de la posición deseada (el centro del plano coordenado), 0 es la posición deseada en el eje z (plataforma de aterrizaje) y $height$ es la altura en centímetros predicha por el MLP.

En la Figura 3a, se presenta el plano coordenado observado desde la perspectiva del MAV. Es importante señalar que el plano está invertido a lo largo del eje y para ajustarse a la imagen invertida de la cámara, causada por el espejo. Además, en la Figura 4 se muestra una representación del sistema de aterrizaje autónomo: el MAV vuela sobre el marcador de aterrizaje QR, mostrando el plano coordenado real (sin la inversión de la imagen causada por el espejo), junto con las posiciones del MAV y del marcador QR, y los errores de posición.

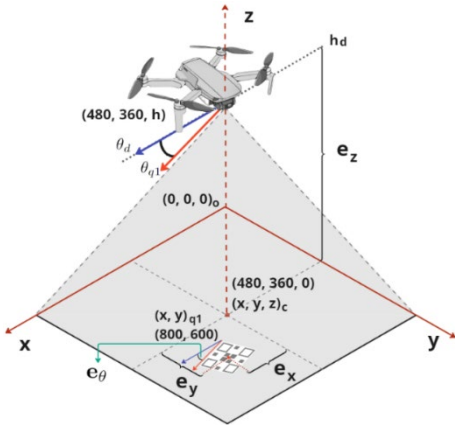
.



a) Plano coordenado desde la perspectiva del MAV. *b) Vectores para el cálculo del error angular.*

Fuente: elaboración propia

Figura 3 Esquemas del plano coordenado para los cálculos de posiciones y errores.



Fuente: elaboración propia

Figura 4 Esquema del sistema de aterrizaje en el plano coordenado real.

Por otro lado, para calcular el error en yaw, el módulo 5 se suscribe al tópico /qr_detection/quad_points, que contiene las coordenadas (en píxeles) de los puntos que forman el borde inferior del cuadrilátero segmentado del QR. Estas coordenadas se convierten en un vector $\overrightarrow{v_1}$ y se transforman en un vector unitario Adicionalmente, se crea un vector unitario horizontal v_1 como referencia (que representa el ángulo del dron) para calcular el error en el ángulo yaw (e_θ). La Figura 3b muestra una representación del marcador QR tal como lo ve la cámara (imagen y plano coordenado invertidos en el eje y), con los vectores utilizados para calcular el error angular en yaw. La Figura 4 también ilustra este proceso, pero mostrando el plano coordenado real.

Sean (x_1, y_1), (x_2, y_2) las coordenadas del borde inferior del QR, el vector formado por estos puntos es la Ecuación 7. El vector unitario $\overrightarrow{v_1}$, y el vector unitario $\hat{v}_2$ están dados por las Ecuaciones 8 y 9.

$$\overrightarrow{v_1} = (x_2 - x_1, y_2 - y_1) \tag{7}$$

$$v_1 = \frac{\overrightarrow{v_1}}{||\overrightarrow{v_1}||} = \frac{(x_2 - x_1, y_2 - y_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \tag{8}$$

$$\hat{v}_2 = (1,0) \tag{9}$$

El producto punto entre estos vectores se calcula con las Ecuaciones 10 y 11. Dado que los vectores son unitarios, el ángulo se determina tomando el arcoseno del producto punto resultante (Ecuación 12).

$$\hat{v}_1 \cdot \hat{v}_2 = \frac{(x_2 - x_1, y_2 - y_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \cdot (1,0) = || \overrightarrow{v_1} ||\ || \overrightarrow{v_2} || \cos(\theta) \quad (10)$$

$$\frac{x_2 - x_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} = \cos(\theta) \quad (11)$$

$$\theta = arcos\left(\frac{x_2 - x_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}\right) \quad (12)$$

El ángulo obtenido se convierte de radianes a grados. Finalmente, se realiza un ajuste de signo para determinar el signo del error. Así, el error en yaw (e_θ) se expresa en la Ecuación 13.

$$e_\theta = -e_\theta \quad si \quad y_4 > y_3, \qquad e_\theta = e_\theta \quad en\ otro\ caso \quad (13)$$

Finalmente, las ganancias se establecieron de manera heurística y las señales de control de salida se calcularon de acuerdo con la Ecuación 3, luego se publican en los tópicos correspondientes. Es importante señalar que el marco de referencia del dron (ejes x, y) no corresponde al plano coordenado del cámara definido anteriormente, ya que están invertidos. Por lo tanto, la señal de control U_x (movimiento derecha-izquierda) se envía en el componente linear.y del mensaje Twist publicado en el tópico /tello/cmd_vel, y la señal de control U_y (movimiento adelante-atrás) se envía en el componente linear.x. Las señales de control para los ejes z y yaw se envían en linear.z y angular.z, respectivamente.

Sistema integrado y pruebas

Una vez que todos los módulos del sistema se ejecutaron y evaluaron individualmente, y se corrigieron errores (ecuaciones incorrectas, ganancias altas, fallas del MAV, etc.), se realizó la prueba de integración del sistema de la siguiente manera:

- Submódulo 4a y módulo 5: configuración probada para verificar la detección del marcador de aterrizaje en vuelo y el controlador de posición PI sobre los ejes x, y. Esta prueba se dividió en dos partes: posicionamiento del dron sobre el centro del marcador de aterrizaje fijo y posicionamiento del dron sobre el centro del marcador de aterrizaje en movimiento.

- Módulos 4 y 5: configuración probada para verificar el correcto aterrizaje autónomo del dron sobre el marcador QR. Esta prueba también se dividió en dos partes: aterrizaje sobre un marcador fijo y aterrizaje sobre un marcador en una plataforma en movimiento.

Usando el modo manual del módulo 3, el dron despega y se posiciona sobre el marcador de aterrizaje QR a $1.4\ m$ de altura. Cuando una porción del QR es visible en la cámara, se activa el modo autónomo y el sistema de aterrizaje autónomo comienza a ejecutarse. En la Figura 5, se muestran los módulos 4 y 5 ejecutándose, ilustrando la detección del marcador, el cálculo del error de posición, las señales de control enviadas al dron y la transmisión de video.



Fuente: elaboración propia

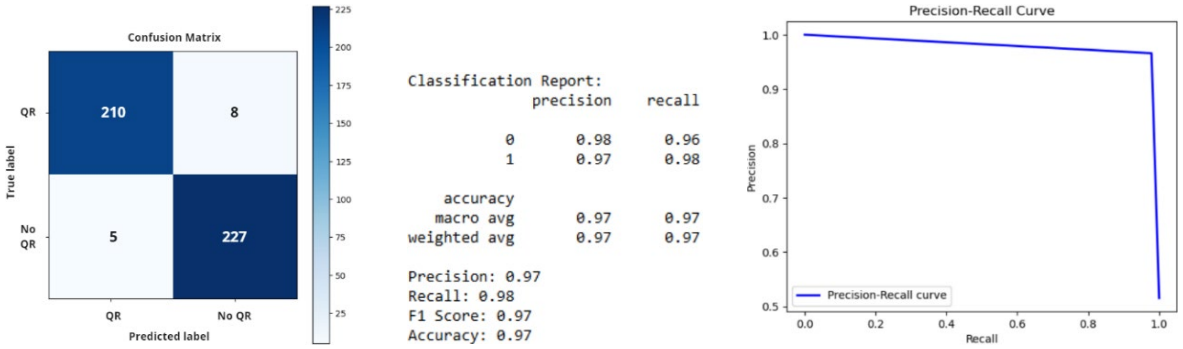
Figura 5 Sistema de aterrizaje autónomo integrado ejecutándose.

3. Resultados

YOLOv8 QRDet mostró un alto rendimiento en la identificación del marcador QR bajo diferentes condiciones de luz y ángulos de cámara. Es importante destacar que fue capaz de detectar códigos QR incompletos, lo cual es una característica importante, ya que el módulo 5 puede activar el controlador de posición antes de que el marcador aparezca completamente en el campo de visión de la cámara. Antes de probar el detector con el MAV en vuelo, se evaluó con un conjunto de imágenes que contenían códigos QR y texturas aleatorias de superficies. Posteriormente, se generó la matriz de confusión y se calcularon las métricas para las detecciones de QR, donde la clase '0' corresponde a 'No QR' y la clase '1' a 'QR' (Figuras 6ª y 6b). Finalmente, se trazó una curva de Precisión-Recall (Figura 6c), mostrando que el detector de QR minimiza tanto los falsos positivos como los falsos

negativos. El estimador MLP fue evaluado con el subconjunto de prueba de la base de datos

generada obteniendo una precisión promedio del 92% y un recall del 89% en la estimación, dependiendo de la altura del MAV (Tabla 2).



a) Matriz de confusión. *b) Métricas de YOLOv8 QRDet.* *c) Curva precisión-recall.*

Fuente: elaboración propia

Figura 6 Resultados obtenidos al evaluar YOLOv8 QRDet.

Tabla 2 Métricas globales y por altura obtenidas por el MLP.

Altura estimada ($height$)	**Precisión**	**Recall**
5	*1.0*	*1.0*
10	*1.0*	*1.0*
20	*1.0*	*1.0*
30	*1.0*	*1.0*
40	*1.0*	*1.0*
50	*0.94*	*0.94*
60	*0.89*	*0.73*
70	*0.90*	*0.97*
80	*0.88*	*0.88*
90	*0.89*	*0.80*
100	*0.69*	*0.82*
110	*1.0*	*0.67*
120	*0.80*	*0.80*
Promedio	*0.92*	*0.89*

Fuente: elaboración propia

La prueba del sistema integrado en el dron Tello se realizó en un entorno controlado. Se llevaron a cabo 20 $vuelos$, obteniéndose los resultados mostradps en la Tabla 3. En la Figura 7a se presenta un aterrizaje con la plataforma moviéndose en un trayecto plano y lineal, mientras que en la Figura 7b se realiza una prueba en un trayecto con una pendiente en la sección inicial, demostrando la capacidad del detector QR para detectar un marcador en diferentes ángulos.

Tabla 3 Resultados de las pruebas de aterrizaje del sistema integrado.

Número de prueba	Distancia Recorrida (m)	Error de aterrizaje (e_x, e_y) (cm)	Tiempo para aterrizar (s)	FPS	Aterrizaje
1	*1.2*	*(1, 1)*	*9*	*4*	Exitoso
2	*1.5*	*(2, 0)*	*10*	*4*	Exitoso
3	*2.0*	*(3, 2)*	*12*	*4*	Exitoso
4	*2.2*	*(1, 1)*	*13*	*4*	Exitoso
5	*1.8*	*(4, 3)*	*11*	*4*	Exitoso
6	-	*(-, -)*	-	*4*	Falla
7	*2.1*	*(1, 1)*	*13*	*4*	Exitoso
8	*1.3*	*(0, 2)*	*9*	*4*	Exitoso
9	*1.7*	*(1, 1)*	*11*	*4*	Exitoso
10	*1.9*	*(5, 4)*	*12*	*4*	Exitoso
11	*2.4*	*(1, 0)*	*15*	*4*	Exitoso
12	*1.4*	*(2, 1)*	*9*	*4*	Exitoso
13	*2.3*	*(1, 5)*	*14*	*4*	Exitoso
14	*1.6*	*(0, 0)*	*10*	*4*	Exitoso
15	*2.0*	*(1, 1)*	*12*	*4*	Exitoso
16	*2.5*	*(3, 2)*	*16*	*4*	Exitoso
17	*1.1*	*(1, 1)*	*8*	*4*	Exitoso
18	-	*(-, -)*	-	*4*	Falla
19	*2.3*	*(1, 1)*	*15*	*4*	Exitoso
20	*2.5*	*(0, 0)*	*16*	*4*	Exitoso
Promedio	*1.88*	*(1.56, 1.44)*	*11.94*	*4*	-

Fuente: elaboración propia



a) Aterrizaje en una pista plana. *b) Aterrizaje en una pista irregular.*

Fuente: elaboración propia

Figura 7 Prueba de aterrizaje del sistema integrado.

4. Discusión

El rendimiento del detector QR se evaluó en términos de precisión y recall siguiendo las definiciones de [Buckland, 1994]:

- Precisión: Durante el aterrizaje, todas las detecciones deben ser verdaderos positivos, ya que un falso positivo podría llevar al controlador PI a mover el MAV a una posición incorrecta o retrasar el proceso de aterrizaje. Una

precisión de 0.97 indica que es poco probable detectar falsos positivos, lo que asegura un aterrizaje rápido y seguro.

- Recall: El marcador QR debe detectarse continuamente durante todo el aterrizaje. Un falso negativo podría detener la rutina de aterrizaje, especialmente en una plataforma en movimiento. Un recall de 0.98 garantiza que es improbable perder el marcador, logrando un seguimiento y aterrizaje exitoso.

En cuanto al estimador de altura, el modelo tiende a estimar la altura correctamente cuando el dron está más cerca del marcador QR, y falla más cuando está volando a mayor altura. Esto se debe a la diferencia en el tamaño del cuadro delimitador (utilizado por el MLP); cuanto más alto vuela el MAV, menor es la diferencia en el área del cuadro delimitador para cada predicción de altura.

Como se muestra en la Tabla 2, el sistema integrado tiene un 90% de éxito en los aterrizajes, con un error promedio de $1.5\ cm$ en el eje x y $1.4\ cm$ en el eje y. El tiempo promedio de aterrizaje fue de $11.94\ s$, recorriendo una distancia de $1.88\ m$, lo que da una velocidad promedio de $0.15\ m/s$. El sistema realiza 4 evaluaciones por segundo ($4\ FPS$) al ejecutar todos los módulos, pero alcanza hasta $21\ FPS$ cuando solo se ejecuta el módulo 4. Es importante recalcar que el tiempo de aterrizaje está limitado por la falta de una GPU dedicada en la computadora utilizada para el procesamiento, así como la latencia en la conexión inalámbrica.

Si bien no existe una métrica común, el sistema desarrollado se comparó respecto a las metodologías existentes (Tabla 4). Es así como el método propuesto presenta una precisión competitiva frente a otros algoritmos, notando que es superado o igualado por enfoques basados en dos o más algoritmos unidos, a excepción del SSD. Esto debido posiblemente a que YOLOv8 QRDet se evaluó con un conjunto de datos que contenían a priori imágenes similares a QRs, tales como marcadores ARUCO. Si bien la eficiencia es inferior en términos de FPS, esto se debe a las limitaciones del hardware. Pese a esta desventaja, el sistema ha sido capaz de realizar aterrizajes precisos en una plataforma en movimiento, con un error de posición de aterrizaje menor a $2\ cm$, lo que demuestra su fiabilidad en entornos

dinámicos, presentando un rendimiento sólido y con el potencial de mejorar su eficiencia con hardware más avanzado.

Tabla 4 Comparación de metodologías con distintas métricas.

Método	**Precisión** (%)	**Tasa de éxito** (%)	**Error de Aterrizaje** (e_x, e_y) (cm)	FPS **detector (PC – A bordo)**
YOLO + SqueezeNet	*83.7*	*N/A*	*(8.2, 9.11)*	*(N/A - 21)*
lightDenseNet + YOLOv2 + Profile Checker v2	*99*	*N/A*	*N/A*	*(40 – 20)*
CNN + Árbol de decisión	*97*	*N/A*	*N/A*	*(40 - 12)*
SSD	*98*	*N/A*	*N/A*	*(90 – 13)*
PID + DDPG	*N/A*	*97*	*N/A*	*N/A*
YOLOv8 + MLP	*97*	*90*	*(1.5, 1.4)*	*(21, N/A)*

Fuente: elaboración propia

5. Conclusiones

Esta investigación desarrolló un sistema de aterrizaje autónomo para MAVs, capaz de aterrizar de manera precisa y segura sobre una plataforma en movimiento, cumpliendo el objetivo propuesto. Se implementó un sistema basado en YOLOv8, utilizando la cámara del dron Tello como único sensor, evitando así una mayor carga útil. El detector de código QR preentrenado basado en YOLOv8 mejoró notablemente la precisión de la detección del marcador de aterrizaje, demostrando ser tan o más robusto que otros sistemas recientes. Esta precisión fue clave para la rutina de aterrizaje autónomo, garantizando que mientras el marcador QR estuviera dentro del campo de visión del dron, su posición no se perdería, lo que permitió un seguimiento preciso del marcador. Además, las características nativas de YOLOv8 se utilizaron para entrenar un estimador de altura basado en un MLP. La salida de este estimador, junto con las características de YOLOv8, alimentaron un controlador PI, lo que permitió un aterrizaje exitoso sobre la plataforma móvil.

Las pruebas realizadas con el MAV Tello y los módulos integrados demostraron que el sistema es capaz de detectar de manera continua el marcador QR, estimar la altura, seguir la plataforma en movimiento y aterrizar con alta precisión y en un tiempo aceptable. Por lo tanto, este sistema presenta una solución viable para aterrizajes autónomos en entornos donde no se dispone de GPS u otros sensores tradicionales. Finalmente, se recomienda investigar cómo mejorar la estimación de

altura y la velocidad de aterrizaje, explorando algoritmos más avanzados y utilizando hardware con GPU dedicada o un sistema a bordo más potente.

6. Bibliografía y Referencias

[1] Adarsh, P., Rathi, P., Kumar, M. YOLO v3-Tiny: Object Detection and Recognition Using One-Stage Improved Model. 6th International Conference on Advanced Computing and Communication Systems (ICACCS), 687-694. 2020.

[2] Balbuena-Palma, J. A. ROS-Tello Driver. https://github.com /JoseBalbuena181096/catkin_ws_ROS_TELLO. 2024.

[3] Buckland, M., Gey, F. The Relationship Between Recall and Precision. Journal of the American Society for Information Science, vol. 45, no. 1, pp. 12-19, Wiley Online Library. 1994.

[4] Bernardeschi, C., Fagiolini, A., Palmieri, M., Scrima, G., Sofia, F. ROS/Gazebo-Based Simulation of Cooperative UAVs. International Conference on Modelling and Simulation for Autonomous Systems, 321-334, Springer. 2018.

[5] Bhujbal, K., Barahate, S. Custom Object Detection Based on Regional Convolutional Neural Network & YOLOv3 With DJI Tello Programmable Drone. 7th International Conference on Innovation & Research in Technology & Engineering (ICIRTE). 2022.

[6] Cañas, E. QRDet. https://github.com/Eric-Canas/qrdet. Septiembre 2023.

[7] Cabrera-Ponce, A. A., Martínez-Carranza, J. Onboard CNN-Based Processing for Target Detection and Autonomous Landing for MAVs. Mexican Conference on Pattern Recognition, 195-208, Springer. 2020.

[8] Hall, J., Mohseni, K. Micro Aerial Vehicles. Encyclopedia of Microfluidics and Nanofluidics, 1-10, Springer US, Boston, MA. 2013.

[9] Hussain, M. YOLOv1 to v8: Unveiling Each Variant - A Comprehensive Review of YOLO. IEEE Access, vol. 12, pp. 42816-42833. 2024.

[10] Idrissi, M., Salami, M., Annaz, F. A Review of Quadrotor Unmanned Aerial Vehicles: Applications, Architectural Design, and Control Algorithms. Journal of Intelligent & Robotic Systems, vol. 104, no. 2, pp. 1-33, Springer. 2022.

[11] Joseph, L. Robot Operating System for Absolute Beginners. Springer. 2018.

[12] Lin, S., Jin, L., Chen, Z. Real-Time Monocular Vision System for UAV Autonomous Landing in Outdoor Low-Illumination Environments. Sensors, vol. 21, no. 18, p. 6226, MDPI. 2021.

[13] Loy, J. Neural Network Projects with Python: The Ultimate Guide to Using Python to Explore the True Power of Neural Networks Through Six Projects. Packt Publishing Ltd. 2019.

[14] Lu, Y., Xue, Z., Xia, G.-S., Zhang, L. A Survey on Vision-Based UAV Navigation. Geo-Spatial Information Science, vol. 21, no. 1, pp. 21-32, Taylor & Francis. 2018.

[15] Matías-Pacheco, A. D. MAV-Autonomous-Landing-Tello-YOLO-ROS. https://github.com/ADanielMt/MAV-Autonomous-Landing-Tello-YOLO-ROS. Agosto 2024.

[16] Narayanan, R. G. L., Ibe, O. C. Joint Network for Disaster Relief and Search and Rescue Network Operations. Wireless Public Safety Networks 1, pp. 163-193, Elsevier. 2015.

[17] Nguyen, P. H., Arsalan, M., Koo, J. H., Naqvi, R. A., Truong, N. Q., Park, K. R. LightDenseYOLO: A Fast and Accurate Marker Tracker for Autonomous UAV Landing by Visible Light Camera Sensor on Drone. Sensors, vol. 18, no. 6, p. 1703, MDPI. 2018.

[18] Wang, Y.-G., Shao, H.-H. Optimal Tuning for PI Controller. Automatica, vol. 36, no. 1, pp. 147-152, Elsevier. 2000.

[19] Wu, L., Wang, C., Zhang, P., Wei, C. Deep Reinforcement Learning with Corrective Feedback for Autonomous UAV Landing on a Mobile Platform. Drones, vol. 6, no. 9, p. 238, MDPI. 2022.

[20] Xiao, Y., Tian, Z., Yu, J., Zhang, Y., Liu, S., Du, S., Lan, X. A Review of Object Detection Based on Deep Learning. Multimedia Tools and Applications, vol. 79, pp. 23729-23791, Springer. 2020.

[21] Yu, L., Luo, C., Yu, X., Jiang, X., Yang, E., Luo, C., Ren, P. Deep Learning for Vision-Based Micro Aerial Vehicle Autonomous Landing. International Journal of Micro Air Vehicles, vol. 10, no. 2, pp. 171-185, SAGE Publications Inc. 2018.