ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

ESTUDIO, DESARROLLO E INTEGRACIÓN DE EQUIPOS AUXILIARES PARA EL CONTROL Y/O MONITOREO DE PLATAFORMAS ROBÓTICAS

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO AUXILIAR QUE SIRVA DE APOYO PARA LA MANIOBRA DE ATERRIZAJE DE UN VEHÍCULO AÉREO NO TRIPULADO

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y AUTOMATIZACIÓN

LUIS DAVID ORTEGA TOAQUIZA

david1997ortega@hotmail.com

DIRECTOR: Ing. PATRICIO JAVIER CRUZ DÁVALOS, Ph.D.

patricio.cruz@epn.edu.ec

DMQ, octubre 2022

CERTIFICACIONES

Yo, Luis David Ortega Toaquiza declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

LUIS DAVID ORTEGA TOAQUIZA

Certifico que el presente trabajo de integración curricular fue desarrollado por Luis David Ortega Toaquiza, bajo mi supervisión.

Ing. PATRICIO JAVIER CRUZ DÁVALOS, Ph.D.

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

LUIS DAVID ORTEGA TOAQUIZA

ERICK STEVEN LOYAGA CARRANZA

Ing. PATRICIO JAVIER CRUZ DÁVALOS, Ph.D.

Ing. HENRY PAUL LEMA ORDÓÑEZ

EDWIN ORLANDO AMAGUAÑA TABANGO

DEDICATORIA

Dedico este trabajo con todo mi amor a mi familia, por su apoyo, amor y confianza durante toda mi vida y carrera universitaria.

AGRADECIMIENTO

Agradezco a Dios por interceder en mi camino, por estar presente durante estos años de estudio y por siempre darme destellos de esperanza cuando más lo necesitaba.

A mi mami Lucy, mami Lucha y papi Rafico por siempre preocuparse por mí, gracias a su esfuerzo, amor y dedicación eh podido llegar a ser quien soy, sin ustedes esto no sería posible.

A mi hermano Edu y su familia, por ser modelo de constancia, amor y ganas de seguir adelante.

A Yoelbis, por ser ejemplo de trabajo, responsabilidad y humildad.

A mi hermanito Yoseph, por haber llegado a nuestras vidas.

A mis amigos y familiares, por su apoyo emocional, sus consejos y motivaciones.

A la Sra. Dina y su familia, por su amabilidad y preocupación durante este tiempo.

A la Escuela Politécnica Nacional, por haberme formado académicamente y por haberme permitido conocer personas maravillosas.

A mis maestros, por transmitir sus conocimientos con vocación.

A mi tutor de tesis, por la paciencia, el aporte y dedicación en la elaboración de este trabajo.

A Erick Loyaga, por ser un gran amigo y por haber puesto su esfuerzo en la realización de nuestros trabajos de titulación.

Al grupo de investigación ATA EPN en especial a quienes conforman el laboratorio de UAS, por guiarme y permitir realizar este trabajo de titulación.

A mi tutor académico, por haber estado al pendiente y dispuesto de atender mis dudas.

A los Brows, quienes siempre estuvieron ahí para mí, gracias por brindarme su amistad y cariño.

ÍNDICE DE CONTENIDO

RESUMEN

El presente Trabajo de Integración Curricular describe el proceso para el diseño e implementación de un prototipo auxiliar que guía la maniobra de aterrizaje de un vehículo aéreo no tripulado (UAV). Se presenta una base teórica que sirve para conocer a breves rasgos temas como: UAVs, aterrizaje autónomo, Pixhawk, MAVLink, que brindan pautas para diseñar el algoritmo que guía el aterrizaje. También se incluye una breve descripción de los sensores más usados en aterrizajes guiados haciendo énfasis en los que conforman el prototipo auxiliar. Este cuenta con una cámara NoIR v1 que permite implementar un sistema de Visión Artificial que identifica, localiza y sigue a un helipuerto representado por un código QR. Adicionalmente, el prototipo integra un sensor LIDAR Lite v3 que permite analizar la altitud real de vuelo y asegurar un aterrizaje controlado. La unidad que procesa la información de los sensores es una Raspberry Pi 4, que además se comunica con el controlador de vuelo a través de mensajes MAVLink. Las pruebas de funcionalidad del prototipo utilizan un UAV tipo cuadricóptero, en el que se adapta una plataforma de carga para poder colocar a bordo el prototipo. Este cuadricóptero está equipado con un controlador de vuelo Pixhawk 2.1. Los resultados obtenidos demuestran las ventajas de emplear un sensor de distancia, especialmente por su mejor precisión y confiabilidad sobre los registros que se procesan en el controlador de vuelo. Además, confirman la flexibilidad del protocolo MAVLink para ejecutar satisfactoriamente las maniobras prestablecidas por el algoritmo de guiado para el aterrizaje.

PALABRAS CLAVE: Aterrizaje Guiado, UAV, MAVLink, Visión Artificial, LIDAR, Helipuerto.

ABSTRACT

This project describes the process for designing and implementing an auxiliary prototype to guide the landing maneuver for an unmanned aerial vehicle (UAV). It presents the theoretical framework that helps to review topics such as UAVs, autonomous landing, Pixhawk and MAVLink, which provide the guidelines to design the algorithm that guides the landing maneuver. Also, this work includes a brief description of the most commonly sensors used on guided landings with emphasis on those that are part of the auxiliary prototype. It has a NoIR v1 camera that allows the implementation of an Artificial Vision system that identifies, locates and follows a heliport represented by a QR code. In addition, the prototype integrates a LIDAR Lite v3 sensor that allows to analyze the real flight altitude and to ensure a controlled landing. The unit that processes the sensor information is a Raspberry Pi 4, which also communicates with the flight controller by MAVLink messages. The functionality tests of the prototype use a quadcopter, in which a loading platform is adapted to place on board the prototype. This quadcopter has a Pixhawk 2.1 flight controller. The test results show the advantages of using a distance sensor, especially with respect to better accuracy and reliability over the records processed by the flight controller. They also confirm the flexibility of the MAVLink protocol to complete successfully the preset maneuvers provided by the landing guidance algorithm.

KEYWORDS: Guided Landing, UAV, MAVLink, Artificial Vision, LIDAR, Helipad.

1 INTRODUCCIÓN

Los vehículos aéreos no tripulados o robots aéreos permiten el desarrollo autónomo o semi-autónomo de diferentes tipos de misiones que cubren desde los sectores de defensa hasta los de agricultura y medio ambiente [1]. Por ejemplo, el grupo de investigación ATA de la EPN viene realizando misiones de campo con UAVs para el monitoreo de volcanes y humedales como es caso del proyecto grupal PIGR 19-01 “ADAPTACIÓN Y OPTIMIZACIÓN DE UN SISTEMA AÉREO NO TRIPULADO DE ALA FIJA (UAS) PARA LA TECNIFICACIÓN DEL PROCESO DE MONITOREO DE HUMEDALES EN EL ÁREA DE CONSERVACIÓN HÍDRICA ANTISANA (ACHA)”. Cabe mencionar que los UAVs que se usan han sido diseñados y construidos por el mismo grupo de investigación.

A bordo de un UAV se encuentran un conjunto de subsistemas y/o elementos que interactúan entre sí para hacer posible el vuelo de esta aeronave [2]. En particular, dentro del subsistema de control es posible integrar autopilotos que pueden recibir misiones desde una estación en tierra o ejecutar acciones a través de un radiocontrol. Lastimosamente si los autopilotos no cuentan con sensores adicionales, la información de las variables físicas de su entorno es desconocida, sin embargo, su diseño hace posible interconectarlos con estos elementos, con el fin de monitorear sus condiciones de trabajo.

Un potencial problema cuando se controla un UAV es el aterrizaje. Las misiones a las que los someten para el análisis de zonas geográficas, como es el caso del proyecto de investigación PIGR 19-01, tienen una amplia zona de vuelo con un relieve muy impredecible. Si bien la simulación de estas misiones tiene resultados favorables, las pruebas reales presentan inconvenientes al momento del aterrizaje. En ocasiones, dentro del proyecto mencionado, los descensos han sido suaves, por lo que difícilmente la estructura del UAV pudo ser afectada, pero también, se han presentado aterrizajes bruscos que han terminado averiando partes de UAV como las hélices o las alas.

En el laboratorio de UAVs del grupo de investigación ATA EPN se tiene a disposición un autopiloto Pixhawk Cube 2.1. Este controlador de vuelo es open–hardware y open–software, con una comunidad amplia de usuarios y desarrolladores, es muy utilizado en el ámbito académico y por usuarios aficionados [3]. Además, tiene una amplia gama de sensores compatibles [4], y que son usados para aplicaciones como: evasión de obstáculos, aterrizaje de precisión, visión por computadora, entre otras. El autopiloto puede recibir comandos a través de MAVLink, un protocolo de mensajería para comunicarse con drones y componentes a bordo [5].

Por esto, es posible extender las tareas del autopiloto Pixhawk Cube 2.1 con sistemas auxiliares que minimicen los riesgos que se pueden presentar al momento de realizar misiones. El presente Trabajo de Integración Curricular busca apoyar específicamente la maniobra de aterrizaje de un UAV implementando un prototipo auxiliar que haga uso del protocolo MAVLink para comunicarse con el autopiloto.

El componente para este Trabajo de Integración Curricular considera varias limitaciones y requisitos, como el tamaño y peso deben ser reducidos con el fin de no ser una carga significativa para el UAV. Además, el protocolo de comunicación del prototipo auxiliar debe ser soportado por el autopiloto al que va apoyar y el procesamiento debe ser suficientemente rápido para tomar decisiones a tiempo. Por esto, está compuesto por una Raspberry Pi 4 que procesará la información del sensor de distancia LIDAR-Lite v3 y de una cámara Pi NoIR V1. Además, el algoritmo de guiado para el aterrizaje es codificado en la Raspberry y se encarga de enviar los comandos al autopiloto Pixhawk.

Este Trabajo de Integración Curricular tiene relación con varios trabajos disponibles en el repositorio digital de la Escuela Politécnica Nacional. Por ejemplo, el prototipo desarrollado en [6] es capaz de transmitir el video de la cámara a bordo en tiempo real dentro de la plataforma Mission Planner. A diferencia del presente componente en donde, la cámara a bordo no transmite el video, sino que procesa la imagen en tiempo real cuando el algoritmo lo decida.

Otro trabajo interesante es [7], donde se compara el controlador de vuelo comercial APM2.6 con los controladores diseñados durante todas las fases de la misión. En cambio, el presente trabajo propone apoyar únicamente en la maniobra de aterrizaje a un autopiloto Pixhawk.

Adicionalmente en [8], se implementó un prototipo que permite dar seguimiento a un UAV. Esta aplicación requirió que el prototipo auxiliar desarrollado se encuentre en tierra, mientras que, para el presente trabajo, el prototipo de apoyo al aterrizaje estará a bordo.

1.1 OBJETIVO GENERAL

Diseñar e implementar un prototipo de un sistema auxiliar para el autopiloto Pixhawk Cube 2.1 que sirva de apoyo para la maniobra de aterrizaje de un vehículo aéreo no tripulado.

1.2 OBJETIVOS ESPECÍFICOS

1. Investigar diferentes opciones para apoyar el aterrizaje de drones basados en autopilotos Pixhawk.

2. Conformar el hardware del prototipo auxiliar con la Raspberry Pi 4, elementos complementarios y los sensores adecuados para caracterizar la zona de aterrizaje.
3. Diseñar e implementar el algoritmo de aterrizaje que será ejecutado por el prototipo auxiliar.
4. Establecer un enlace entre el prototipo auxiliar y el autopiloto Pixhawk Cube 2.1 para comunicar ambos dispositivos a través del protocolo MAVLink.
5. Verificar el funcionamiento del prototipo auxiliar sometiéndole a pruebas en donde se evidencie el apoyo del equipo auxiliar durante la maniobra de aterrizaje.

1.3 ALCANCE

- Se realiza una revisión bibliográfica de las diferentes formas para guiar el aterrizaje de un UAV, identificando los sensores necesarios para su aplicación y que se ajusten al prototipo priorizando sus especificaciones técnicas.

- Se revisan los manuales y guías relacionados con la Raspberry Pi 4, el autopiloto Pixhawk Cube 2.1 y el protocolo de comunicación MAVLink.

- Se eligen los sensores basados en sus especificaciones técnicas, interfaces de comunicación y en la carga útil que soporta el UAV de prueba. Estos sensores se usan para conocer a que distancia se encuentra el UAV de la superficie y para identificar una zona segura de aterrizaje.

- Se conforma el prototipo con los sensores apropiados, la Raspberry Pi 4, circuitos impresos necesarios y las estructuras de soporte impresas en 3D con la ayuda de un ingeniero mecánico, considerando que el peso no represente una carga significativa para el UAV.

- Se diseña el algoritmo de apoyo al aterrizaje y el código que ejecuta el prototipo auxiliar teniendo en cuenta que, debe basarse en el protocolo Mavlink para poder enviar o recibir mensajes del autopiloto Pixhawk.

- Se realiza la conexión entre el prototipo auxiliar y el autopiloto Pixhawk Cube 2.1 para verificar la transferencia de datos, la recepción de comandos y su ejecución.

- Se realizan pruebas en donde el autopiloto no cuente con la asistencia del prototipo auxiliar y otras en donde sí se tiene el apoyo del prototipo auxiliar, a fin de comparar de forma cualitativa ambos escenarios e identificar los beneficios de apoyar al autopiloto mientras realiza la maniobra de aterrizaje.

- Se somete a prueba el funcionamiento del prototipo auxiliar en diferentes zonas de aterrizaje a través de pruebas similares al punto anterior, con el propósito de observar las decisiones que toma el sistema auxiliar de apoyo al aterrizaje en situaciones diferentes.

1.4 MARCO TEÓRICO

En el primer capítulo se desarrollan de forma general los temas más relevantes usados en este trabajo de integración curricular. Se empieza con la información básica referente a los vehículos aéreos no tripulados, luego se describen técnicas de guiado para el aterrizaje, así como sobre la estación de control Mission Planner.

1.4.1 VEHÍCULOS AÉREOS NO TRIPULADOS

El desarrollo de los Vehículos Aéreos No Tripulados (UAVs por sus siglas en inglés) tiene una extensa historia antes de lograr consolidarse como uno de los avances tecnológicos más abordados en la actualidad, cada vez son más las personas que se familiarizan con ellos a medida que se incrementa el número de aficionados [9].

Hoy en día, los UAV hacen referencia solamente a la aeronave no tripulada y son parte integral de un sistema más complejo conocido como Sistema Aéreo No Tripulado (UAS) [10]. Estos sistemas aéreos no tripulados cuentan con los subsistemas que se pueden observar en la Figura 1.1. donde los principales son: el UAV que es la aeronave no tripulada, la estación de control que controla la aeronave de forma remota, el enlace de datos que es el medio por el cual se conecta el UAV a la estación de control y los sensores que se pueden encontrar a bordo del UAV.

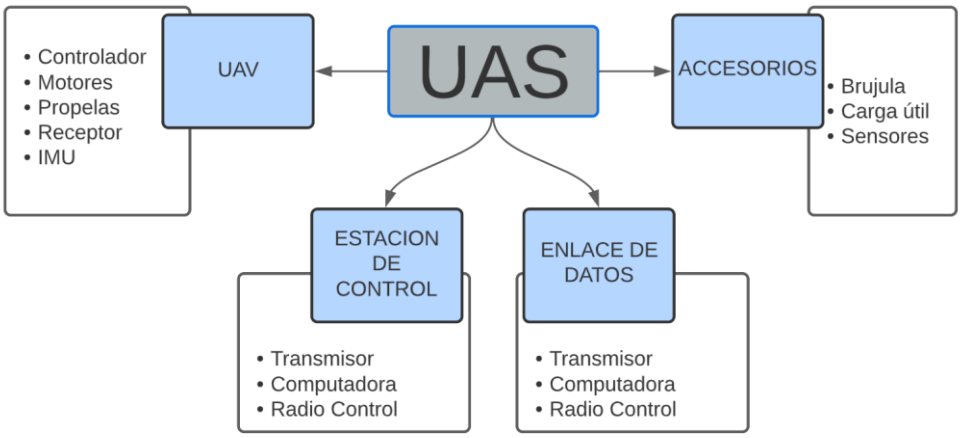


Figura 1.1. Subsistemas de los UAS [Fuente propia].

Los UAVs debido a su gran variedad han sido clasificados en base a diferentes parámetros como: peso, altitud y alcance, motores y alas, tipo de despegue, entre otros [11]. Posiblemente la clasificación de UAVs más conocida es según su tipo de despegue. Como se puede ver en la Figura 1.2. se tiene cuatro categorías: despegue y aterrizaje vertical (VTOL), despegue y aterrizaje horizontal (HTOL), UAVs híbridos y Bio-Based [12]. La principal ventaja de identificarlos con esta clasificación es que describe de forma previa la disposición de sus elementos como: motores, propelas, alas o eslabones según sea el caso. Esto es indispensable para implementar o desarrollar controladores de vuelo, porque dependiendo del tipo de UAV se definen las salidas de control para cada elemento.

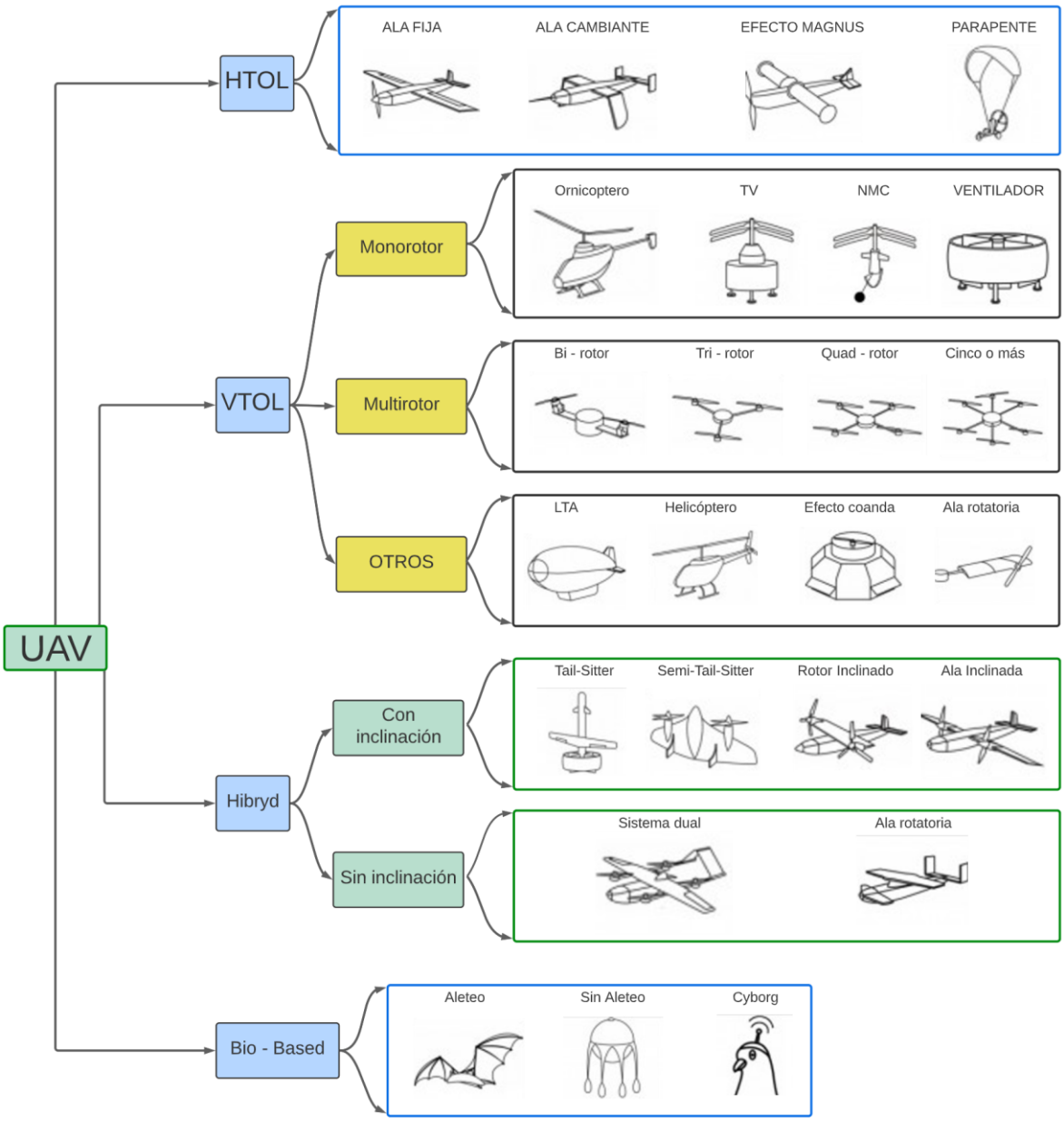


Figura 1.2. Clasificación de los UAV por su despegue [Fuente propia].

El UAV de prueba que será guiado por el prototipo auxiliar es de tipo quad-rotor o cuadricóptero. Es un sistema de seis grados de libertad, como se puede observar en la Figura 1.3., tiene tres grados de libertad traslacionales determinados por los tres ejes fijos (x, y, z) y tres grados de libertad rotacionales debido a sus ángulos de navegación (roll, pitch, yaw) [13].

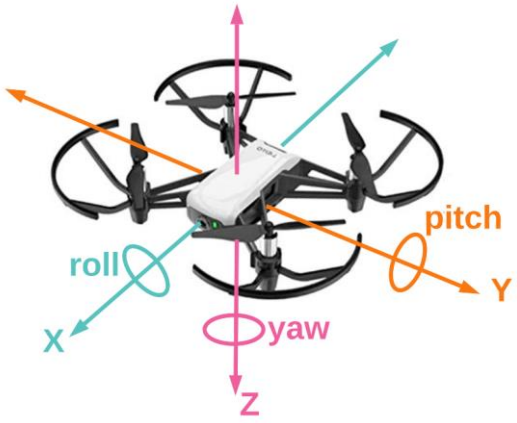


Figura 1.3. Rotaciones yaw, pitch y roll de un UAV cuadricóptero [Fuente propia].

Para poder controlar el movimiento traslacional y rotacional de un UAV se requiere implementar técnicas de control como: controladores PID, controladores no lineales y control inteligente. Estas técnicas controlan directamente el movimiento del UAV y hoy en día forman parte del firmware de autopilotos. Por este motivo el esfuerzo de sistemas auxiliares como el aterrizaje autónomo, se centra en proporcionar las referencias de movimiento necesarias para cumplir con su función.

1.4.2 ATERRIZAJE AUTÓNOMO EN UAVs

Durante el vuelo del UAV se tienen diferentes fases que indican su estado como: take off, on air y landing. El despegue o take off puede ser autónomo si son del tipo VTOL o apoyado si son del tipo HTOL. Una vez en el aire (on air), es posible hacer movimientos tridimensionales o ejecutar misiones, y el aterrizaje o landing da por finalizada la misión. Este en ocasiones, debe realizarse en un tiempo y espacio limitado, motivo por el cual se debe implementar técnicas de detección y control precisas durante esta maniobra.

Además, para lograr un buen aterrizaje se debe tener en cuenta factores como: visibilidad, tipo de terreno, perturbaciones de viento, etc. Además, se deben contar con dos aspectos, la detección y el control. La detección apoyada con sensores o cámaras busca estimar la posición y orientación del UAV, esta información es usada por controladores de cualquier tipo para generar comandos guías como cambios de velocidad, aceleración y rotación para seguir la trayectoria deseada [20].

1.4.2.1 Aterrizaje basado en visión artificial

La visión artificial o visión por computadora permite procesar imágenes digitales con el fin de replicar la visión humana y tomar acciones basadas en la percepción de su entorno. Se usa en sistemas de aterrizaje autónomo donde se requiere identificar zonas de aterrizaje que se encuentren en ubicaciones desconocidas o zonas no estacionarias.

El aterrizaje se puede realizar en interiores y en exteriores. En interiores al tratarse de ambientes controlados, las perturbaciones ambientales se reducen significativamente y el esfuerzo computacional del sistema se reduce. Mientras que, en exteriores, la presencia de perturbaciones como el viento, la visibilidad o luminosidad, asume un reto mayor para el sistema de visión artificial [14].

A continuación, se describen sistemas de visión artificial comunes dentro de esta disciplina y que es necesario diferenciarlos [15]:

- Clasificación de objetos: clasifica una imagen dentro de una o varias categorías también conocidas como clases.
- Identificación de objetos: identifica objetos dentro de una misma imagen y a su vez los clasifica con un cierto grado de certidumbre. Dicho de otro modo, reconoce instancias específicas de una clase.
- Detección y localización de objetos: identifica y clasifica objetos dentro de una misma imagen, además que los localiza dentro de la misma a través de coordenadas rectangulares. Son muy usados en detección de rostros, radiografías y detección de componentes averiados.
- Segmentación e instanciación de objetos: retorna máscaras que alteran la imagen a través de filtros, binarización o segmentación. Busca diferenciar regiones con alguna característica en común, es muy usado en robótica y autos inteligentes para la comprensión de su entorno.
- Seguimiento de objetos: localiza e identifica objetos específicos que pueden o no cambiar de lugar durante una secuencia de imágenes.

1.4.2.2 Aterrizaje guiado

El término guiado se refiere a determinar la trayectoria desde la ubicación local hasta el objetivo. Involucra determinar la orientación, dirección y velocidad con la que se va a recorrer la trayectoria deseada. Existen dos tipos de guiado: el guiado proporcional y el guiado de persecución.

El guiado proporcional mantiene el ángulo constante entre la línea de vista (LOS) y el objetivo, para lo cual la velocidad de rotación yaw es proporcional a la velocidad de rotación del ángulo entre la LOS y el UAV [16]. En la Figura 1.4. se puede apreciar la geometría de este tipo de guiado, donde se introduce el parámetro V_p que es la velocidad de acercamiento.

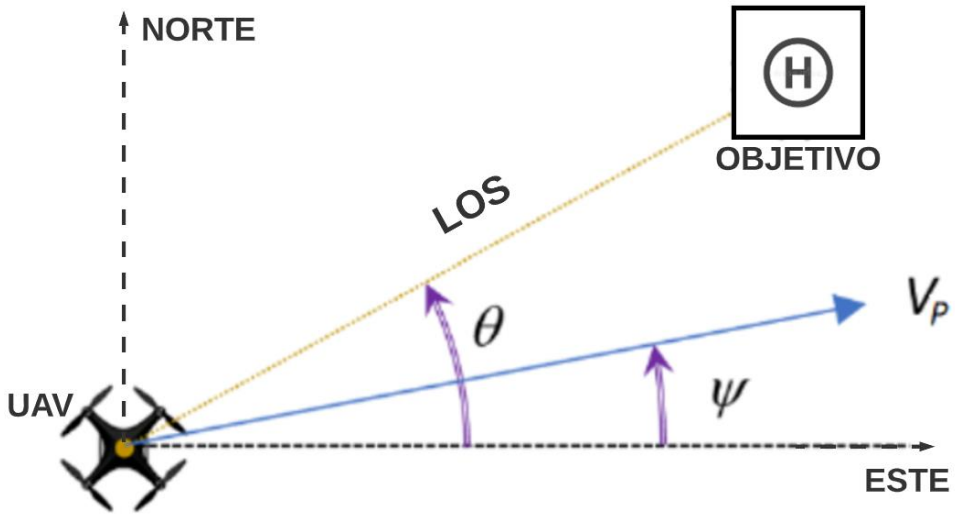


Figura 1.4. Geometría del guiado proporcional.

Este tipo de guiado se formula a partir del control escalar y esta expresado como:

$$\dot{\psi} = \lambda\dot{\theta} \tag{1.2}$$

Donde:

$\dot{\psi} =$ velocidad de rotación del ángulo yaw

$\lambda =$ constante de proporcionalidad

$\dot{\theta} =$ velocidad de rotación de la LOS

Para el guiado de persecución, en la Figura 1.5. se puede observar su geometría y comprender la regla geométrica de la persecución pura, en la cual se deja que el perseguidor se dirija hacia el objetivo y hacer que la dirección de su velocidad coincida con la LOS [17].

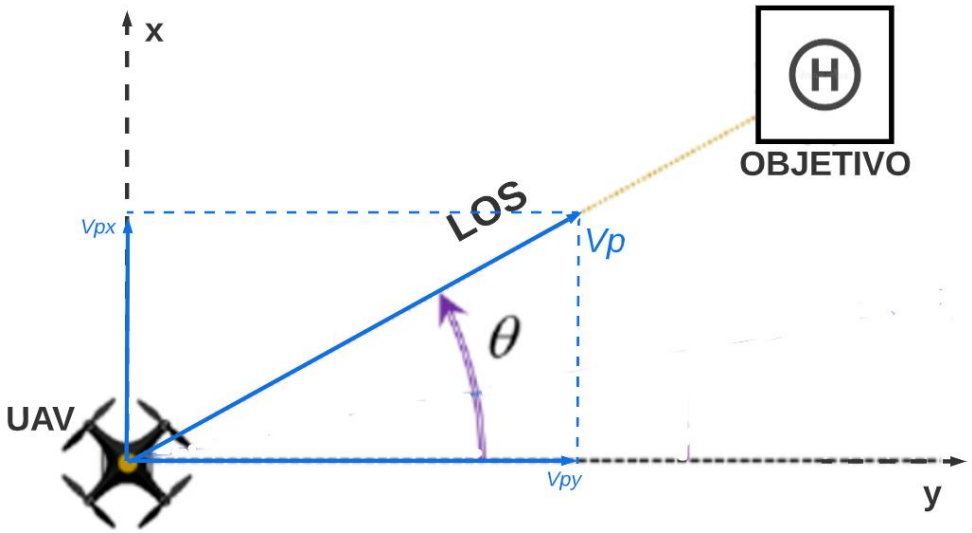


Figura 1.5. Geometría del guiado de persecución.

Tanto el aterrizaje guiado como el aterrizaje basado en visión artificial no pueden actuar por sí solos sobre los actuadores del UAV. Por esta razón se integran con técnicas de control convencionales para conseguir mayor robustez y los autopilotos disponen de hardware y software suficientes para controlar un UAV con dichas técnicas. Su confiabilidad hace posible desarrollar aplicaciones específicas, como es el caso de las técnicas de aterrizaje descritas. Dentro de los autopilotos más conocidos está el Pixhawk, considerado el precursor de la mayoría de autopilotos comerciales.

1.4.3 Pixhawk

Pixhawk es un proyecto de hardware y software libres realizado con el propósito de proveer un autopiloto de bajo costo para la academia y desarrolladores independientes. Comenzó en el año 2008 en Suiza a cargo de un grupo de catorce compañeros de estudio liderados por Lorenz Meier y que buscaban lanzar su trabajo como código abierto. [18].

La arquitectura de control de sus autopilotos es diferente para cada tipo de UAV, por ejemplo, en multicópteros se emplea el control en cascada con una mezcla entre controladores P y PID. La realimentación utiliza filtros EKF para procesar las mediciones de los sensores y estimar estados. En la Figura 1.6. se puede observar la arquitectura de control usada por multicópteros, un control en cascada con lazo externo de posición y lazo interno de velocidad, la notación utilizada se puede revisar en [19]. Según se requiera, se puede omitir el lazo de posición, pues es solo necesario cuando se requiere mantener la posición o cuando la velocidad deseada en un eje es nula [20].

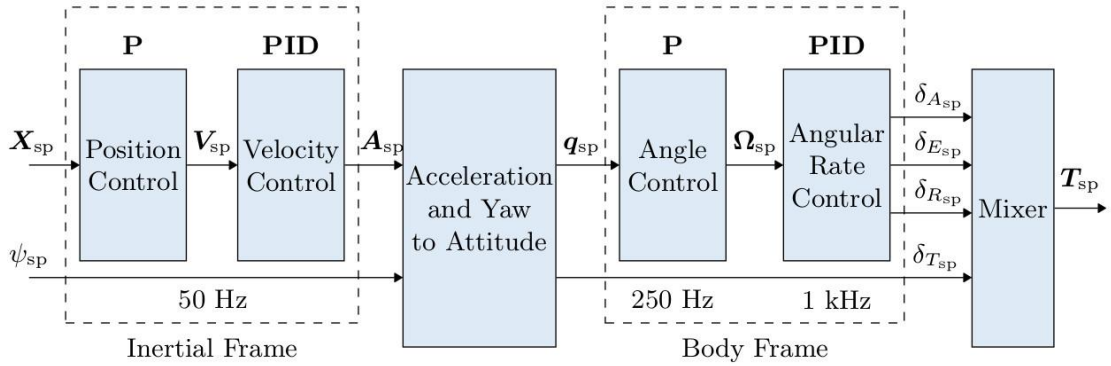


Figura 1.6. Arquitectura de control de multicópteros usada por autopilotos Pixhawk [20].

Quizá lo más importante del software Pixhawk es el protocolo de mensajería MAVLink, mismo que no tardó en usarse en todo el mundo para comunicar UAVs con todo dispositivo que pueda soportarlo.

1.4.3.1 MAVLink

MAVLink es un protocolo de mensajería ligero diseñado para la comunicación con UAVs y sus componentes, pudiendo estos últimos estar o no bordo. Los mensajes MAVLink están definidos en archivos XML conocidos como dialectos y pueden ser enviados a través de cualquier conexión serial sin depender de las tecnologías subyacientes [21].

Los datos pueden ser transmitidos de dos formas: publish-suscribe o point-to-point. Cuando el flujo de datos es de telemetría se usa el modo publish-suscribe donde el suscriptor no es conocido o único, pues la información puede solicitarse de diferentes puntos ya sea un computador a bordo, una estación en tierra, una aplicación móvil y demás. Mientras que, cuando se requiere de una entrega garantizada se usa el modo point-to-point, como es el caso de datos con información de la misión, escritura de parámetros o ejecución de comandos [22].

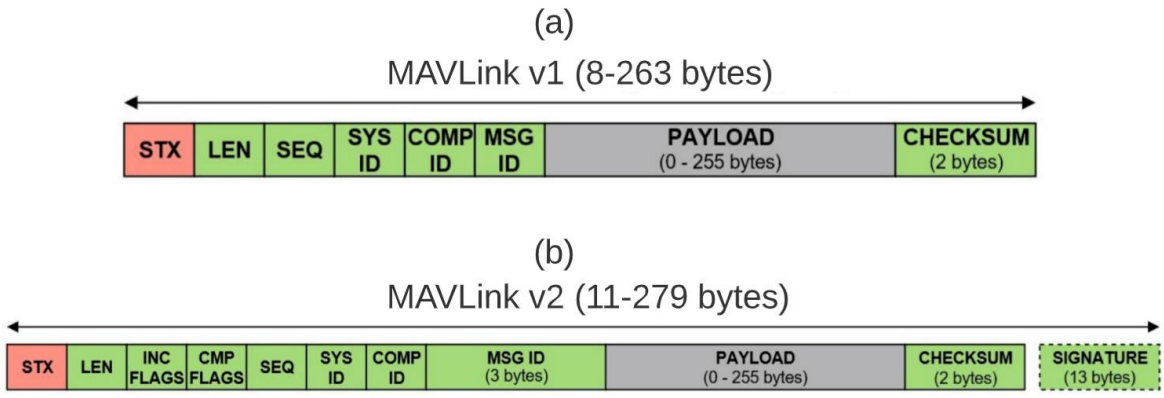


Figura 1.7. Cabeceras de mensajes (a) MAVLink v1 y (b) MAVLink v2.

Actualmente el protocolo cuenta con dos versiones que admiten hasta 255 sistemas simultáneos conectados en red. En la Figura 1.7. (a) se puede observar la cabecera de los mensajes de la versión MAVLink v1 y en la Figura 1.7. (b) la cabecera de los mensajes de la versión MAVLink v2. Esta comparte todos los campos de la primera versión y agrega el campo signature, una mayor cantidad máxima de mensajes y dos banderas de compatibilidad que sirven para identificar ciertas características del mensaje [23]. En la siguiente tabla se describen los campos comunes entre ambas versiones.

Tabla 1.1. Explicación de la trama MAVLink acorde a su contenido.

Acrónimo	Contenido	Descripción
STX	0XFE (v1) 0Xfd (v2)	Describe el inicio de la trama y su valor es constante dependiendo de la versión MAVLink.
LEN	0 – 255	Describe la longitud del mensaje.
SEQ	0 – 255	Secuencia del paquete, empezando con 0 para el primer mensaje. Se usa para la detección de paquetes perdidos.
SYS	1 – 255	ID del sistema no tripulado.
COMP	0 – 255	ID del componente que está enviando el mensaje.
MSG	0 – 255	ID del mensaje y representa a que tipo pertenece.
Payload	0 – 255 bytes	Dato real del mensaje que dependerá de su tipo.
CKA Y CKB	2 bytes	Forman parte del checksum o chequeo de errores. La firma del paquete ocurre desde el bit menos significativo hasta el bit más significativo.

1.4.3.1.1 Tipo de mensajes MAVLink

Los mensajes MAVLink pueden ser de varios tipos y se identifican por su ID, los primeros 255 mensajes son comunes para ambas versiones y los mensajes con identificación superior a 255 son específicos de MAVLink v2. Los dos tipos de mensajes son:

- Mensajes de estado: son los mensajes enviados desde el UAV hacia la estación de control, contienen información del estado del sistema, pose y localización. Por ejemplo: heartbeat (muestra que un sistema o componente está presente y respondiendo) , system_status (indica el estado del sistema), global_position (indica las coordenadas de altitud, longitud y latitud del componente), entre otros.
- Mensajes de mando: son los mensajes enviados desde la estación de control hacia el UAV, son comandos que permiten su maniobrabilidad. Un comando muy útil es

COMMAND_LONG con ID=76, es un comando multipropósito que permite enviar diferentes tipos de mensajes según su ID y sus parámetros. En la Figura 1.8. se ilustra sus 11 campos donde, target_system y target_component especifica el sistema que ejecutará el comando, command se refiere al tipo de comando que será ejecutado, y los parámetros del 1 al 7 que dependen del tipo de comando.

target system uint8_t	target component uint8_t	command unit16_t	confirmation unit8_t	param1 float	param2 float	param3 float	param4 float	param5 float	param6 float	param7 float

Figura 1.8. Comando COMMAND_LONG [23].

A continuación, se presentan los mensajes de mando más comunes y los más usados al momento de realizar misiones automáticas.

Tabla 1.2. Comandos MAVLink más comunes.

Comando	ID	Parámetros	Descripción
TAKEOFF	22	param7: doublé	Hace despegar al UAV a una altitud especificada en param7
LAND	21	-.	Hace aterrizar el UAV
GET_HOME	410	-	Obtiene la posición del primer waypoint de la misión
SET_HOME	179	param5: double param6: double param7: double	Cambia la posición del primer waypoint de la misión
ARM-DISARM	400	param1: booleano	Arma los motores si param1=1 y desarma los motores si param1=0

Como se mencionó, el UAV que soporte MAVLink continuamente está enviando y recibiendo mensajes desde una estación de control. Este protocolo de mensajería permitió que múltiples proyectos de autopiloto desarrollen sus propias estaciones de control, tanto para la instalación del firmware como para la planificación de misiones basadas en waypoints (puntos de ruta). Por ejemplo, Ardupilot lanzo su plataforma Mission Planner como un planificador de misiones que admite una amplia variedad de UAVs y vehículos terrestres.

1.4.3.2 Mission Planner

Las plataformas de software abierto (OSS) aprovecharon el software del proyecto Pixhawk para desarrollar sus propias versiones, además crearon sus propias estaciones de control. Estas cumplen con los requerimientos funcionales que se muestran a continuación [24]:

- Control aéreo del vehículo: capacidad para controlar de forma efectiva el vuelo del UAV durante la misión.
- Control de carga: se refiera a poder control los sensores y dispositivos a bordo del vehículo.
- Planeación de misiones: capacidad para diseñar y ejecutar misiones desde la estación.
- Diagnóstico del sistema: capacidad para verificar el estado del vehículo para verificar si necesita o no mantenimiento.
- Análisis post misión: capacidad fundamental para guardar los datos de vuelo con el fin de analizarlos después.

Mission Planner es una aplicación de estación de control desarrollada por el proyecto ArduPilot y que está disponible únicamente para Windows. Sus principales características son:

- Entrada de waypoints usando varios mapas como: Google Maps, Bing, Open Street maps y Custom WMS.
- Selección de comandos desde menús desplegables.
- Descarga y análisis de archivos de vuelo.
- SITL para UAVs de la serie ArduPilot y Pixhawk.

Además, dentro de la estación se puede cargar el firmware en la placa de piloto automático, configurarlo y calibrar sus sensores para un rendimiento óptimo [25]. Con el hardware adecuado también es posible revisar registros de telemetría y operar el UAV con vista en primera persona.

Aun así, por si sola la estación de control no puede garantizar una buena precisión. Como se había descrito, los autopilotos realizan sus misiones de vuelo a través de waypoints que son ubicados por GPS e IMU. Si bien Mission Planner puede integrar sensores adicionales al autopiloto para mejorar la precisión, la lista es muy corta. Por esta razón en aplicaciones más específicas como la evasión de obstáculos y el aterrizaje autónomo, se necesita de una unidad de procesamiento que acompañe al autopiloto, procese los sensores adicionales y guíe ciertas maniobras a través de mensajería MAVLink.

2 METODOLOGÍA

Para la implementación del componente propuesto en el presente Trabajo de Integración Curricular (TIC) se realiza una investigación aplicada, a través de la cual se consigue guiar el aterrizaje de un UAV de tipo VTOL multirotor (cuadricóptero). La información se obtiene a través de: libros, revistas científicas, artículos científicos, hojas de datos, manuales, repositorios virtuales, etc.

Su diseño está sustentado por la base teórica expuesta en el primer capítulo y su funcionalidad está determinada por las pruebas realizadas. Los resultados obtenidos de dichas pruebas serán analizados en el Capítulo 3.

2.1 GENERALIDADES DEL UAV DE PRUEBA

El prototipo auxiliar del presente TIC apoya la maniobra de aterrizaje de un cuadricóptero con modelo de armazón F450 con las dimensiones de la Figura 2.1. Su propulsión es posible gracias a cuatro motores Brushless A2212/13T de 1000KV. Para el modelo de 1000KV el empuje es de aproximadamente 800 g usando propelas 1045 (diámetro de 10'' y paso de 4.5'') [26], por lo que el cuadricóptero puede levantar hasta 3200 g.

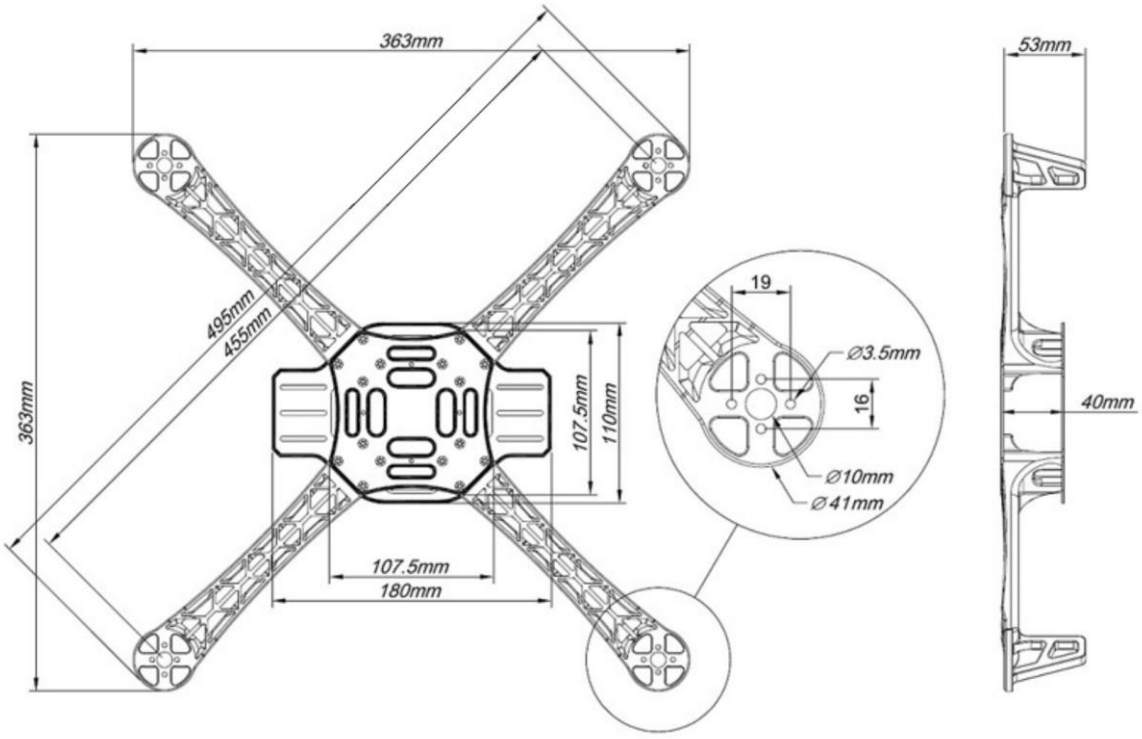
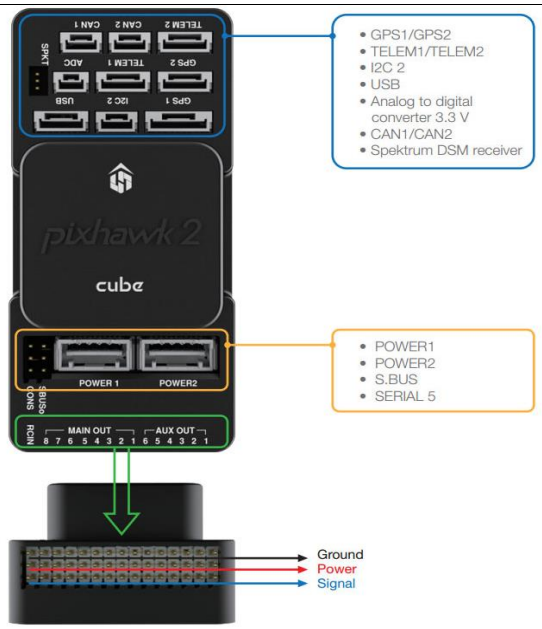


Figura 2.1. Dimensiones del Armazón F450 [27].

La estación de control que se usa es Mission Planner, donde se descarga el firmware del autopiloto y donde se calibran los sensores. El controlador de vuelo que se usa es un Hex

Cube Black conocido también como Pixhawk 2.1, cuyas características se resumen en la Tabla 2.1.

Tabla 2.1. Características del Pixhawk 2.1 [28].

Pixhaw 2.1	Características
	STM32F427 de 32 bits
	168 MHz
	256 kb de RAM
	2MB de memoria flash
	Procesador auxiliar STM32F103 de 32 bits
	14 salidas PWM (8 con seguridad a prueba de fallos y 6 auxiliares compatible con alta potencia)
	Interfaces: UART, I2C, CAN
	Interruptor de seguridad externo
	Micro SD para registro de alta velocidad

Además, para su comunicación con la estación en tierra, control de motores, alimentación y recepción del radio control dispone de los componentes especificados en la Tabla 2.2.

Tabla 2.2. Elementos complementarios del controlador Pixhawk 2.1.

Elemento	Características	Función
GPS Here 2 [29] 	Procesador: STM32F302 Brújula: ICM20948 Barómetro: MS5611 Tasa de actualización de navegación: 10 Hz Sensibilidad de navegación de -167 dBm Conexión UART	Usa los datos GPS para guiar al UAV hacia los puntos de misión.
Tranceptor RFD900+ [30] 	Frecuencia de 902 Hz hasta 928 MHZ Alcance: >40km Potencia de transmisión: 1W (+30dBm) Alemntación: 5V, 800 mA máx	Comunica la estación de control con el UAV

Receptor Flysky FS-IA10B [31] 	Receptor de 10 canales RC 2.4 Ghz Alimentación: 4.0 – 6.5 V	Recibe las señales provenientes del radio control.
Módulo de alimentación PMU [32] 	Medidor de consumo de hasta 30 A Soporta baterías liPo de máximo 8s Medición de voltaje y corriente configurado para 5V	Mide el consumo de la batería, además que regula el voltaje de la batería para alimentar circuitos de potencia y de control.

Todos los dispositivos mencionados se ubican a bordo del UAV y forman parte de la carga que debe soportar. A continuación, se detalla el peso total de la carga incluyendo motores y batería (LiPo de 3s/11.1V):

Tabla 2.3. Carga del UAV de prueba.

Elemento	Peso/u (g)	Cantidad	Peso Total (g)
Motor A2212/1000KV	48	4	192
Pixhawk 2.1	73	1	73
Transceptor RFD900+	41	1	41
Receptor FS – IA10B	19	1	17
GPS here 2	49	1	49
Módulo de alimentación	16	1	16
Batería	675	1	675
Armazón	300	1	300
		TOTAL:	1363

Como se mencionó, la capacidad máxima de carga del cuadricóptero es 3200 g, usando un factor de seguridad de 0.8, la carga máxima ahora es de 2560 g. Si se considera la carga que aporta el UAV de prueba resumida en la Tabla 2.3, se tiene una carga fija de 1363 g. Si esta carga fija es descontada de la carga máxima de 2560 g se obtiene 1197 g de peso máximo que puede tener el prototipo auxiliar. Una vez conocido el peso disponible para el prototipo auxiliar, se realiza una revisión bibliográfica de los sensores de distancia y cámaras más usados para la implementación de sistemas de aterrizaje.

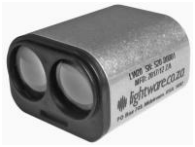
2.2 ESTUDIO DE SENSORES

Para las diferentes técnicas de control implementadas en el aterrizaje de UAVs se requiere la realimentación del sistema usando las mediciones de los sensores a bordo. Los pilotos automáticos usan en sus lazos de control la información de la Unidad de Medida Inercial (IMU) y GPS que son elementos básicos requeridos para su operación. Sin embargo, según su diseño disponen de puertos seriales que admiten la conexión de sensores con diferentes principios de funcionamiento como: flujo óptico, ultrasonido, infrarrojos, entre otros. Para el caso de los autopilotos Pixhawk los sensores más relevantes para guiar el aterrizaje de UAVs son: sensores de distancia y cámaras apoyadas con un sistema de visión por computadora.

Los sensores de distancia recomendados por la guía de usuario Pixhawk se pueden revisar en [33]. A continuación, se muestran sus características más importantes:

Tabla 2.4. Sensores comerciales de distancia [Fuente propia].

NOMBRE	CARACTERISTICAS	PESO	RANGO	CONEXIÓN	USO	PRECIO
ARK Flow 	Sensor de flujo óptico PixArt PAW3902 Sensor de distancia de tiempo de vuelo AFBR-S50LV85D IMU Bosh BMI088 de 6 ejes STM32F412CEU6 MCU LED IR de 40 mW Alimentación: 5V y 76mA Max	5.0 g	>8 cm	CAN	Exteriores	$250.00
Holybro ST VL53L1X Lidar 	Sensor de rango láser. Emisor: láser invisible de 940 nm. Frecuencia de alcance de hasta 50 hz. FoV típico: 27° Matriz de recepción SPAD Interfaz I2C (hasta 400 kHz) Alimentación: 3.3 - 5 V.	2.4 g	hasta 40 m	I2C	Interiores	$14.00
LIDAR – Lite v3 	Telémetro Láser Resolución: 1cm Voltaje de funcionamiento: 4.75 - 6 V Temperatura de funcionamiento: -20 a 60° C Longitud de onda láser: 905 nm Potencia máxima: 1.3 W Apertura ótica: 12.5 mm	22g	0.05 - 40 m	PWM I2C	Exteriores	$220.00

MaxBotix I2C - MaxSonar-EZ 	Telémetro basado en sonar de corto alcance Resolución: 1cm Sensor: ultrasónico de 42 Khz Alimentación: 3-5.5 V a 4.4 mA	5.9g	0.25 - 7.65 m	I2C	Interiores	$51.69
Lightware LW20 Lidar 	Altímetro láser de uso general. Impermeabilizado (IP67) con servo para aplicaciones de detección y evitación 48 - 388 lecturas por segundo Alimentación: 4.5 - 5.5 V a 100 mA Apertura: 28 mm x 15 mm	20 g	0.2 - 100 m	I2C	Exteriores	$279.00
TeraRanger Evo 60 m 	Sensor basado en tecnología infrarroja. Alimentación: 5 V a 330mA max Más de 240 lecturas por segundo	12 g	0.5 - 60 m	I2C USB 2.0	Interiores/ Exteriores	$111.30
Altímetro Ainstein US - D1 	Telémetro de microondas (IP67). Temperatura de funcionamiento: --20 a 60° C Alimentación: 5 - 13 V Potencia: 2 W Campo de visión: 43° x 30°	100 g	0.5 - 50.0 m	UART CAN	Exteriores	$599.00
LeddarOne 	Módulo lidar Haz de luz difusa de 3 grados. Velocidad de adquisición de datos rápida (hasta 140 hz) Alimentación: 5 V Potencia: 1.3 W	14 g	1cm - 40 m	UART serial	Exteriores	$135

Tfmini 	Sensor lidar de bajo costo. FOV: 2° Temperatura de funcionamiento: 0 a 60° C	5 g	0.1 - 12 m	UART I2C	Exteriores	$76.77
PSK-CM9JL65-CC5 	Sensor de distancia por infrarrojos IP50 Fuente de luz: LED infrarrojo (850 nm) Ángulo de divergencia: ± 1.4 ° Resolución: 1 mm Alimentación: 5 V a 85 mA max	10 g	0.17 - 8 m	UART/serial	Exteriores	$26.90

En el caso de las cámaras, para el sistema de aterrizaje, existen módulos que integran sensores infrarrojos y microcontroladores que procesan las imágenes en tiempo real. Conforman un sistema de visión artificial completo que se puede usar directamente sobre autopilotos Pixhawk [34], como es el caso de las opciones especificadas en la Tabla 2.5.

Tabla 2.5. Cámaras para aplicaciones de visión por computadora.

NOMBRE	CARACTERISTICAS	PESO	PRECIO
DUO MLX 	0.1 - 3000 fps Lente gran angular de 165 ° Baja distorsión < 8.5% Distancia focal: 2.0 mm - 2.1 mm Detección de movimiento hasta 250 hz IMU de 6 GDL Consumo: 2.5 W a 5V DUO OS - Kernel	12.5 g	$695.00
Intel RealSense Depth Camera D415 	Campo de visión:69ºx42ºx77º Cámar Full HD RGB calibrada y sincronizada para datos de profundidad. Procesador: Intel® RealSense™ Vision Processor D4 Resolución: 1080p Sensor de imagen: OV2740 Alimentación: 5V a 700 mA max	72 g	$259.00

Como se puede apreciar, las cámaras DUO MLX y Depth Camera D415 tienen características enfocadas a la visión artificial como un seguidor de objetos integrado y análisis de profundidad. Sin embargo, los precios elevados pueden limitar su implementación en prototipos de bajo costo. Por esta razón la alternativa es integrar cámaras comerciales de bajo costo y buen rendimiento como las que se tienen a continuación:

Tabla 2.6. Cámaras comerciales de bajo costo.

NOMBRE	CARACTERISTICAS	PESO	PRECIO
Cámara Raspberry Pi v2 [35] 	Sensor: Sony IMX 219 PQ CMOS Resolución: 8MP Tamaño de lente: ¼''	3 g	$50.00
Cámara NoirIR v1 [36] 	Sensor: OmniVision OV5647 Apertura: 1.8 Resolución: 5MP Tamaño de lente: ¼''	17 g	$35.00
Cámara VGA OV7670 [37] 	Sensor: OmniVision CMOS VGA OV7670 Sensor: 0.3 MP Voltaje: 3.3. V Consumo: 60 mW Lente óptico: 1/6''	1 g	$6.39
Arducam [38] 	Sensor: OmniVision OV2640 Sensor: 2 MP Voltaje: 5 V Consumo: 125 mW Lente óptico: ¼ '' Interfaz: I2C, SPI.	3 g	$26.00

Expuestos los sensores más conocidos para guiar el aterrizaje de UAVs y teniendo en consideración su peso, disponibilidad e interfaz de comunicación. Se eligen los adecuados para ser parte del prototipo auxiliar que se describe a continuación.

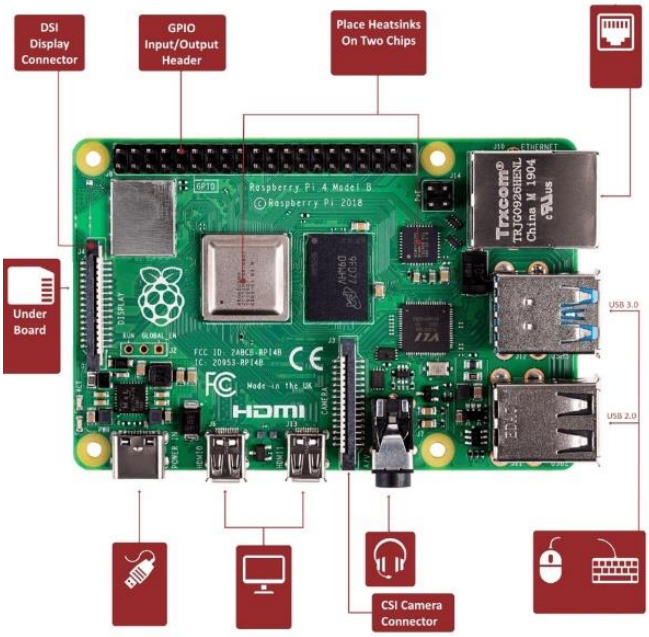
2.3 GENERALIDADES DEL COMPONENTE

El prototipo auxiliar que guía el controlador de vuelo Pixhawk 2.1 está conformado por una un sensor de distancia y una unidad de procesamiento proporcionados por el grupo de

investigación ATA EPN. Adicionalmente se adquirió una cámara de bajo costo para implementar herramientas de visión por computadora.

La unidad de procesamiento disponible es una Raspberry Pi 4, que es una computadora de placa reducida con sistema operativo Raspbian. Dentro de sus aplicaciones más comunes están: proyectos de automatización, implementación de clústeres, visión por computadora, entre otros. Esto porque contiene un conjunto de entradas y salidas de propósito general (GPIO) destinadas para el control de dispositivos electrónicos [39]. A continuación, se presentan sus características más importantes:

Tabla 2.7. Características de la Raspberry Pi 4 [40].

Raspberry Pi 4	Características
	CPU: Procesador Cortex-A72 de cuatro núcleos a 1,5 GHz
	GPU: VideoCore VI de 32 bits a 500 MHz
	Memoria: 1/2/4GB LPDDR4 RAM
	Conectividad: Wi-Fi 802.11ac Wi-Fi / Bluetooth 5.0 / Gigabit Ethernet
	Puertos: 2 x USB 3.0, 2 x USB 2.0
	Alimentación: 5V/3A vía USB-C, 5V vía GPIO
	Expansión: Cabezal GPIO de 40 pines

La unidad de procesamiento se encarga de realizar la lectura del sensor de distancia disponible que es un LIDAR Lite v3 y de capturar la imagen de la cámara NoIR v1. La cámara posee una interfaz CSI (interfaz serie para cámaras) y su módulo principal es una cámara Raspberry v1, que es compatible con todas las versiones de Raspberry Pi.

El sensor de distancia LIDAR Lite v3 tiene una interfaz de comunicación I2C que puede conectarse al cabezal GPIO de la Raspberry Pi, además que sus características para trabajar en exteriores, mediciones de hasta 40 metros y resolución de 1centímetro, son adecuadas para apoyar la medición de altura real del prototipo auxiliar. Con esto el peso aproximado del componente se resume de la siguiente manera:

Tabla 2.8. Carga del prototipo auxiliar.

Elemento	Peso (g)
Raspberry Pi 4	45
Sensor Lidar Lite v3	22
Cámara NoIR v1	17
TOTAL:	84

Con los 84 g de peso total del prototipo auxiliar se cumple con ser menor a la carga útil disponible de 1197 g.

2.3.1 ADAPTACIÓN DEL CUADRICÓPTERO – ESTRUCTURAS 3D

Las dimensiones del cuadricóptero de prueba no disponen de un espacio suficiente para colocar el prototipo auxiliar. Por esta razón se adapta una plataforma de carga que permite ubicar los sensores y la Raspberry en una posición segura con fácil acceso al controlador Pixhawk 2.1. Como se observa en la Figura 2.2. se adapta el tren de aterrizaje junto a la plataforma de carga, adicionalmente cuenta con estructuras que facilitan cambiar la orientación vertical de los sensores a una posición horizontal. Esto con el fin de emplear el mismo cuadricóptero para probar el funcionamiento de otros prototipos auxiliares, como por ejemplo uno basado en evasión de obstáculos que puede requerir ubicar los sensores con la mira al frente.

Figura 2.2. Modelo 3D del cuadricóptero de prueba con la adaptación del tren de aterrizaje (Los planos a detalle se puede ver en el ANEXO I).

El tren de aterrizaje está conformado por cuatro soportes, uno de ellos como el que se observa en la Figura 2.3., y que incluyen dos travesaños en los que se asegura la plataforma de carga.

Figura 2.3. Modelo 3D de dos soportes con un travesaño.

La plataforma de carga que se puede observar en la Figura 2.4. cuenta con orificios para pernos M3 que permiten asegurar: la cámara NoIR v1 en la parte central, la plataforma a los travesaños y las estructuras adicionales al extremo. También tiene orificios rectangulares para el paso de cables y sujetadoras para asegurar los demás dispositivos.

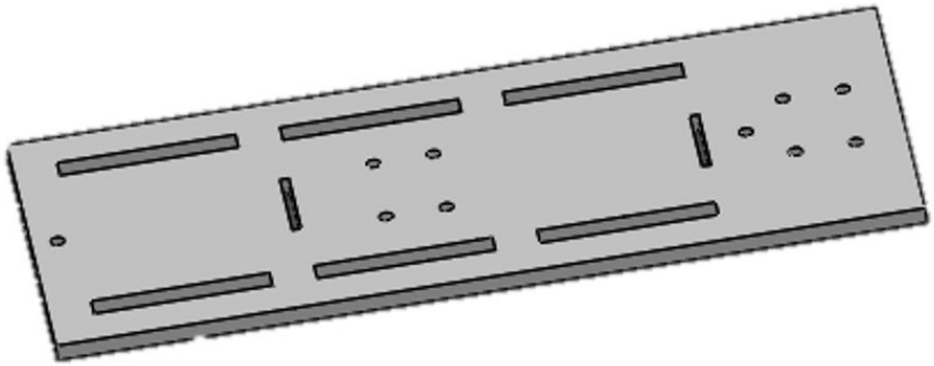


Figura 2.4. Modelo 3D de la plataforma de carga.

Con la colaboración de personal del Departamento de Ingeniería Mecánica de la EPN y pertenecientes al Grupo ATA, todas las piezas 3D se sometieron a pruebas simuladas en SolidWorks (software de diseño CAD para modelar piezas) para verificar su resistencia a deformaciones y desplazamientos con una carga sobredimensionada de 3kg (30N aproximadamente) y usando como material ABS (polímero termoplástico para el modelado en impresoras 3D). Un ejemplo de esta verificación es lo presentado en la Figura 2.5. que muestra la simulación de desplazamiento de la plataforma de carga al aplicarse una fuerza de 30N, y que muestra un desplazamiento máximo en su centro de 0.964 mm.

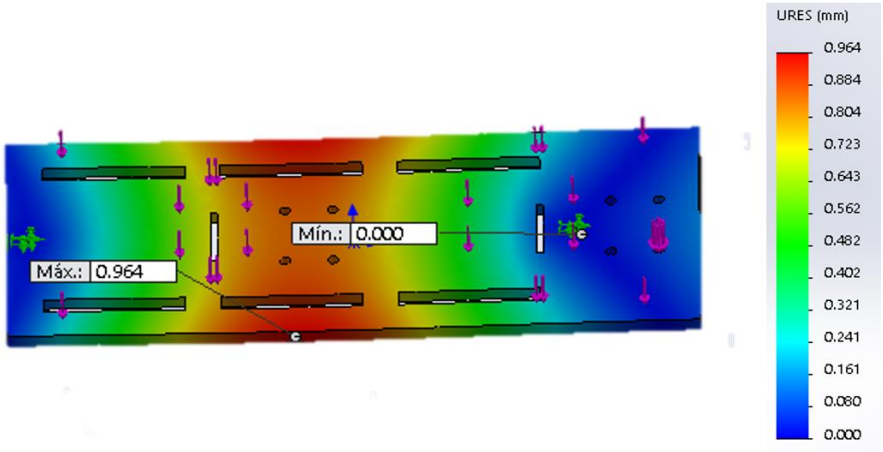


Figura 2.5. Simulación de desplazamiento de la plataforma de carga.

Las adaptaciones del tren de aterrizaje junto a la plataforma de carga aportan un peso adicional. Las condiciones de carga se resumen en la Tabla 2.9. que consideran los valores detallados en la Sección 2.1 y en la Sección 2.2. El incremento de 38 g de carga con las adaptaciones realizadas no afecta significativamente la carga útil disponible de la Sección 2.1.

Tabla 2.9. Carga total del UAV.

Elemento	Peso Total (g)
Carga del UAV de prueba (Sección 2.1)	1363
Carga del tren de aterrizaje	38
Carga del prototipo auxiliar (Sección 2.2)	84
Carga total	1485

2.3.2 INTEGRACIÓN DEL PROTOTIPO Y CONEXIÓN CON EL AUTOPILOTO

La compatibilidad de interfaces de la cámara NoIR v1, sensor LIDAR Lite v3 y Pixhawk 2.1 con la Raspberry hacen que no sea necesario realizar circuitos impresos para su conexión. La cámara NoIR v1 se conecta directamente al puerto CSI de la Raspberry que se configura a través de los siguientes los pasos:

- Abrir la configuración de la Raspberry escribiendo en el terminal: sudo raspi-config.
- Seleccionar “Opciones de interfaz”.
- Habilitar Legacy camera.
- Reiniciar la Raspberry.

El sensor LIDAR Lite v3 de igual forma se conecta directamente al cabezal GPIO de la Raspberry por medio del protocolo I2C que debe ser habilitado en la Raspberry con el siguiente proceso:

- Abrir la configuración de la Raspberry escribiendo en el terminal: sudo raspi-config.
- Seleccionar “Opciones de interfaz”.
- Habilitar el bus I2C.
- Reiniciar la Raspberry.

Y el Pixhawk 2.1 puede conectarse con la Raspberry por uno de sus puertos UART, pero antes se configuran de la siguiente manera.

Configuración de la Raspberry:

- Abrir la configuración de la Raspberry escribiendo en el terminal: sudo raspi-config.
- Seleccionar “Opciones de interfaz”.
- Habilitar el hardware del Puerto serie (Se habilita el puerto: /dev/serial0).

Configuración del Pixhawk:

- Habilitar MAVLink en el puerto serie de Telem 2: SERIAL2_PROTOCOL=2.
- Configurar la velocidad de transmisión: SERIAL2_BAUS=57600.

Ahora con los dispositivos ya seleccionados y el hardware del prototipo auxiliar conformado, en la Figura 2.6. se esquematiza la conexión de los sensores y el controlador de vuelo con la Raspberry. Los detalles completos del diagrama de conexión se encuentran en el ANEXO II.

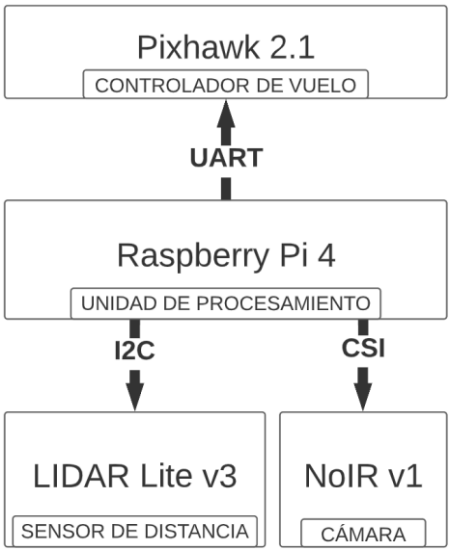


Figura 2.6. Diagrama de conexiones del prototipo auxiliar [Fuente propia].

A continuación, se realiza el diseño del algoritmo de guiado para el aterrizaje que toma en cuenta la medición del sensor de distancia y la imagen captada por la cámara para establecer las acciones necesarias para asegurar un aterrizaje sin complicaciones.

2.4 ALGORITMO DE GUIADO PARA EL ATERRIZAJE

En la Figura 2.7. se muestra el diagrama de flujo del algoritmo que guía el aterrizaje del cuadricóptero. Inicialmente se establece la comunicación a través de mensajes MAVLink y se empieza a comprobar el estado de vuelo. Solo cuando se detecta que el aterrizaje ha empezado, este se interrumpe manteniendo su última posición. Con la ayuda del sensor de distancia se verifica la altitud real respecto al suelo a la que se encuentra el cuadricóptero y en función de ella se decide si desciende a una altura segura o mantiene su altitud.

Una vez en ella, se encenderá la cámara y entrará en funcionamiento el sistema de visión artificial. Este reconoce si en el campo de visión de la imagen existe un helipuerto, mismo que es representado por un código QR y su función se detalla en la Sección 2.7 que abarca la implementación del sistema. En el caso de existir, se procede a centrar el cuadricóptero con la plataforma de aterrizaje, y luego se realiza el descenso para finalizar con la maniobra.

Caso contrario, se asume la ausencia del helipuerto y con el sensor de distancia se testea la zona en la que se encuentra. Luego de verificar que el suelo debajo es lo suficientemente seguro para aterrizar, se realiza el descenso que finaliza la maniobra de aterrizaje. En cambio, si la zona de aterrizaje no es segura se cambia de posición del cuadricóptero y se reinicia el proceso desde el encendido de la cámara.

Para ambos casos, cuando se reconoció un helipuerto y cuando se testeo la zona de aterrizaje, el descenso posterior se realiza disminuyendo la velocidad de aterrizaje progresivamente hasta aterrizar.

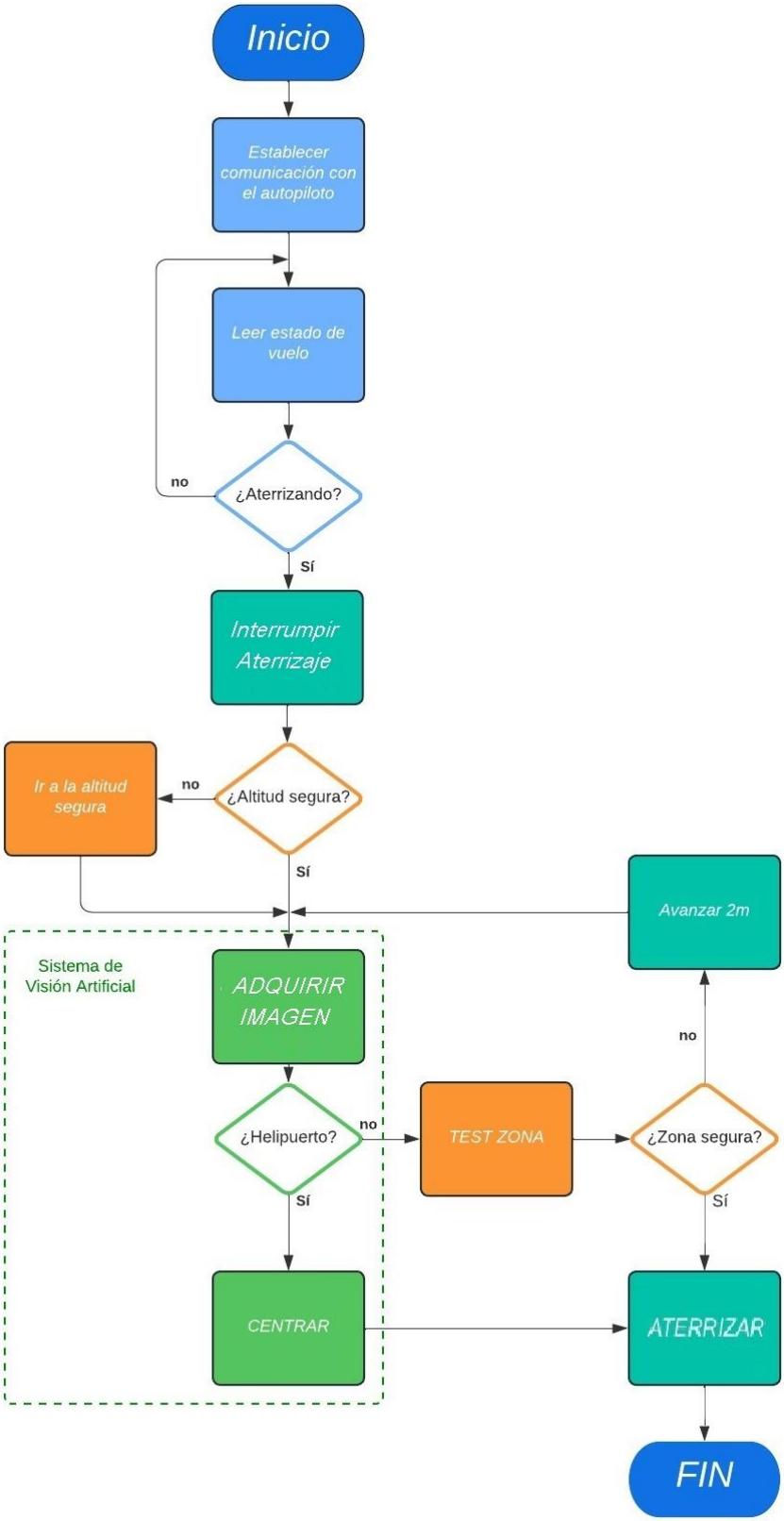


Figura 2.7. Algoritmo de guiado para el aterrizaje, guía de colores: recepción de datos desde el autopiloto (celeste), transmisión de datos y comandos hacia el autopiloto (verde turqueza), cámara NoIR como sensor principal (verde brillante) y LIDAR Lite v3 como sensor principal (naranja) [Fuente propia].

Este algoritmo se implementa en la Raspberry Pi 4 usando el lenguaje de programación Python debido a que es de código abierto, portable y multiplataforma. También permite integrar múltiples herramientas para el control del cabezal GPIO de la Raspberry, acceder a la cámara NoIR v1, instalar OpenCV e incluir dialectos (archivos XML como se indicó en la sección 1.4.3.1) para usar el protocolo MAVLink.

2.5 COMUNICACIÓN VÍA MAVLink

Para implementar el protocolo de comunicación MAVLink en la Raspberry es necesario disponer de un dialecto estándar, es decir que contenga al menos los siguientes archivos XML [41]:

- minimal.xml: conjunto de mensajes y comandos necesarios para configurar una red MAVLink.
- standard.xml: conjunto estándar de mensajes y comandos implementados por casi todas las *pilas de vuelo* (conjunto de algoritmos de guía, navegación y control para UAVs). Incluye el archivo miniam.xml.
- common.xml: conjunto de mensajes y comandos incluidos en la pila de vuelo central.

Existen bibliotecas en Python que procesan mensajes MAVLink y permiten la creación de los mismos. Una de ellas es Pymavlink (para versiones superiores a Python 2.7) que procesa MAVLink de bajo nivel, y es compatible con las dos versiones del protocolo MAVLink [42].

2.5.1 ESTABLECER COMUNICACIÓN

Para abrir una conexión, el módulo *mavutil* de la bliblioteca Pymavlink tiene el método `mavlink_connection()`, (`Las implementaciones en Python tendrán este estilo de fuente en este documento`). En este se definen las propiedades del enlace [48]. Su implementación de forma general es de la siguiente manera: `the_connection=mavutil.mavlink_connection('port:localhost:baudrate')`. En la Raspberry usando el puerto serial, los parámetros de comunicación se configuran con port=/dev/serial0 y baudrate=57600:

```
# ABRIR CONEXIÓN
the_connection= mavutil.mavlink_connection('/dev/serial0',baud=57600)
```

Cuando ya se abre la conexión, se buscan UAVs (sistemas) que estén presentes y respondan a la conexión. El mensaje HEARTBEAT con ID=0, recibe datos enviados por

sistemas a 1Hz que sirven para identificar: tipo de sistema (vehículo o componente), tipo de autopiloto, estado del sistema, entre otros [43]. Pymavlink tiene el método `wait_heartbeat()` que almacena mensajes que identifican los sistemas que responden a la conexión. Con esto se establece la comunicación entre el punto de conexión y los sistemas que respondieron, lo que les permite enviarse y recibir mensajes MAVLink.

2.5.2 MENSAJES MAVLink

Pymavlink tiene tres herramientas para enviar mensajes: para crear un paquete sin procesamiento adicional (`command_long_send`), para enviar mensajes simples (`<message_name>_send`) y para abstraer mensajes (`mavutil`) [44].

Si se usa `mavutil`, el atributo `mav` permite acceder a un objeto de clase que se puede usar para enviar mensajes simples [48]. Por ejemplo, para publicar un latido (mensaje heartbeat):

```
# Desde una estación de control
# Los tipos de sistema están definidos como enum dentro del dialecto).
the_connection.mav.heartbeat_send(mavutil.mavlink.MAV_TYPE_GCS,
                                  mavutil.mavlink.MAV_AUTOPILOT_INVALID, 0, 0, 0)
```

El envío de mensajes hacia el sistema se utiliza con múltiples propósitos como: escribir parámetros, ejecutar comandos, solicitar mensajes específicos, entre otros. Uno de los objetos más utilizados para el envío de datos y que permite utilizar casi todos los mensajes MAVLink es el `command_long_send()`. Permite enviar mensajes específicos conociendo su identificación y definiendo claramente sus parámetros, se pueden revisar el listado de mensajes comunes junto a su ID y parámetros en [43]. Un ejemplo de su aplicación para realiza la maniobra de TAKEOFF a una altura de 10 metros es el siguiente:

```
# TAKEOFF
the_connection.mav.command_long_send(the_connection.target_system,
        the_connection.target_component,mavutil.mavlink.MAV_CMD_NAV_TAKEOFF,
        0, 0, 0, 0, 0, 0, 0, 10)
```

Mientras que, para recibir mensajes, primero se debe solicitar que el sistema los envíe. Para esto se emplea el formato anterior y se puede usar cualquiera de los mensajes: REQUEST_DATA_STREAM, SET_MESSAGE_INTERVAL o REQUEST_MESSAGE. Sus parámetros se pueden revisar en [49] y un ejemplo de su implementación es el siguiente:

```
 # REQUERIR MENSAJE SYS_STATE CON ID=245 Y CON FRECUENCIA DE 100 us
# SET_MESSAGE_INTERVAL
```

```
the_connection.mav.command_long_send(the_connection.target_system,
        the_connection.target_component,mavutil.mavlink.MAV_CMD_SET_MESSAGE_I
        NTERVAL,0, 245, 100, 0, 0, 0, 0, 10)
```

Pymavlink puede recibir mensajes de forma sincrónica y asincrónica con mavutil. Para recibir el último mensaje específico, se utiliza `mavutil.messages` que forma parte del diccionario de la conexión. Por ejemplo, para adquirir el dato de la altitud actual con el mensaje GPS_RAW_INT:

```
# RECIBIR MENSAJE GPS_RAW_INT
altitud=the_connection.messages['GPS_RAW_INT'].alt
```

Mientras que, para interceptar mensajes a medida que van llegando se usa el método `recv_match()` que puede recibir todos los mensajes que se estén transmitiendo como filtrar mensajes específicos. Es común usar esta alternativa para verificar la ejecución de un comando con el mensaje COMMAND_ACK:

```
 # FILTRAR MENSAJE DE RECONOCIMIENTO DE COMANDOS
msg=the_connection.recv_match(type='COMMAND_ACK',blocking=True)
```

Dentro del mensaje COMMAND_ACK se tiene un campo especial que indica si se ejecutó o no el comando enviado, este es el MAV_RESULT y en la Tabla 2.10. se indica el significado de sus valores.

Tabla 2.10. Valores del parámetro MAV_RESULT [43].

Valor	Nombre del campo	Descripción
0	MAV_RESULT_ACEPTED	El comando es válido y se ejecutó.
1	MAV_RESULT_TEMPORARYLY_REJECTED	El comando es válido, pero no se puede ejecutar debido al estado del sistema. Se debe intentar después.
2	MAV_RESULT_DENIED	El comando no es válido porque tiene parámetros incorrectos.
3	MAV_RESULT_UNSUPPORTED	El comando es incompatible o desconocido.
4	MAV_RESULT_FAILED	El comando es válido, pero falló la ejecución.
5	MAV_RESULT_IN_PROGRESS	El comando se está ejecutando y es válido.
6	MAV_RESULT_CANCELLED	El comando fue cancelado

Como se observa, el resultado del reconocimiento de comandos (MAV_RESULT) indica solo la forma en la que se ejecutó el comando, pero no se tiene una realimentación de si se completó o no la acción, lo que es un problema evidente cuando se envía mensajes que cambian la posición del sistema. Por lo que, en el presente prototipo auxiliar para asegurarse que el UAV se ubicó en la posición deseada antes de continuar con la siguiente, *se debe recibir continuamente la posición actual y compararla con la deseada*.

2.5.3 MODOS DE VUELO Y COMANDOS DE MOVIMIENTO

Antes de enviar mensajes que controlen la posición, velocidad u orientación, se debe considerar que esto solo es posible en modo guiado. Es uno de los varios modos de vuelos disponibles por el autopiloto, estos se pueden controlar con interruptores del radio control, comandos enviados desde una estación de control o por una computadora a bordo [45]. A continuación, se presentan los principales modos de vuelo para cuadricópteros:

Tabla 2.11. Principales modos de vuelo [45].

Modo	**Descripción**
Alt Hold	Mantiene la altitud con niveles seguros de roll y pitch
Auto	Ejecuta misiones predefinidas
Guided	Navega a coordenadas simples con una estación de control
Loiter	Mantiene la altitud y posición, se usa el GPS para su movimiento
RTL	Regresa al punto de donde se realizó el despegue y aterriza.
Stabilize	Asegura niveles seguros de roll y pitch

Los comandos de movimiento son: SET_POSITION_TARGET_LOCAL_NED, SET_POSITION_TARGET_GLOBAL_INT y SET_ATTITUD_TARGET [46]. Para los dos primeros, la configuración del comando depende de dos factores importantes, primero el MAV_FRAME que indica el formato de las coordenadas con las que se configura el mensaje. Los valores más comunes que puede tomar este parámetro se presentan en la Tabla 2.12. donde se muestran los nombres asociados a cada valor y una breve descripción de los formatos.

Para el comando SET_POSITION_TARGET_LOCAL_NED se usan los comandos MAV_FRAME con prefijos MAV_FRAME_LOCAL o MAV_FRAME_BODY, mientras que para el comando SET_POSITION_TARGET_GLOBAL_INT se usan los MAV_FRAME con prefijos MAV_FRAME_GLOBAL.

Tabla 2.12. Valores del parámetro MAV_FRAME [43].

Valor	Nombre	Descripción
0	MAV_FRAME_GLOBAL	Formato de coordenadas globales (WGS84). x: latitud y: longitud z: altitud en metros sobre el nivel del mar (msnm)
1	MAV_FRAME_LOCAL_NED	Tramas geográficas en relación a la posición LOCAL (desde la posición inicial). x: norte (North) y: este (East) z: altitud (Down) en metros, con signo positivo hacia abajo y negativo hacia arriba.
3	MAV_FRAME_GLOBAL_RELATIVE_ALT	Coordenadas WGS84 pero con altitud relativa a la posición inicial del sistema.
4	MAV_FRAME_LOCAL_ESP	Mantiene la altitud y posición, se usa el GPS para su movimiento
5	MAV_FRAME_GLOBAL_INT	Coordenadas WGS84 escaladas. x: latitud en grados x 1e7 y: longitud en grados x 1e7 z: altitud en metros sobre el nivel del mar (msnm)
6	MAV_FRAME_GLOBAL_REALATIVE_ALT_INT	Coordenadas de (5) pero con altitud relativa a la posición inicial del sistema.
7	MAV_FRAME_LOCAL_OFFSET_NED	Coordenadas geográficas relativas a la posición del sistema (con origen a bordo). x: norte (North)

		y: este (East) z: altitud (Down) en metros, con signo positivo hacia abajo y negativo hacia arriba.
8	MAV_FRAME_BODY_NED	Igual que (1) cuando se usa para valores de posición. Y para valores de velocidad/aceleración (origen en el sistema): x: adelante (+), atrás (-) y: derecha (+), izquierda (-) z: abajo (+), arriba (-)
9	MAV_FRAME_BODY_OFFSET_NED	Coordenadas relativas a la posición del sistema (con origen a bordo). x: adelante (+), atrás (-) y: derecha (+), izquierda (-) z: abajo (+), arriba (-)

Segundo el TYPE_MASK, que es una trama de 12 bits que indica que dimensiones del movimiento se ignoran, para el control de posición se usan los valores de POSITION_TARGET_TYPEMASK, los más comunes están descritos en la Tabla 2.13.

Tabla 2.13. Valores del parámetro TYPE_MASK para posición [46].

Valor en decimal	**Descripción**
1	Ignora la posición en x: 0b000000000001 / 1 (decimal)
8	Ignora la velocidad vx: 0b000000001000 / 8 (decimal)
64	Ignora la aceleración ax: 0b000001000000 / 64 (decimal)
1024	Ignora yaw: 0b010000000000 / 1024 (decimal)
3576	Usa posición: 0b110111111000 / 3576 (decimal)
3527	Usa velocidad: 0b110111000111 / 3527 (decimal)
3128	Usa aceleración: 0b110000111000/ 3128 (decimal)
3520	Usa posición + velocidad: 0b110111000000/ 3520 (decimal)
3072	Ignora yaw y su velocidad: 0b110000000000/ 3072 (decimal)
2559	Usa yaw: 0b100111111111 / 2559 (decimal)
1535	Usa la velocidad en yaw: 0b010111111111/ 1535 (decimal)

Y para realizar movimientos de orientación, se usa el SET_ATTITUDE_TARGET que puede modificar el yaw, pitch y roll, así como sus respectivas velocidades. En la Tabla 2.14 se muestran los valores de ATTITUDE_TARGET_TYPEMASK más comunes, y que para este caso se trata de una trama de 8 bits.

Tabla 2.14. Valores del parámetro TYPE_MASK para orientación [46].

Valor en decimal	Descripción
1	Ignora la velocidad de roll: 0b00000001 / 1 (decimal)
2	Ignora la velocidad de pitch: 0b00000010 / 2 (decimal)
4	Ignora la velocidad yaw: 0b00000100 / 4 (decimal)
64	Ignora el acelerador: 0b01000000 / 64 (decimal)
128	Ignora la postura: 0b10000000 / 128 (decimal)
7	Usar velocidades de roll, pitch y yaw: 0b0000111 / 7 (decimal)

Para implementar los comandos de movimiento en el prototipo auxiliar se usan MAV_FRAME locales con origen en el sistema, esto para facilitar la configuración de los mensajes y evitar introducir errores de precisión al usar coordenadas globales. Además, se deben configurar como mensajes simples según se explicó en la Sección 2.5.2. Un ejemplo para mover el sistema o UAV dos metros al norte desde su posición actual, con control de posición es el siguiente:

```
# SET_POSITION_TARGET_LOCAL_NED con MAV_FRAME_LOCAL_OFFSET_NED
the_connection.mav.send(mavutil.mavlink.MAVLink_set_position_target_local_ned
        _message(10,the_connection.target_system,the_connection.target_compon
        ent, mavutil.mavlink.MAV_FRAME_LOCAL_OFFSET_NED, int(0b110111111000),
        2, 0, 0, 0, 0, 0, 0, 0, 0, 0))
```

Otro ejemplo puede ser, mover el UAV cinco metros a la derecha desde su posición actual, con control de velocidad, que se implementa de la siguiente forma:

```
# SET_POSITION_TARGET_LOCAL_NED con MAV_FRAME_BODY_OFFSET_NED
the_connection.mav.send(mavutil.mavlink.MAVLink_set_position_target_local_ned
        _message(10,the_connection.target_system,the_connection.target_compon
        ent, mavutil.mavlink.MAV_FRAME_BODY_OFFSET_NED, 3527, 0, 0, 0, 0, 5,
        0, 0, 0, 0, 0, 0))
```

Con la información presentada para enviar mensajes, recibir mensajes y ordenar movimientos usando el protocolo MAVLink, es posible guiar al UAV para que realice las funciones requeridas. Ahora se detallarán los subprocesos que conforman el algoritmo de guiado para el aterrizaje y las herramientas que hacen posible su implementación dentro de la Raspberry.

2.6 SISTEMA DE VISIÓN ARTIFICAL

Este sistema que forma parte del algoritmo de guiado para el aterrizaje procesa los fotogramas capturados por la cámara NoIR v1 y consta de dos etapas. Primero una de detección y localización del helipuerto, y luego una de seguimiento o tracker que evita perderlo de vista.

El helipuerto está representado por un código QR que, al estar en el campo de visión de la cámara, es decodificado y reconocido. En la Figura 2.8. se observa el código QR que se usa, es de tipo texto y contiene la palabra HELIPAD para identificarlo.

Figura 2.8. Código QR que representa el helipuerto (Elaborado empleando grcode-monkey.com).

Antes de detallar cada una de las etapas del sistema de visión artificial, se realiza una descripción del proceso de calibración de la cámara NoIR v1 para corregir su distorsión con el fin de evitar posibles problemas durante el tratamiento de imagen.

2.6.1 CALIBRACIÓN DE LA CÁMARA

Cualquier cámara que integre sistemas de visión artificial debe ser calibrada, su lente puede presentar deformaciones en la imagen que no aparecen en la realidad. Las deformaciones aparecen debido a dos tipos de distorsión. La distorsión radial, como se observa en la Figura 2.9., provoca el desplazamiento de los puntos de imagen hacia afuera o hacia adentro del centro de la imagen, lo que se aprecia como la deformación de líneas rectas en líneas curvas.

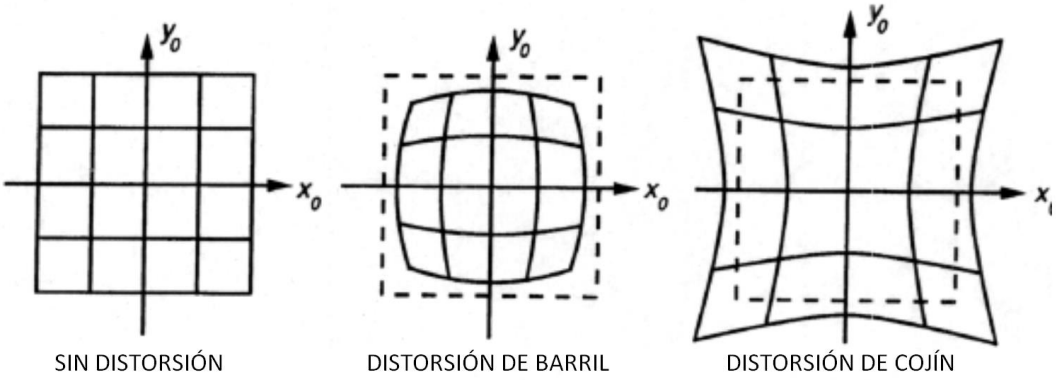


Figura 2.9. Tipos de distorsión radial [47].

La distorsión radial se resuelve con las siguientes expresiones:

$$x_{corregido} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{corregido} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (2.1)$$

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$$

Donde:

$k_1, k_2, k_3 =$ coeficientes de distorsión radial

$x_0, y_o =$ origen del eje óptico

$x, y =$ pares distorsionados

$r =$ distancia del par distorsionado al origen del eje óptico

Mientras que la distorsión tangencial, sucede cuando el punto principal del lente está descentrado por lo que las imágenes que se toman no están alineadas paralelamente al plano de la imagen. Esto provoca que áreas de la imagen se vean más cercanas de lo que deberían [47]. Su corrección se realiza con las expresiones:

$$x_{corregido} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$

$$y_{corregido} = y + [2p_2 xy + p_1(r^2 + 2y^2)] \quad (2.2)$$

Donde:

$p_1, p_2 =$ coeficientes de distorsión tangencial

Adicionalmente, para la calibrar una cámara se necesita conocer sus parámetros que incluyen la distancia focal (f_x, f_y), centros ópticos (c_x, c_y), entre otros. Para obtener estos parámetros y los coeficientes de distorsión radiales y tangenciales de la cámara NoIR v1, se usa un tablero de ajedrez de 9x6 del que se toman 34 patrones de prueba, de los que se obtienen los siguientes parámetros:

$$Matriz_{cámara} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 724.9468 & 0 & 294.0785 \\ 0 & 726.5749 & 224.9038 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

$$Coeff_{distorsión} = [k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3]$$
$$= [-0.49049 \quad 0.57357 \quad -0.00123 \quad -0.0004 \quad -0.99174]$$

Se puede revisar el repositorio de [48] que dispone de archivos para capturar los patrones de prueba y para obtener los parámetros de la cámara. Luego, con OpenCV se usa el método cv2.undistort() para calibrar la imagen, este admite como atributos la matriz de cámara y los coeficientes de distorsión como se muestra:

```
# CALIBRACIÓN DE LA CÁMARA
dst=cv2.undistort(img, mtx, dist)
```

Se consigue así la imagen calibrada `dst`, que elimina la distorsión de la imagen `img` capturada por la cámara. En la Figura 2.10. se puede apreciar la calibración de la cámara NoIR v1 donde se corrige la distorsión radial.

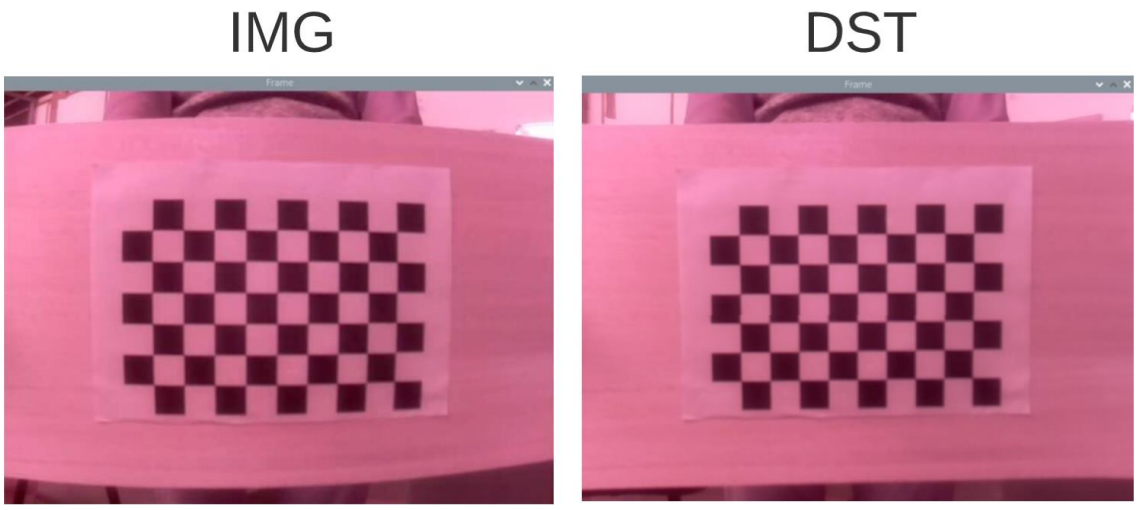


Figura 2.10. Ejemplo de calibración de la cámara NoIR v1.

Con la cámara calibrada, es posible implementar el sistema de visión artificial de forma más confiable. Para este caso un sistema con dos etapas que permite identificar, localizar y seguir el helipuerto donde aterriza el UAV de prueba.

2.6.2 DETECCIÓN Y LOCALIZACIÓN

La primera etapa detecta el helipuerto, como se mencionó en la descripción del sistema de visión artificial, está representado por un código QR. La ventaja de usar este tipo de código es que su lectura y procesamiento es rápido, además que por su estructura permite localizarlo fácilmente dentro de la imagen.

Los códigos QR pueden transmitir información de distinto tipo, son códigos 2D para lectura de alta velocidad. Están compuestos por diferentes marcas como: los módulos que son los elementos más pequeños (cuadrados negros o blancos), patrones de detección, patrones de posición, entre otros [49]. Su distribución para la versión 1 de códigos QR se puede observar en la Figura 2.11.

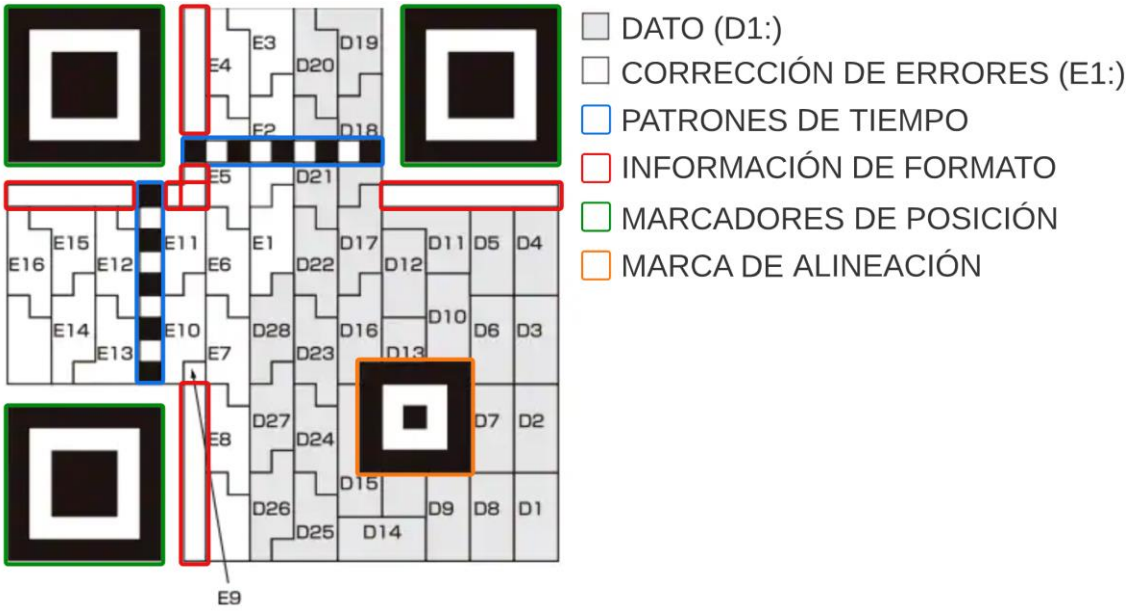


Figura 2.11. Estructura de un código QR (versión 1) [Fuente propia].

Para su lectura usando la cámara NoIR, se usa la biblioteca pyzbar escrita en Python. Esta biblioteca permite leer códigos de barras unidimensionales y códigos QR [50]. Su función es decodificarlos y localizarlos. Como se observa en la Figura 2.12. su localización se define con parámetros rectangulares que contiene el código: origen (x,y), ancho y altura.

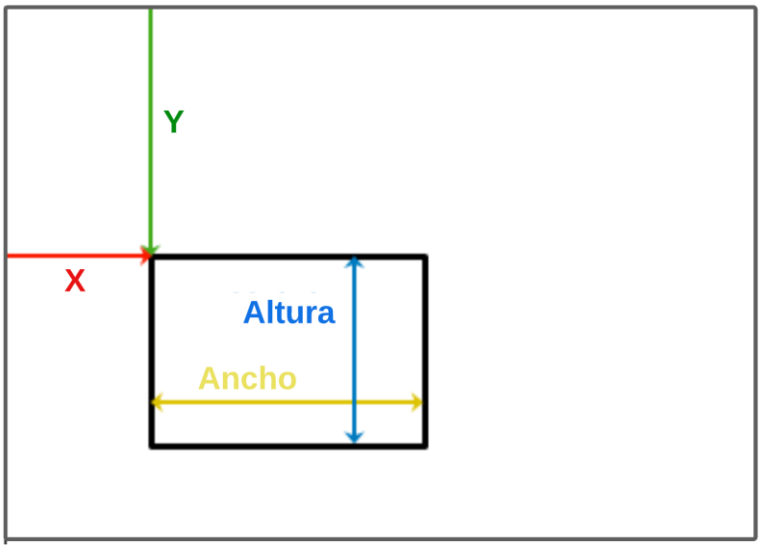


Figura 2.12. Parámetros de ubicación dentro de la imagen [Fuente propia].

Una vez identificado y localizado el código QR que representa el helipuerto, su seguimiento no puede depender de su lectura continua. Algún reflejo de luz o perturbaciones que afecten su decodificación pueden interrumpir esta maniobra. Por esta razón se incluye la siguiente etapa que evita perder de vista la región que abarca el código.

2.6.3 SEGUIDOR DE OBJETOS

Un seguidor de objetos conocido también como Tracker, permite seguir un área específica de la imagen en los siguientes fotogramas. La localización del código QR sirve para definir la región de interés (ROI) definida por un cuadro limitador o Bounding Box (BB) que inicializa este sistema. El resultado de la ejecución del Tracker es la actualización del BB en los fotogramas siguientes y se representa con los parámetros de ubicación (x, y, ancho, altura). Esto se usa para realizar un guiado de persecución, donde el movimiento del UAV se controla enviando mensajes como los descritos en la Sección 2.5.4 con sus parámetros establecidos en función de las coordenadas del centro del BB. Como se observa en la Figura 2.13., para hacer coincidir las coordenadas de posición de la cámara con las percibidas por el UAV, se debe cambiar la orientación del eje óptico para que coincida con el eje x e y del autopiloto.

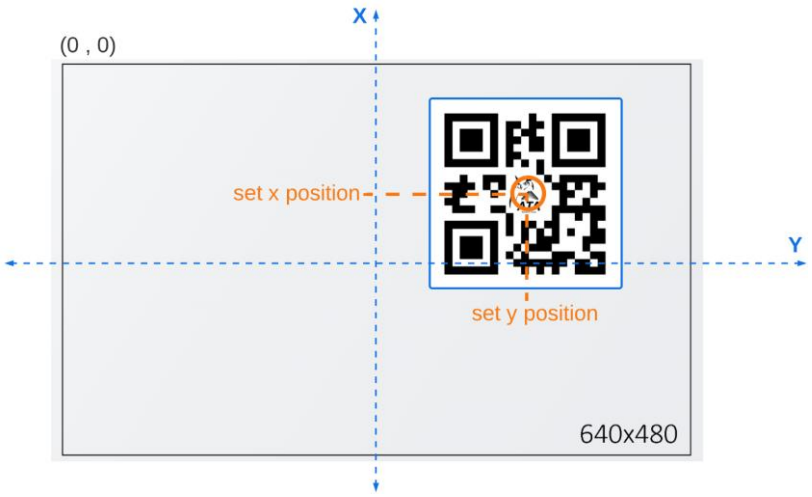


Figura 2.13. Eje óptico coincidente con el eje local del UAV durante el Tracker ("set x position" y "set y position" hacen referencia a los mensajes de movimiento para ejecutar el guiado de persecución) [Fuente propia].

Para implementar el Tracker en Python, OpenCV tiene 8 tipos de seguidores de objetos, sus principales características se presentan en la Tabla 2.15., mismos que se pueden revisar a detalle en [51].

Tabla 2.15. Seguidores de objetos en OpenCV.

Tipo	Características
BOOSTING	Se basa en AdaBoost, algoritmo que usa un detector de rostros basado en cascada HAAR (clasificador que debe entrenarse con ejemplos positivos y negativos).
MIL	Similar al BOOSTING pero este genera ejemplos potencialmente positivos para su entrenamiento.
KCF	Usa filtros de correlación Kernel. Aprovecha propiedades matemáticas para que el rastreo sea más rápido y preciso.
TLD	Seguimiento, aprendizaje y detección. Sigue al objeto cuadro a cuadro y corrige si es necesario.
MEDIANFLOW	Rastrea objetos en fotogramas futuros como pasados. Puede detectar fallas de seguimiento y selecciona trayectorias más confiables.
GOTURN	Único seguidor basado en una red neuronal convolucional.
MOSSE	Usa una correlación adaptativa, resiste a variaciones de iluminación, pose y deformaciones.
CSRT	Usa un filtro de correlación discriminativa con confiabilidad espacial y de canal. Seguimiento mejorado de regiones u objetos.

Para este caso se usa un seguidor de tipo CSRT que brinda una mayor precisión para el seguimiento de objetos, puede funcionar bien con fps (fotogramas por segundo) bajos y asegura la ampliación y la localización de la región a seguir [51].

En la Figura 2.14. se puede observar el diagrama de implementación del Tracker en Python, incluye la detección del código QR como fase previa a su inicialización. Es importante que primero se cree el seguidor (`cv2.TrackerCSRT_create()`) antes de inicializarlo (`tracker.init()`). Una vez inicializado el Tracker, las imágenes que lee la cámara ya no pasan por la primera etapa del sistema, automáticamente el seguidor las procesa (`tracker.update(frame)`) y se actualiza el BB con la nueva posición del área que está siguiendo, en este caso la región que contiene el código QR.

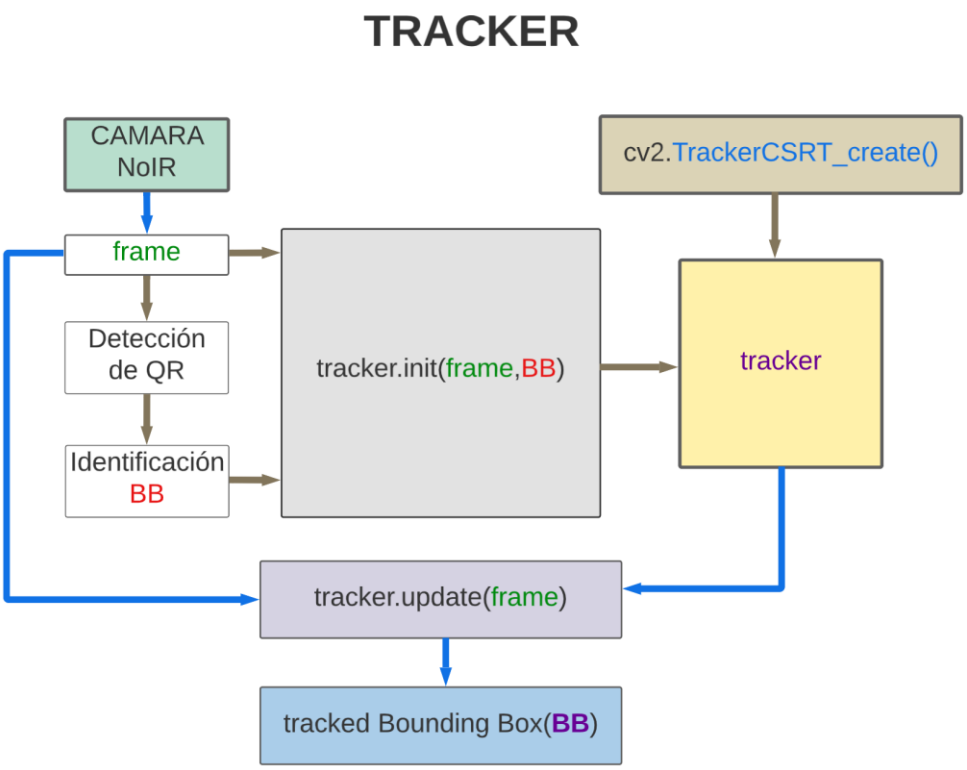


Figura 2.14. Diagrama de implementación del Tracker [Fuente propia].

El sistema de visión artificial tiene la principal función de detectar y centrarse con el helipuerto. De darse el caso de no detectar el helipuerto, se continua con el proceso TEST ZONA. Proceso que usa el sensor de distancia LIDAR Lite v3 para verificar que tan seguro es aterrizar en esa zona.

2.7 TEST DE ZONA DE ATERRIZAJE

Cuando no se tiene previsto el lugar de aterrizaje o cuando el comando LANDING se ejecuta en puntos intermedios de la misión, es necesario aterrizar sin contar con el helipuerto. En estas ocasiones se verifica que la zona sea segura, para lo cual se utiliza el sensor distancia LIDAR Lite v3 para realizar diferentes mediciones que indican la altura real del cuadricóptero. Con esta información es posible indicar si es seguro o no aterrizar en la ubicación actual.

Para su implementación en Python, se debe leer el bus I2C para identificar el sensor LIDAR Lite v3 y acceder a sus registros para obtener la medida de distancia. Con la biblioteca adafruit_lidarlite [52], se crea un objeto que permite acceder directamente a la medida de distancia, y con la librería busio se puede controlar el cabezal GPIO de la Raspberry para leer el bus i2c:

```
# LECTURA LIDAR LITE V3
i2c=busio.I2C(board.SCL,board.SDA)
sensor=adafruit_lidarlite.LIDARLite(i2c)
```

La forma en la que se testea la zona es recolectar datos de distancia que permitan determinar que tan plana es la zona de aterrizaje. Para este caso, como se observa en la Figura 2.15., se tiene una trayectoria que consta de 7 movimientos, durante su ejecución se toman mediciones de distancia que se almacenan en un arreglo. Posterior a la maniobra se calcula la desviación estándar de los puntos y en base a su valor se decide si la zona es apropiada para aterrizar.

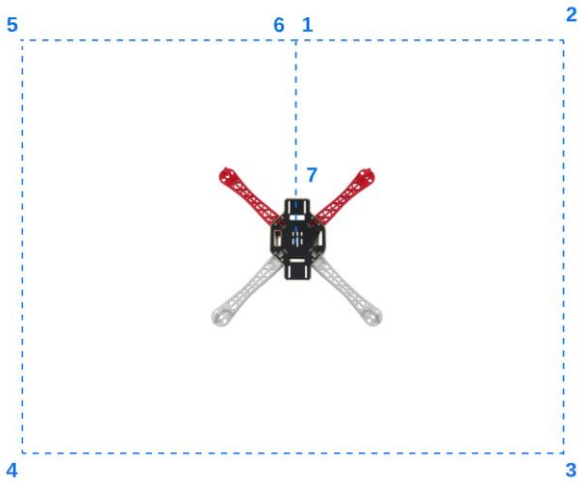


Figura 2.15. Patrón de movimiento para la medición de distancia [Fuente propia].

El movimiento del cuadricóptero considera coordenadas de MAV_FRAME locales para evitar introducir el error del GPS y facilitar la configuración de mensajes al tomar su posición relativa. Al concluir ya sea con esta maniobra o con el seguimiento del helipuerto, la fase final del algoritmo de guiado para el aterrizaje es el proceso de aterrizar, misma que se describe a continuación.

2.8 REFERENCIA DE VELOCIDAD DE ATERRIZAJE

Para el proceso de aterrizaje, el descenso se realiza controlando la velocidad en función de la altura obtenida del sensor LIDAR Lite v3. Primero se envía el comando para aterrizar (LAND) y el parámetro para controlar la velocidad de descenso es LAND_SPEED expresado en cm/s. Su valor está definido por la función lineal:

$$LAND\_SPEED = \frac{v_o}{(h_o - h_f)}\left(altitud - h_f\right) \tag{2.4}$$

Donde:

$LAND\_SPEED =$ referencia de velocidad de aterrizaje

$v_o =$ velocidad de aterrizaje inicial cuando se envía el comando LAND

$h_o =$ altura inicial cuando se envía el comando LAND

$hf =$ altura final (altura con el cuadricóptero en reposo)

$altitud =$ medida continua de altura

Además, cuando la medida del sensor LIDAR Lite v3 falla, por seguridad se establece la velocidad de descenso en 30 cm/s con el fin de asegurar que la maniobra culmine. Para modificar algún parámetro de vuelo se envía el mensaje PARAM_SET como se describe en la sección 2.5.2. Un ejemplo para el cambio del parámetro LAND_SPEED es:

```
# CAMBIO DE LAND_SPEED
the_connection.mav.param_set_send(the_connection.target_system,the_connection
        .target_component,b'LAND_SPEED',velocidad,mavutil.mavlink.MAV_PARAM_T
        YPE_UINT8)
```

Finalmente, es necesario deshabilitar el control de los motores con el comando COMPONENT_ARM_DISARM, para que el autopiloto detenga el registro de datos de vuelo (archivos con el registro del estado de todos los parámetros de vuelo). Cada proceso que del algoritmo de guiado para el aterrizaje se somete a prueba de forma independiente, luego se integran y se ejecutan en conjunto. Las pruebas realizadas junto a sus resultados se describen en el siguiente Capítulo.

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 PRUEBAS Y RESULTADOS

Para evidenciar el correcto funcionamiento del prototipo auxiliar, se realizan *pruebas individuales* para cada uno de los procesos del algoritmo de guiado previo a su ejecución completa. Cabe mencionar que dichas pruebas se realizan en un ambiente controlado, en este caso en la cancha principal de la EPN. En las siguientes secciones se detallan cada una de estas pruebas junto a sus resultados, que se comparan y analizan usando los Data Logs del controlador de vuelo y el registro de datos que se incluye en la Raspberry para un post procesamiento. Los Data Logs son los archivos que genera el controlador de vuelo con los datos almacenados durante la misión y se pueden analizar en: https://plot.ardupilot.org/#/ .

El registro de datos que se incluye en la programación de la Raspberry genera archivos que contienen los dataset (conjuntos de datos) de forma tabular y de fácil lectura. Estos archivos se escriben de forma automática a medida que transcurre la maniobra.

3.1.1 ALTITUD SEGURA

Estas pruebas corresponden a la verificación de la altura empleando el sensor LIDAR Lite v3. Se realizan cuatro pruebas en donde, se procesan los datos registrados por la Raspberry Pi 4 y el archivo generado por el controlador de vuelo. En la Figura 3.1. se muestran los datos de la prueba 1 registrados del sensor de distancia y los datos obtenidos del mensaje LOCAL_POSITION_NED (ID=32) denominados como *altitud_imu*, que obtiene sus registros con la información de los sensores GPS e IMU del controlador de vuelo.

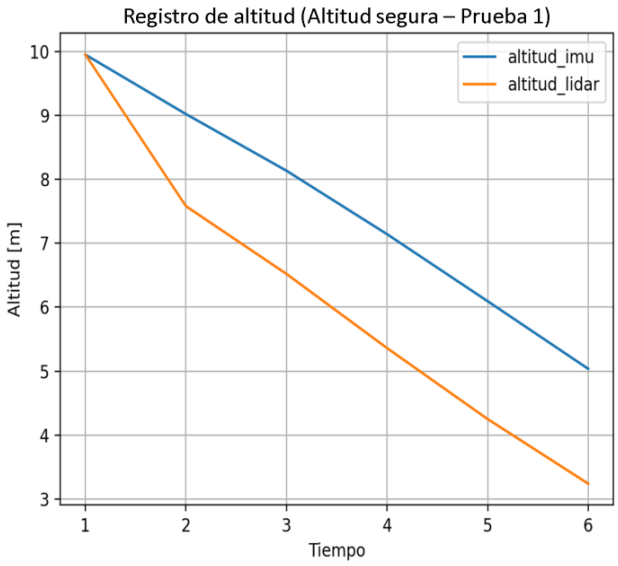


Figura 3.1. Registro de altitud durante la prueba 1 – altitud segura.

Además, del archivo de vuelo del controlador Pixhawk 2.1 se puede obtener la altitud relativa registrada durante la maniobra. En la Figura 3.2. se muestra los datos de altitud para la prueba 1 registrados por el barómetro (BARO(0).Alt [m]) y por el parámetro POS.RelHomeAlt [m], este último del mensaje POS del archivo de vuelo que contiene la estimación más confiable de posición del UAV debido a que sus datos provienen de un sistema AHRS. Sistema de referencia de postura y rumbo por sus siglas en inglés que utiliza la mayor cantidad de sensores del controlador para rechazar errores de medición con el empleo de filtros EKF (Flitros Kalman), mismos que estiman la posición, velocidad y orientación angular [53].

Respecto a la Figura 3.2. y a las similares presentadas en este capítulo, es necesario mencionar que, este tipo de gráficas se capturan del procesamiento del archivo de vuelo obtenidas a través del enlace que se compartió en la Sección 3.1 y por este motivo tienen un mismo formato. Además, el color en el fondo de la gráfica varía de forma aleatoria, sirve para diferenciar los modos de vuelo que se establecen durante la misión.

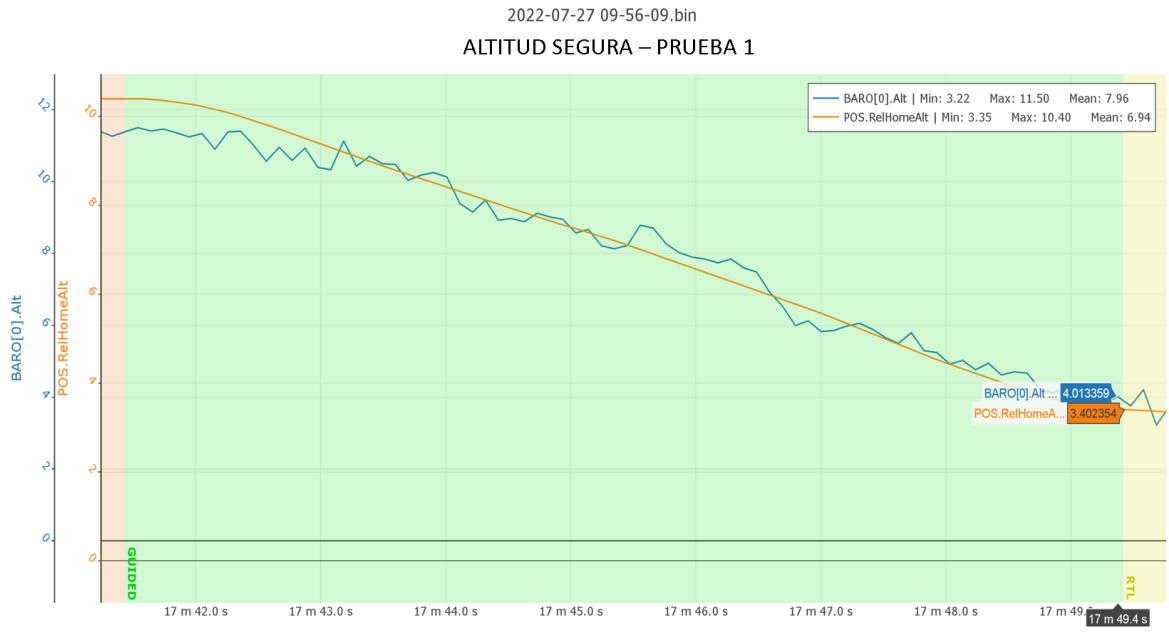


Figura 3.2. Registro de altitud del archivo de vuelo durante la prueba 1 – Altitud segura.

Los resultados a detalle para estas pruebas se pueden revisar en el ANEXO III y se encuentran resumidos en la Tabla 3.1., que hace énfasis en las mediciones al inicio y al final de la maniobra. Es importante analizar los datos de la prueba 4, al ser la altitud inicial inferior a la altitud de seguridad (3 metros para todas las pruebas de Altitud segura), no se tiene variación de altitud y su valor se considera seguro por lo que inmediatamente finaliza este proceso.

Tabla 3.1. Variación de altitud – Altitud segura.

Nro. Prueba	Altitud Inicial [m]			Altitud Final [m]		
	IMU	LIDAR Lite v3	DataLog POS.RelHomeAlt	IMU	LIDAR Lite v3	DataLog POS.RelHomeAlt
1	9.95	9.95	10.40	5.04	3.24	3.40
2	6.94	7.00	7.60	4.36	3.05	3.68
3	7.69	6.58	6.82	3.75	2.85	3.59
4	3.86	2.69	2.40	3.86	2.69	2.40

Con los datos de altitud inicial y final obtenida por las diferentes alternativas, es posible determinar los errores relativos entre ellos. En la Tabla 3.2. se presentan este tipo de errores en comparación con el parámetro POS.RelHomeAlt y expresados en porcentaje, con la intención de evaluar que tan confiable resulta emplear un sensor de distancia. Se usa este parámetro como referencia debido a su confiabilidad al ser obtenido de un sistema AHRS como se mencionó anteriormente.

Tabla 3.2. Errores relativos en comparación con el registro de vuelo.

Nro. Prueba	Altitud Inicial		Altitud Final	
	altitud_imu	LIDAR Lite v3	altitud_imu	LIDAR Lite v3
1	4.33 %	4.33 %	48.23 %	4.71 %
2	8.68 %	7.89 %	18.48 %	17.12 %
3	12.75 %	3.52%	4.46 %	20.61 %
4	60.83 %	12.08 %	60.83 %	12.08 %

Como se presenta, los datos registrados *altitud_imu* poseen los errores más relevantes siendo el mayor de 60.83% en la prueba 1, mientras que el error más reducido es del LIDAR Lite v3 de 3.52% en la prueba 3. Esto evidencia la baja precisión que tiene el GPS e IMU para estimar coordenadas de posición a escalas reducidas, lo que es una de las principales razones para emplear el sensor LIDAR Lite v3.

Luego, para las pruebas donde la altitud inicial es superior a la de seguridad, se comparan en la Tabla 3.3. la medición de la altitud final del sensor de distancia con la altitud segura. Los errores al llegar a la posición final no afectan significativamente los procesos posteriores.

Tabla 3.3. Errores relativos en comparación con la altitud segura.

Nro. Prueba	Altitud segura = 3 m	
	LIDAR Lite v3 [m]	Error relativo [%]
1	3.24	8.0
2	3.05	1.6
3	2.85	5.0

4	**No aplica:** Al tener una altura inicial inferior a la altura de seguridad no se necesita hacer una corrección de posición sobre el eje vertical.

3.1.2 SISTEMA DE VISIÓN ARTIFICIAL

Para constatar el seguimiento que realiza el cuadricóptero hasta centrarse con el helipuerto, se establece una altura arbitraria desde la que es posible visualizarlo y ejecutar la maniobra correspondiente, este procedimiento se realiza tres veces. Los resultados de estas pruebas se pueden revisar a detalle en el ANEXO III en donde se presentan las trayectorias seguidas por el cuadricóptero.

En la Figura 3.3. correspondiente a la prueba 1, se muestra el resultado de la trayectoria que sigue el cuadricóptero en el plano xy (Po – posición inicial y Pf – posición final) durante el seguimiento del helipuerto. También se presenta la altitud registrada durante la ejecución del proceso.

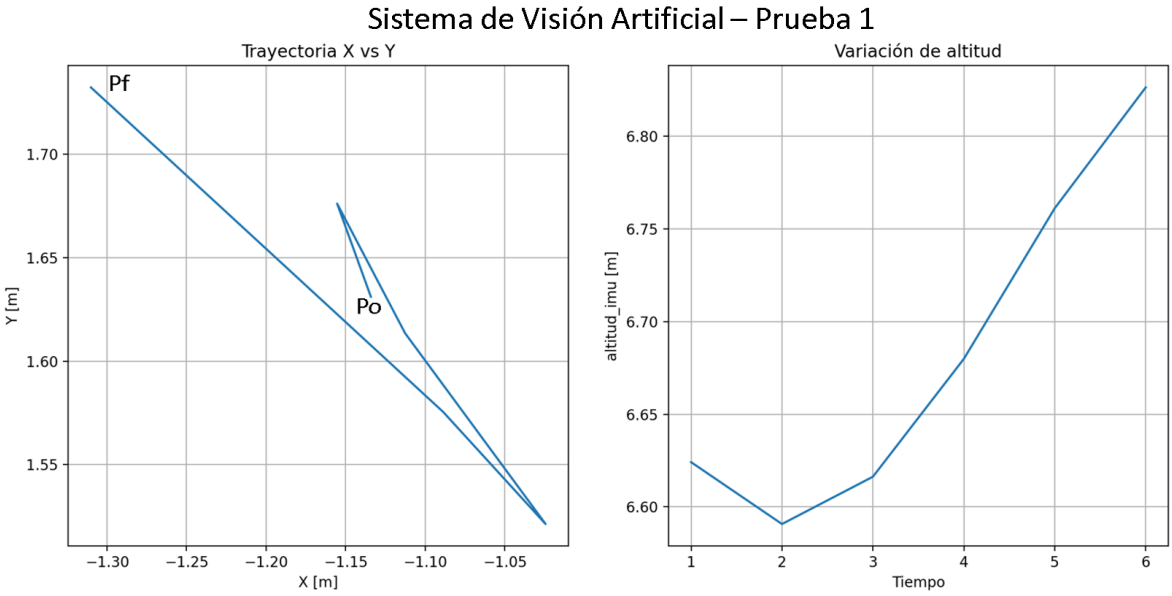


Figura 3.3. Trayectorias durante la prueba 1 – Sistema de Visión Artificial.

Luego de realizar las tres pruebas, se verifica que el sistema funciona, pues una vez centrado el helipuerto con el eje óptico de la cámara NoIR v1, se da por finalizada la maniobra correspondiente a este proceso. En la Figura 3.4. se puede observar la ejecución del Tracker para la prueba 3, que solo puede iniciarse una vez detectado el código QR, y muestra el seguimiento del helipuerto. Se presentan estos resultados porque durante la prueba 3 se procesaron el mayor número de imágenes.

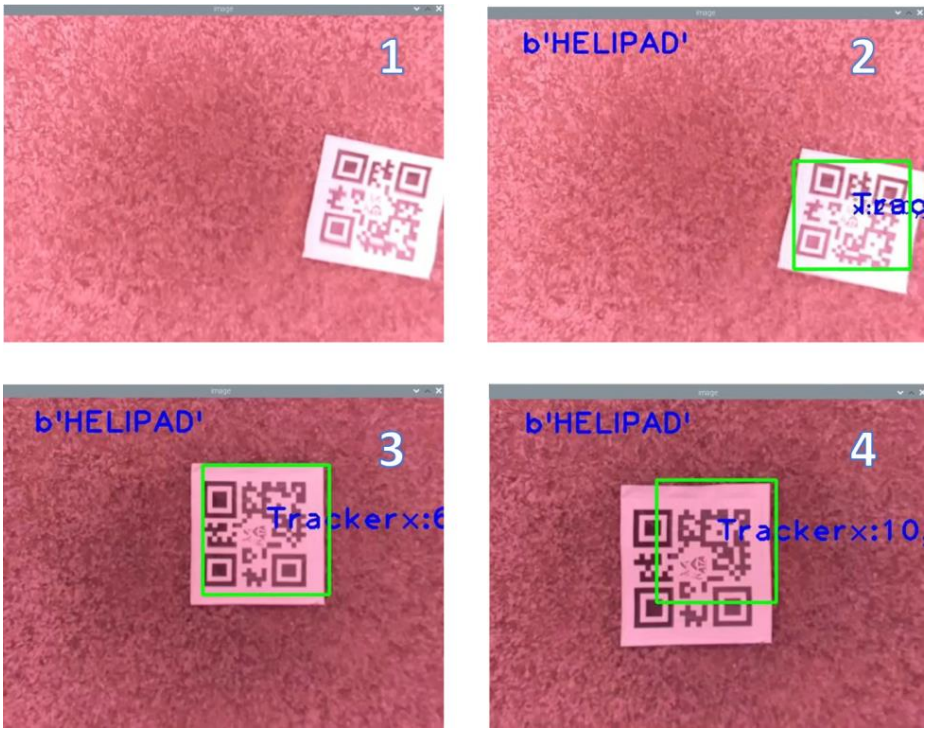


Figura 3.4. Seguimiento del helipuerto durante la prueba 3 (secuencia: 1,2,3,4)– Sistema de Visión Artificial.

Adicionalmente, en la Tabla 3.4. se resume la variación de altura que se registró durante las pruebas. Aunque el movimiento programado por el sistema de visión artificial establece puntos de referencia en el plano xy, la variación de altitud que se observa fue producto de perturbaciones externas al cuadricóptero como, por ejemplo, la influencia de corrientes de viento o el desplazamiento mismo sobre el eje xy.

Tabla 3.4. Variación de altitud – Sistema de Visión Artificial.

Nro. Prueba	Altitud durante la Etapa 2 [m]		
	Máxima [m]	Mínima [m]	Diferencia [m]
1	6.83	6.59	0.24
2	5.23	5.12	0.11
3	4.76	4.44	0.32

3.1.3 TEST ZONA

Esta sección corresponde a la verificación del terreno con la ayuda del sensor LIDAR Lite v3. Para esto se realizan tres pruebas en las que se comprueba la trayectoria que sigue el cuadricóptero y se analiza la variación de altura, esto con el fin de determinar el valor aceptable de desviación estándar para afirmar que el terreno es lo suficientemente plano para aterrizar. El ANEXO III que presenta a detalle los gráficos obtenidos a través de las pruebas.

En la Figura 3.5. se observa la trayectoria en el plano xy para la prueba 1, misma que muestra una trayectoria irregular, pero siguiendo la forma detallada en la Sección 2.7. Para estas pruebas se decide realizar la maniobra cuadrada con dimensiones de lado 2m x 2m, esto con el fin asegurar un aterrizaje dentro de la zona, considerando así el error de posición por parte del GPS e IMU observado en las pruebas de Altitud segura.

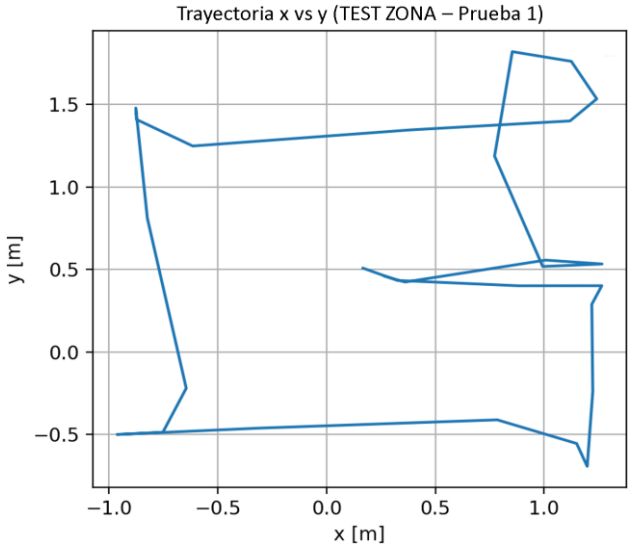


Figura 3.5. Trayectoria en el plano xy durante la prueba 1 – TEST ZONA.

Al igual que en las pruebas del Sistema de Visión Artificial, el cuadricóptero está propenso a sufrir variaciones en su posición vertical durante la ejecución de la maniobra. En la Figura 3.6. se puede observar la altitud registrada durante el proceso y en la que se manifiestan las variaciones antes mencionadas que en este caso se tiene una variación máxima de 0.15 m.

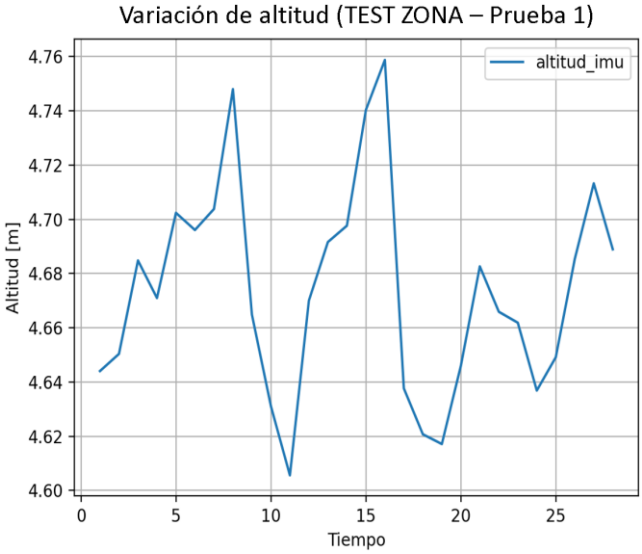


Figura 3.6. Variación de altitud durante la prueba 1 – TEST ZONA.

Con lo expuesto y para las tres pruebas, el cuadricóptero realiza la maniobra de forma irregular, pero siguiendo la forma prestablecida. Además, en la Tabla 3.5. se resumen las desviaciones estándar de los registros de altitud para cada prueba, esto con el fin de determinar un valor de desviación estándar que tolere las variaciones de altura provocadas por las perturbaciones. La desviación de 0.1554 m resulta ser la mayor registrada en comparación con las 3 pruebas realizadas y se puede considerar como la adecuada para indica un terreno plano y seguro para aterrizar.

Tabla 3.5. Desviación estándar del registro de altitud – TEST ZONA.

Nro. Prueba	STD [m]
1	0.0388
2	0.1554
3	0.0699

3.1.4 ATERRIZAJE CONTROLADO

Estas pruebas corresponden al proceso final del aterrizaje guiado, tanto para el caso en el que se localiza e identifica el helipuerto como para el caso en donde se debe realizar el proceso TEST ZONA. Se realiza un descenso controlado modificando el parámetro LAND_SPEED del controlador de vuelo en función de la altitud medida por el sensor LIDAR Lite v3. Durante maniobras previas se identifica que el cuadricóptero a velocidades de aterrizaje inferiores a 20 cm/s no se desplaza verticalmente de forma continua, y en ocasiones mantiene la altura. Por esta razón, durante el descenso también se verifica la velocidad resultante de la Ecuación (2.4) y se establece que, la velocidad sea constante e igual a 20 cm/s cuando el cálculo de velocidad sea inferior a dicho valor.

De igual forma, los resultados a detalle de las pruebas se los puede revisar en el ANEXO III. En la Figura 3.7. se observa la variación de altitud registrada por el sensor de distancia y por *altitud_imu* para la prueba 1, se observa el error persistente de aproximadamente 1 metro y del que se trató en las pruebas de Altitud segura.

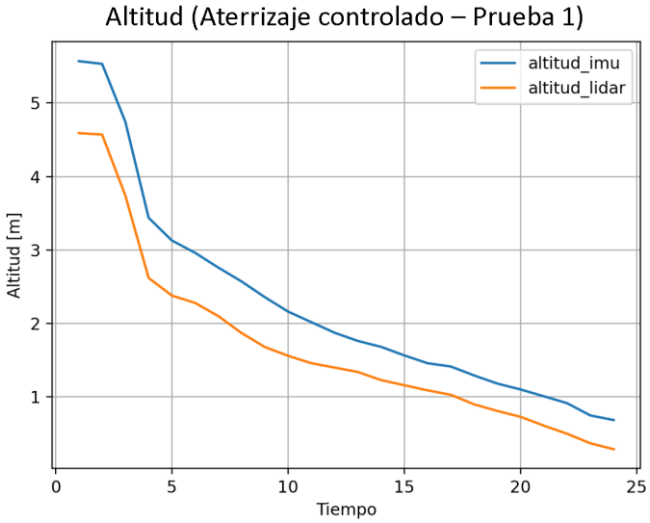


Figura 3.7. Registro de altitud durante la prueba 1 – Aterrizaje controlado.

También se registra la altitud medida por el sensor de distancia vs la velocidad en el eje z obtenida del mensaje LOCAL_POSITION_NED como se observa en la Figura 3.8. La relación no es lineal y presenta oscilaciones, esto debido a que la velocidad sobre un eje z también es influenciada por perturbaciones provocadas por el ambiente.

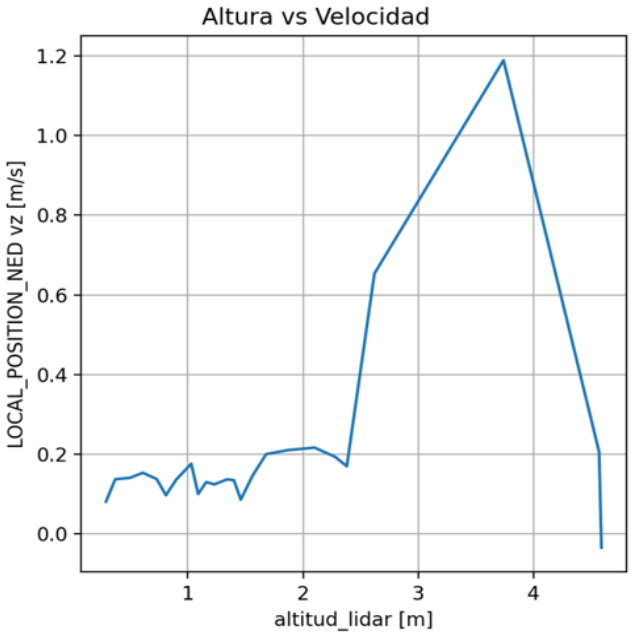


Figura 3.8. Registro de altura vs velocidad durante la prueba 1 – aterrizaje controlado.

Para verificar lo antes mencionado se procesan los datos del archivo de vuelo, en la Figura 3.9. se tiene la velocidad vertical deseada (PSCD.DVD [m/s]) en comparación con la velocidad vertical (PSCD.VD [m/s]), ambos parámetros pertenecen al mensaje PSCD que registra el control de posición de descenso. Aquí se observa que la velocidad deseada se

reduce gradualmente hasta establecerse en 0.2 m/s, sin embargo, en la velocidad medida se presentan las perturbaciones registradas anteriormente en la Figura 3.8. Luego, tras completar el aterrizaje satisfactoriamente, los motores se detienen por lo que se aprecia un valor nulo en la velocidad.

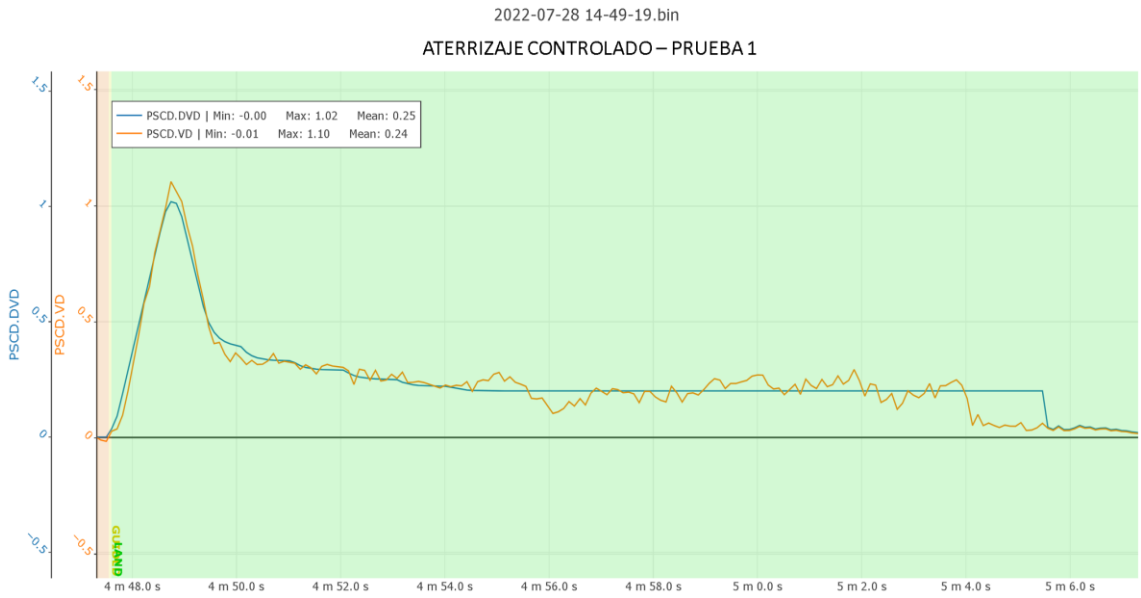


Figura 3.9. Registro de velocidad vertical (PSCD [m/s]) durante la prueba 1 – Aterrizaje controlado.

3.1.5 PRUEBAS COMPLETAS

Estas pruebas corresponden a la ejecución del algoritmo de guiado al aterrizaje que integra todos los procesos ya puestos a prueba y que además considera los resultados de las sus pruebas. Se establece una altitud segura de 4 metros suficiente para detectar el helipuerto en el Sistema de Visión Artificial, las dimensiones 2m x 2m de la región cuadrada del proceso TEST ZONA se conservan al igual que el límite inferior de velocidad en 20 cm/s para del Aterrizaje controlado. También, se integra la verificación constante del estado de aterrizaje LANDED_STATE perteneciente al mensaje EXTENDED_SYS_STATE (ID=245). El inicio de la maniobra está condicionado al detectar el comando LAND (LANDED_STATE=4). De igual forma que en las pruebas descritas, los resultados de esta sección se encuentran detallados en el ANEXO III.

En la Figura 3.10. se presenta el registro de altitud durante la ejecución del algoritmo para la prueba 1, donde se puede observar que empieza descendiendo hasta el rango de altitud segura correspondiente a al proceso Altitud segura, posteriormente se mantiene la altitud durante la ejecución de los procesos de Sistema de Visión Artificial o TEST ZONA, y finalmente el descenso progresivo del Aterrizaje controlado.

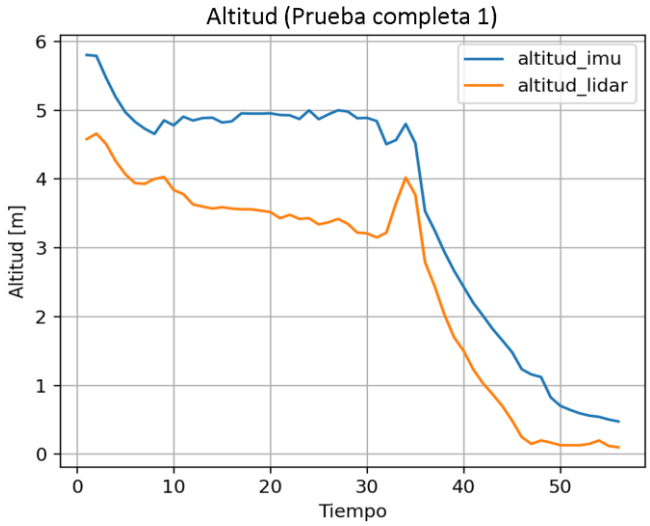


Figura 3.10. Altitud registrada – Prueba completa 1.

Para revisar que sucedió una vez finalizado el proceso Altitud segura, en la Figura 3.11 se observa la trayectoria seguida en el plano xy para la prueba 1, que muestra la ejecución del proceso TEST ZONA, lo que indica que el Sistema de Visión Artificial no detectó ningún helipuerto en las imágenes captadas por la cámara.

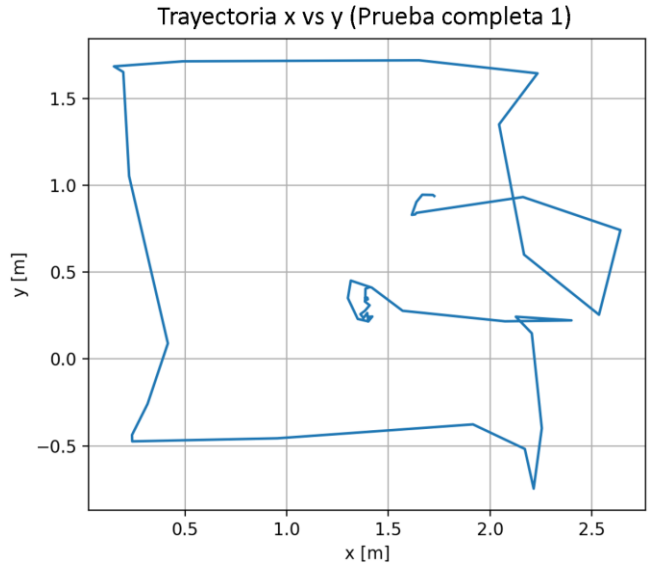


Figura 3.11. Trayectoria en el plano xy – Prueba completa 1.

Además, para el Aterrizaje controlado que finaliza la maniobra, se comprueba el cambio de modo de vuelo a LAND y el cambio de velocidad de aterrizaje en la Figura 3.12. que, muestra las gráficas obtenidas al procesar el archivo de vuelo.

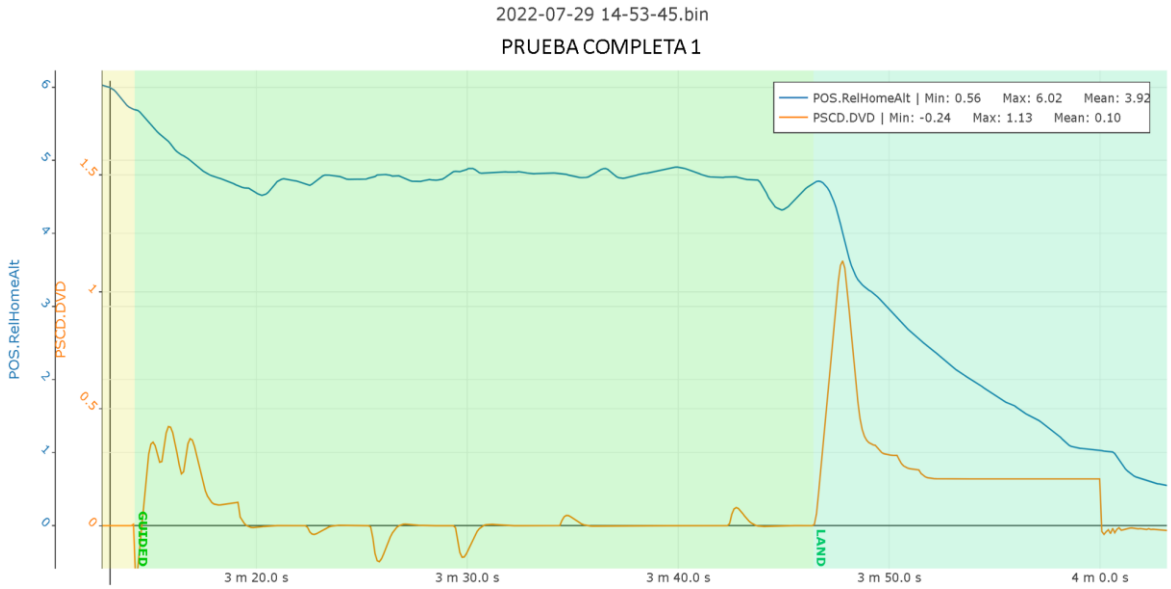


Figura 3.12. Altitud (POS.RelHomeAlt [m]) y velocidad vertical deseada (PSCD.DVD [m/s])– Prueba completa 1.

Por último, en la Figura 3.13 se presenta la trayectoria tridimensional del cuadricóptero durante el aterrizaje guiado, que resume la ejecución satisfactoria del algoritmo.

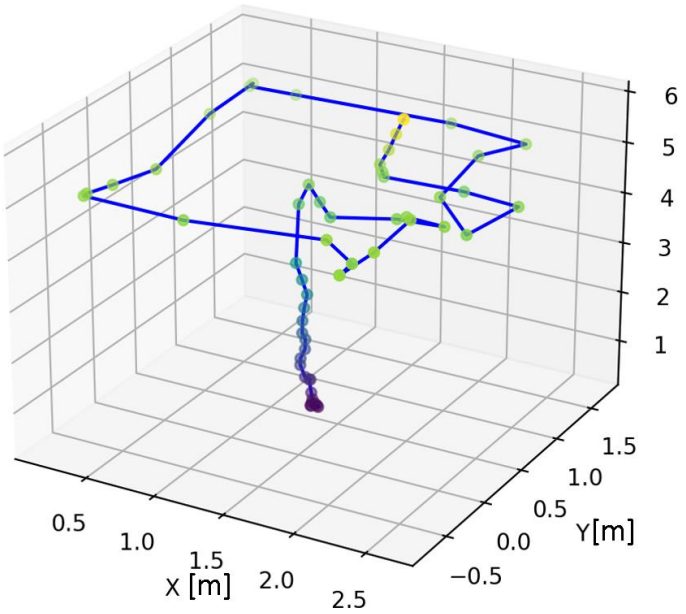


Figura 3.13. Trayectoria del cuadricóptero – Prueba completa 1.

Durante la ejecución de estas pruebas se intentó obtener una en la que el Sistema de Visión Artificial detecte al helipuerto, sin embargo, en ninguno de los intentos se consiguió que el QR sea detectado. El algoritmo respondió según su lógica e inicio el proceso TEST ZONA y después el Aterrizaje controlado, completando así la función de guiar el aterrizaje.

La dificultad para maniobrar el cuadricóptero, ubicarlo en una posición adecuada para visualizar el helipuerto y el retardo en el procesamiento de imágenes son las principales razones para no conseguirla. Aun así, el Sistema de Visión Artificial funciona y tiene la capacidad de realizar la maniobra, según las pruebas y resultados de la Sección 3.1.2, siempre y cuando el helipuerto sea legible en la primera imagen procesada. Para evidenciar una prueba completa en la que actúe el Sistema de Visión Artificial, se puede revisar el ANEXO III en el que se agrega este caso.

3.1.6 ATERRIZAJE SIN APOYO DEL PROTOTIPO

Durante las pruebas anteriores, en las que se verifica la funcionalidad de los procesos de forma individual, excepto en el Aterrizaje controlado, se cambia el modo de vuelo a RTL (Return to Launch) para finalizar las pruebas. Este modo de vuelo hace que el UAV regrese y aterrice en la posición inicial (Home position) y es propio del controlador de vuelo, por lo que el prototipo auxiliar no interfiere en su proceso.

En la Figura 3.14 se observa la trayectoria de altitud (POS.RelHomeAlt [m]) y velocidad deseada (PSCD.DVD [m/s]) del cuadricóptero. Mientras está en RTL (similar en todas las pruebas en las que se implementó), se puede observar como su velocidad de aterrizaje es constante e igual a 0.5 m/s, mientras que en el modo de vuelo LAND, que es el empleado por el algoritmo de aterrizaje, establece su velocidad de aterrizaje en 0.2 m/s después de reducir su valor gradualmente. Se debe recordar que este último es modificado constantemente en función del sensor de distancia. También se observa que el modo de vuelo RTL presenta un aterrizaje más suave, por su bajo sobre impulso de 20% en el control de velocidad vertical, a comparación del modo de vuelo LAND (utilizado en el proceso Aterrizaje controlado), que presenta un sobre impulso elevado de aproximadamente 100 % en el control de velocidad vertical.

Sin embargo, el modo de vuelo RTL necesariamente hace regresar al UAV al Home position, una desventaja cuando se necesita aterrizar en alguna posición arbitraria o por emergencia, como, por ejemplo, un bajo nivel de batería o desorientación. Por esta razón se emplea el comando LAND que permite el aterrizaje en cualquier sitio, y gracias al apoyo del prototipo auxiliar dirige el aterrizaje controlando su velocidad en función de la altura. Además, al emplear el sensor de distancia, se consigue que el UAV tolere los desniveles del suelo que por sí solo es incapaz de estimar, pues sus datos son relativos únicamente a la su posición de origen.

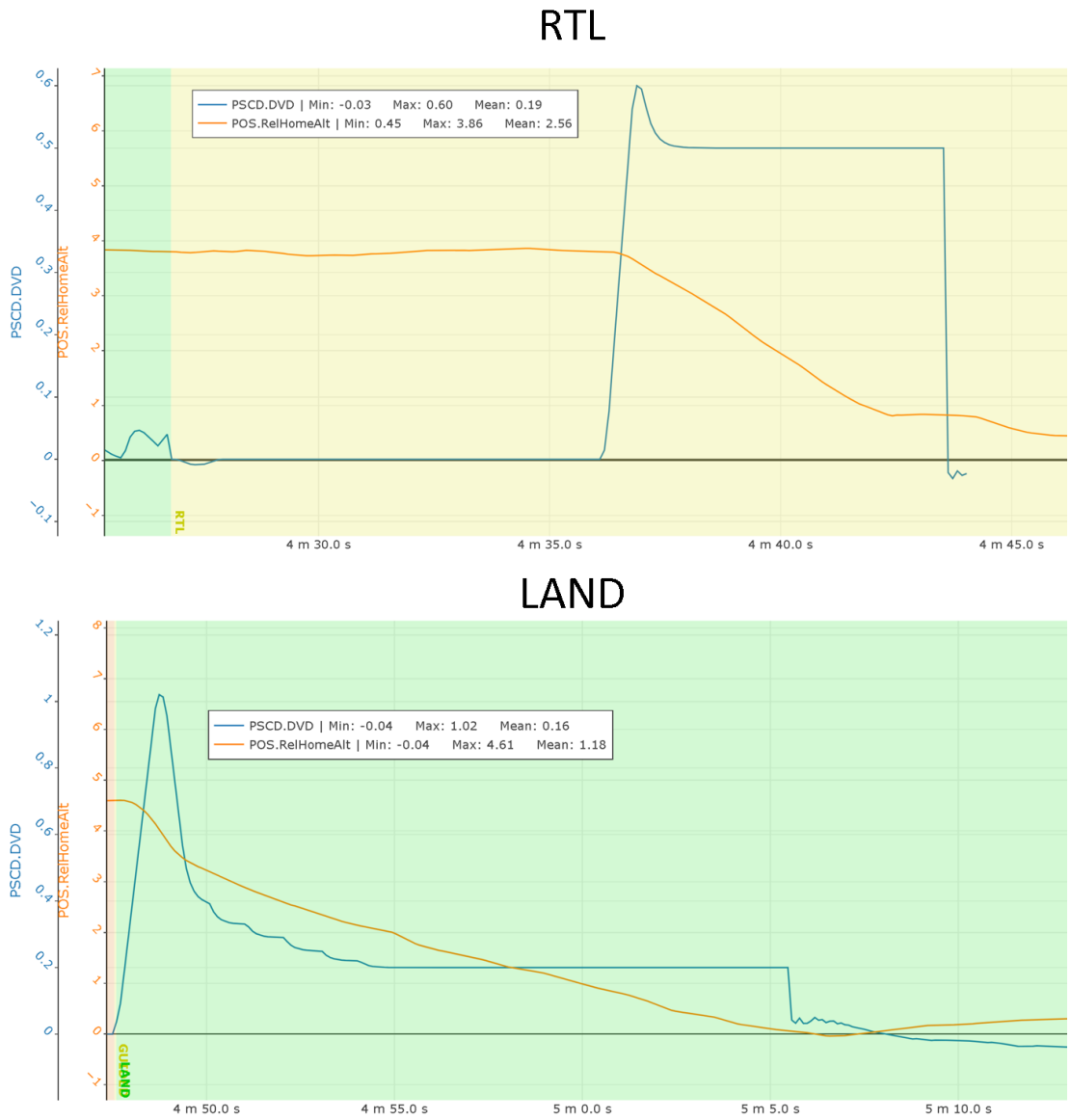


Figura 3.14. Aterrizaje durante los modos de vuelo RTL (sin apoyo del prototipo auxiliar) y LAND (con apoyo del prototipo auxiliar)

Adicionalmente, y para finalizar esta sección, el prototipo auxiliar cuenta con un manual de usuario que se encuentra en la carpeta compartida del ANEXO IV. Mismo que describe cómo debe ser el montaje del prototipo auxiliar a bordo del UAV, muestra el procedimiento para conectarse a la Raspberry y el procedimiento para verificar el estado del UAV empleando Mission Planner. También muestra las formas de ejecutar el algoritmo de guiado para el aterrizaje y como descargar los datos registrados por la Raspberry y por el controlador de vuelo.

3.2 CONCLUSIONES

El aterrizaje autónomo en UAVs es un tema muy abordado en la actualidad, los problemas relacionados a la falta de información del entorno en el que se desenvuelven, hace necesario incluirlo dentro de cualquier sistema aéreo no tripulado. Esto con el fin de evitar pérdidas materiales que pueden desembocar en un fallo total del UAV y de su controlador de vuelo.

Los sistemas auxiliares que guían la maniobra de aterrizaje son versátiles y pueden adaptarse de diferentes formas según el hardware que se utilice. El presente prototipo al contar con un sensor de distancia y una cámara permiten incluir, además de un sistema de visión artificial, una medición altitud, misma que sirve de referencia para la mayoría del algoritmo de guiado, especialmente en la etapa final del aterrizaje.

El trabajo multidisciplinario es primordial cuando se requiere obtener un producto confiable y con buenas prestaciones. El prototipo auxiliar del presente trabajo necesitó un espacio adecuado en el cuadricóptero de prueba para ubicar los sensores de forma que apunten hacia la superficie y estén lo más centrados posible. Gracias a la colaboración en el diseño mecánico, la plataforma de carga que se adaptó en el cuadricóptero es lo suficientemente rígida para soportar la carga y esfuerzo sin deformarse.

El protocolo de comunicación MAVLink hace posible que el prototipo auxiliar guíe la maniobra de aterrizaje implementando mensajes de fácil configuración. Estos mensajes permiten la transferencia bidireccional de parámetros de vuelo que dan a conocer la información del estado del UAV y también pueden modificarlo.

El envío de mensajes MAVLink hacia el UAV alteran instantáneamente su estado y no se remite automáticamente alguna información de su recepción, para esto es necesario constantemente solicitar el parámetro que indique si el mensaje se ejecutó. En el caso de mensajes de movimiento, además de solicitar si el mensaje o comando se ejecutó, se debe leer continuamente el parámetro de interés y compararlo con el deseado antes de enviar el siguiente mensaje, caso contrario el primero se interrumpe para darle prioridad al siguiente.

La implementación de sistemas de visión artificial requiere de un procesamiento cada vez mayor según su complejidad. La identificación, localización y seguimiento implementados en el prototipo auxiliar reducen la frecuencia de fotogramas procesados, lo que se manifiesta como un retardo en la captura de los mismos y puede ocasionar que no se ejecute el sistema de visión artificial, sin embargo, este retardo no impide que la maniobra se ejecute con éxito una vez detectado el helipuerto.

Los controladores de vuelo que reciben comandos de movimiento desde una estación en tierra están afectados continuamente por el error de posición del GPS. Aunque en aplicaciones de gran cobertura esto no representa un problema durante la navegación, las maniobras como el despegue y aterrizaje que requieren de posiciones especificas si pueden afectarse por este tipo de error.

Los datos de altitud obtenidos por mensajes MAVLink desde el cuadricóptero muestran un error persistente de aproximadamente 1 metro por encima de la medición del sensor LIDAR Lite v3. Durante la maniobra de aterrizaje, es necesario contar con una mejor precisión en cuanto a la altitud, pues al depender únicamente del controlador de vuelo, la diferencia puede provocar aterrizajes bruscos o accidentes.

Durante las pruebas realizadas se pudo evidenciar que cualquier cambio en el entorno puede afectar la posición del cuadricóptero y esto no depende únicamente de los mensajes que se envían o se reciben. El cuadricóptero es un sistema que continuamente es afectado por perturbaciones externas, aunque los mensajes de movimiento se enfoquen en una sola dirección y sentido, las demás componentes del movimiento también se verán afectados, para este caso se evidenció en mayor grado con la posición vertical.

3.3 RECOMENDACIONES

Se recomienda verificar el tipo del UAV que se va a guiar, existen variaciones de modelo a modelo en lo que se refiera al uso de comandos o mensajes comunes de MAVLink, así como variaciones en la descripción de los modos de vuelo disponibles para cada tipo de UAV.

El prototipo auxiliar está limitado a UAVs de tipo VTOL, pues el algoritmo diseñado respeta su característica más importante que es el despegue y aterrizaje vertical. Si se pretende implementar la lógica del algoritmo planteado en UAVs de tipo HTOL se debe considerar que el aterrizaje será horizontal, por lo que se recomienda usar una técnica de guiado tridimensional y no planar como la usada por el sistema de visión artificial en el presente TIC.

Si se desea implementar sistemas de visión artificial que introduzcan técnicas como: redes neuronales, detectores de objetos, entre otros. Se recomienda usar unidades de procesamiento con GPU (Graphic Proccessor Unity) que están diseñadas específicamente para procesar imágenes, lo que daría mejores resultados y evitaría problemas de retardos entre las lecturas de fotogramas.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] A. Barrientos *et al.*, "Vehículos aéreos no tripulados para uso civil. Tecnología y aplicaciones", Madrid: UPM.

[2] E. Santana Cruz, "Propuesta de sistema multi-UAV para aplicaciones de cobertura aérea", Barcelona: UAB, 2017.

[3] L. Garberoglio, "Diseño de un autopiloto para pequeños vehículos no tripulados", Buenos Aires: UTN, 2017.

[4] "Hardware (Drones & Drone Parts) | PX4 User Guide", *Docs.px4.io*, 2021. [Online]. Available: https://docs.px4.io/master/en/hardware/drone_parts.html. [Accessed: 22-Dec- 2021].

[5] "Introduction · MAVLink Developer Guide," *Mavlink.io*, 2021.[Online]. Available: https://mavlink.io/en/?fbclid=IwAR0TKuuyyAQv1TqFazqFvdCDolhDheN7A_A8xVoJD-rK6Dg9weT8El2GuDo [Accessed: 29-Nov-2021].

[6] V.K. Guevara Balarezo, "Diseño e implementación de un sistema de telemetría y video para vehículos aéreos no tripulados (UAVS)", 90 hojas, Quito : EPN, 2019.

[7] J.R Baldeón Osorio, "Automatización de un hexacóptero para despegue, aterrizaje y vuelo en un camino cerrado", 124 hojas, Quito : EPN, 2015.

[8] S.O. Cayo Guamushig, I.D. Changoluisa Caiza, "Integración y automatización de sistema de seguimiento de un UAV para establecer un enlace de comunicación con una estación de monitoreo en tierra", 164 hojas, Quito : EPN, 2018.

[9] U. E. Franke, "Civilian drones: Fixing an image problem?", en *ISN Blog. International Relations and Security Network.* Recuperado, vol. 5, 2015.

[10] G. Shingal, B. Bansod, L. Mathew, "Unmanned Aerial Vehicle classification, Applications ans challenges: A Review", *Preprints* 2018, 2018110601 (DOI: 10.20944/preprints201811.0601.v1).

[11] V. Chamola *et al.*, "A Comprehensive Review of Unmanned Aerial Vehicle Attacks and Neutralization Techniques", *Ad Hoc Networks,* vol. 111, 2021.

[12] "Classifications and Flight Mechanisms of UAVs", *Shahin Darvishpoor*, 2022. [Online]. Available: https://shahindarvishpoor.ir/2020/12/12/flight-mechanism-of-uavs/. [Accessed: 01- May- 2022].

[13] E. Nieto, F. Vaca De La Torre, "Desarrollo de un modelo matemático, cinemático y dinámico con la aplicación de software, para modificar el funcionamiento de un dron, para que este realice monitoreo automático", *RECIMUNDO*, 4(1(ESP)), 2020. 332-343. DOI:10.26820/recimundo/4.(1).esp.marzo.2020.332-343

[14] Alvika G, Sujit P.B, S. Saripalli. "A survey of autonomous landing techniques for UAVs", *International Conference on Unmanned Aircraft Systems*, 2014, DOI: 10.1109/ICUAS.2014.6842377.

[15] D. Bohorquez, "Sistema de visión por computador embebido para la detección y el seguimiento de objeto móvil desde un vehículo aéreo no tripulado (UAV) usando ROS", M.S. thesis, Bogotá, Colombia,2021.

[16] T. Layman, T. Fields, O. Yakimenko, "Evaluation of Proportional Navigation for Multirotor Pursuit", 2021, doi: 10.2514/6.2021-1813.

[17] N. A. Shneydor, "Missile guidance and pursuit: kinematics, dynamics and control", *Elsevier*, 1998, pp 47-67.

[18] "The history of Pixhawk", *Auterion.com*, 2022. [online]. Available: https://auterion.com/company/the-history-of-pixhawk/. [Accessed 13 May 2022].

[19] "Terminology | PX4 User Guide", *Docs.px4.io*, 2022. [Online]. Available: https://docs.px4.io/v1.12/en/contribute/notation.html. [Accessed: 13- May- 2022].

[20] "Controller Diagrams | PX4 User Guide", *Docs.px4.io*, 2022. [online]. Available: https://docs.px4.io/v1.12/en/flight_stack/controller_diagrams.html. [Accessed: 13- May- 2022].

[21] "MAVLink Basics — Dev documentation", *Ardupilot.org*, 2022. [Online]. Available: https://ardupilot.org/dev/docs/mavlink-basics.html#:~:text=MAVLink%20is%20a%20serial%20protocol,common.xml%20and%20ardupilot.xml. [Accessed: 15- May- 2022].

[22] "Introduction · MAVLink Developer Guide", *Mavlink.io*, 2022. [Online]. Available: https://mavlink.io/en/?q=. [Accessed: 15- May- 2022].

[23] A. Koubâa *et al.,* " Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey", en *CISTER-TR-190701,* vol. 4, 2019. DOI: 10.1109/ACCESS.2019.2924410

[24] G. Natarajan, “Ground control stations for unmanned air vehicles (Review Paper)”, *DSJ*, vol. 51, no. 3, pp. 229-237, Jan. 2002.

[25] "Mission Planner Overview — Mission Planner documentation", *Ardupilot.org*, 2022. [Online]. Available:https://ardupilot.org/planner/docs/mission-planner-overview.html. [Accessed: 16- May- 2022].

[26] "A2212 1000KV BLDC Brushless DC Motor", *Dzinotech.com*, 2022. [Online]. Available: https://dzinotech.com/product/a2212-1000kv-bldc-brushless-dc-motor/. [Accessed: 26- May- 2022].

[27] "F450 Frame dimensions", *Researchgate.net*, 2022. [Online]. Available: https://www.researchgate.net/figure/F450-Frame-dimensions_fig2_355023499. [Accessed: 26- May- 2022].

[28] "Hex Cube Black Flight Controller | PX4 User Guide", *Docs.px4.io*, 2022. [Online]. Available: https://docs.px4.io/v1.12/en/flight_controller/pixhawk-2.html. [Accessed: 26- May- 2022].

[29] "HEX/ProfiCNC Here2 GPS | PX4 User Guide", *Docs.px4.io*, 2022. [Online]. Available: https://docs.px4.io/v1.12/en/gps_compass/gps_hex_here2.html. [Accessed: 26- May- 2022].

[30] "RFD900+ Modem - RFDesign", *RFDesign*, 2022. [Online]. Available: http://rfdesign.com.au/products/rfd900-modem/. [Accessed: 26- May- 2022].

[31] "FlySky FS-iA10B 2.4Ghz AFHDS 10CH Receiver, PPM, Telemetry", *Flying Tech*, 2022. [Online]. Available: https://www.flyingtech.co.uk/electronics/flysky-fs-ia10b-24ghz-afhds-10ch-receiver-ppm-telemetry. [Accessed: 26- May- 2022].

[32] P. PMU, U. SL and U. SL, "Pixhawk 2 - Módulo de Alimentación PMU", *UNMANNED TECHNOLOGY SL*, 2022. [Online]. Available: https://rc-innovations.es/modulo-alimentacion-pixhawk2-original-sensor-consumo. [Accessed: 26- May- 2022].

[33] "Distance Sensors (Rangefinders) | PX4 User Guide", *Docs.px4.io*, 2022. [Online]. Available: https://docs.px4.io/v1.12/en/sensor/rangefinders.html. [Accessed: 25- May- 2022].

[34] "Companion Computer Peripherals | PX4 User Guide", *Docs.px4.io*, 2022. [Online]. Available: https://docs.px4.io/v1.12/en/peripherals/companion_computer_peripherals.html. [Accessed: 25- May- 2022].

[35] *Cdn.sparkfun.com*, 2022. [Online]. Available: https://cdn.sparkfun.com/datasheets/Dev/RaspberryPi/RPiCamMod2.pdf. [Accessed: 26- May- 2022].

[36] "Cámaras Raspberry: Cámara Raspberry Pi 5MP visión nocturna, RPi Camera (F)", *Didacticaselectronicas.com*, 2022. [Online]. Available: https://didacticaselectronicas.com/index.php/sistemas-de-desarrollo/raspberry/accesorios-raspberry/c%C3%A1maras-1/camara-raspberry-pi-5mp-visi%C3%B3n-nocturna-camaras-sensor-sensores-de-imagen-con-vision-nocturna-ov5647-para-raspberry-camara-waveshare-detail. [Accessed: 26- May- 2022].

[37] "OV7670 Camera Module: Datasheet, Specifications and Comparison", *Utmel.com*, 2022. [Online]. Available: https://www.utmel.com/components/ov7670-camera-module-datasheet-specifications-and-comparison?id=797. [Accessed: 26- May- 2022].

[38] "Arducam Mini 2MP Plus - OV2640 SPI Camera Module for Arduino UNO Mega2560 Board & Raspberry Pi Pico - Arducam", *Arducam*, 2022. [Online]. Available: https://www.arducam.com/product/arducam-2mp-spi-camera-b0067-arduino/. [Accessed: 26- May- 2022].

[39] "What is a Raspberry Pi?", *Opensource.com*, 2022. [Online]. Available: https://opensource.com/resources/raspberry-pi. [Accessed: 16- May- 2022].

[40] "Caracteristicas de la nueva Raspberry Pi 4 Model B", *pccomponentes.com*, 2022. [Online]. Available: https://www.pccomponentes.com/caracteristicas-raspberry-pi-4. [Accessed: 26- May- 2022].

[41] "Dialects · MAVLink Developer Guide", *Mavlink.io*, 2022. [Online]. Available: https://mavlink.io/en/messages/. [Accessed: 31-Mayo-2022].

[42] "Python (mavgen) · MAVLink Developer Guide", *Mavlink.io*, 2022. [Online]. Available: https://mavlink.io/en/mavgen_python/. [Accessed: 31- May- 2022].

[43] "Messages (common) · MAVLink Developer Guide", *Mavlink.io*, 2022. [Online]. Available: https://mavlink.io/en/messages/common.html. [Accessed: 31- May- 2022].

[44] "Pymavlink · GitBook", *Ardusub.com*, 2022. [Online]. Available: https://www.ardusub.com/developers/pymavlink.html. [Accessed: 01- Jun- 2022].

[45] "Flight Modes — Copter documentation", *Ardupilot.org*, 2022. [Online]. Available: https://ardupilot.org/copter/docs/flight-modes.html. [Accessed: 02- Jun- 2022].

[46] "Copter Commands in Guided Mode — Dev documentation", *Ardupilot.org*, 2022. [Online]. Available: https://ardupilot.org/dev/docs/copter-commands-in-guided-mode.html. [Accessed: 02- Jun- 2022].

[47] "Calibración de la cámara Opencv", 2022. [Online]. Available: https://unipython.com/calibracion-la-camara-opencv/. [Accessed: 06- Jun- 2022].

[48] "GitHub - EveryWhereLab/camera-calibration-using-opencv-python", *GitHub*, 2022. [Online]. Available: https://github.com/EveryWhereLab/camera-calibration-using-opencv-python. [Accessed: 06- Jun- 2022].

[49] "What is a QR code? | Basics of 2D codes | Barcode Information & Tips | KEYENCE America", *Keyence.com*, 2022. [Online]. Available: https://www.keyence.com/ss/products/auto_id/codereader/basic_2d/qr.jsp. [Accessed: 06- Jun- 2022].

[50] "pyzbar", PyPI, 2022. [Online]. Available: https://pypi.org/project/pyzbar/. [Accessed: 06- Jun- 2022].

[51] "Object Tracking using OpenCV (C++/Python)", *LearnOpenCV – OpenCV, PyTorch, Keras, Tensorflow examples and tutorials*, 2022. [Online]. Available: https://learnopencv.com/object-tracking-using-opencv-cpp-python/#:~:text=There%20are%208%20different%20trackers%20available%20in%20OpenCV,5%20trackers%20%E2%80%94%20BOOSTING%2C%20MIL%2C%20KCF%2C%20TLD%2C%20MEDIANFLOW. [Accessed: 08- Jun- 2022].

[52] "GitHub - adafruit/Adafruit_CircuitPython_LIDARLite: CircuitPython library for Garmin LIDARLite sensor", GitHub, 2022. [Online]. Available:

https://github.com/adafruit/Adafruit_CircuitPython_LIDARLite. [Accessed: 10- Jun- 2022].

[53] "Extended Kalman Filter (EKF) — Copter documentation", Ardupilot.org, 2022. [Online]. Available: https://ardupilot.org/copter/docs/common-apm-navigation-extended-kalman-filter-overview.html. [Accessed: 18- Aug- 2022].

5 ANEXOS

ANEXO I. Planos de las estructuras 3D

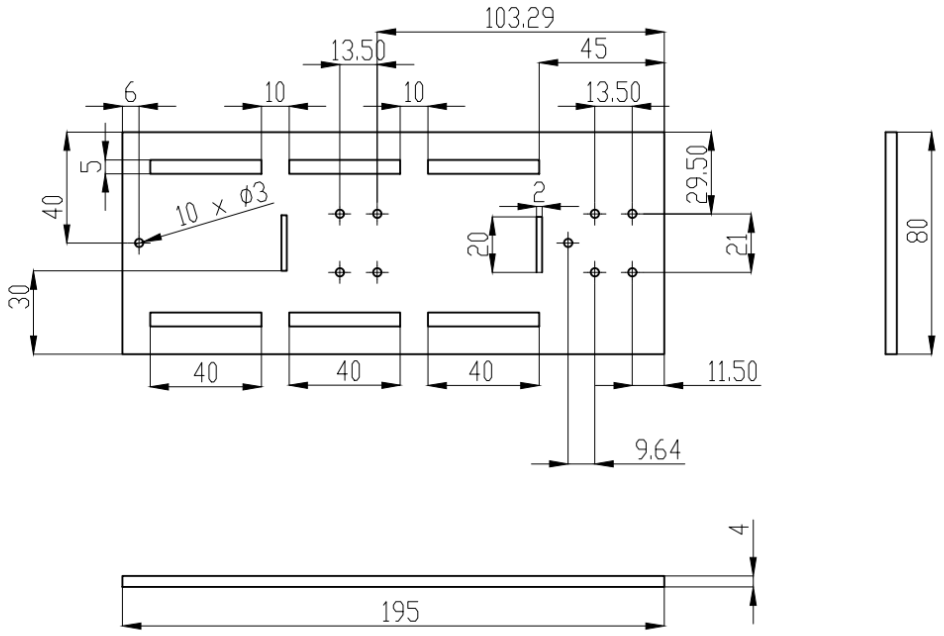
ANEXO II. Diagrama de conexión del prototipo auxiliar

ANEXO III. Pruebas de funcionamiento

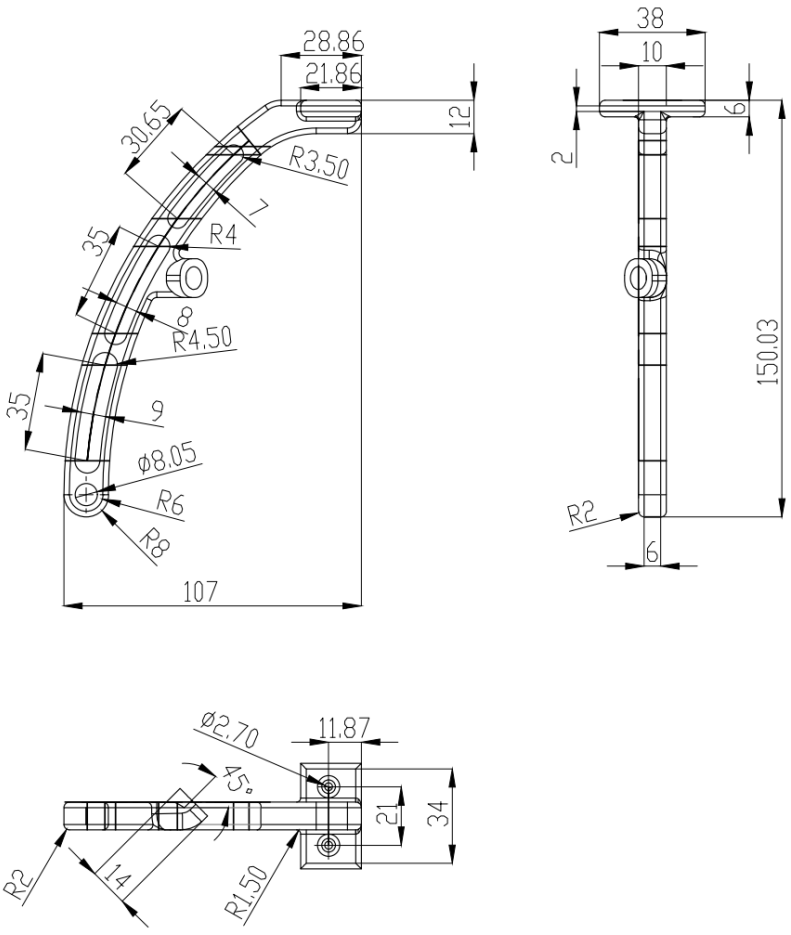
ANEXO IV. Enlaces

** Dentro de la carpeta compartida, cuyo enlace está en el último Anexo, se encuentra disponible el Manual de Usuario*

ANEXO I. Planos de las estructuras 3D

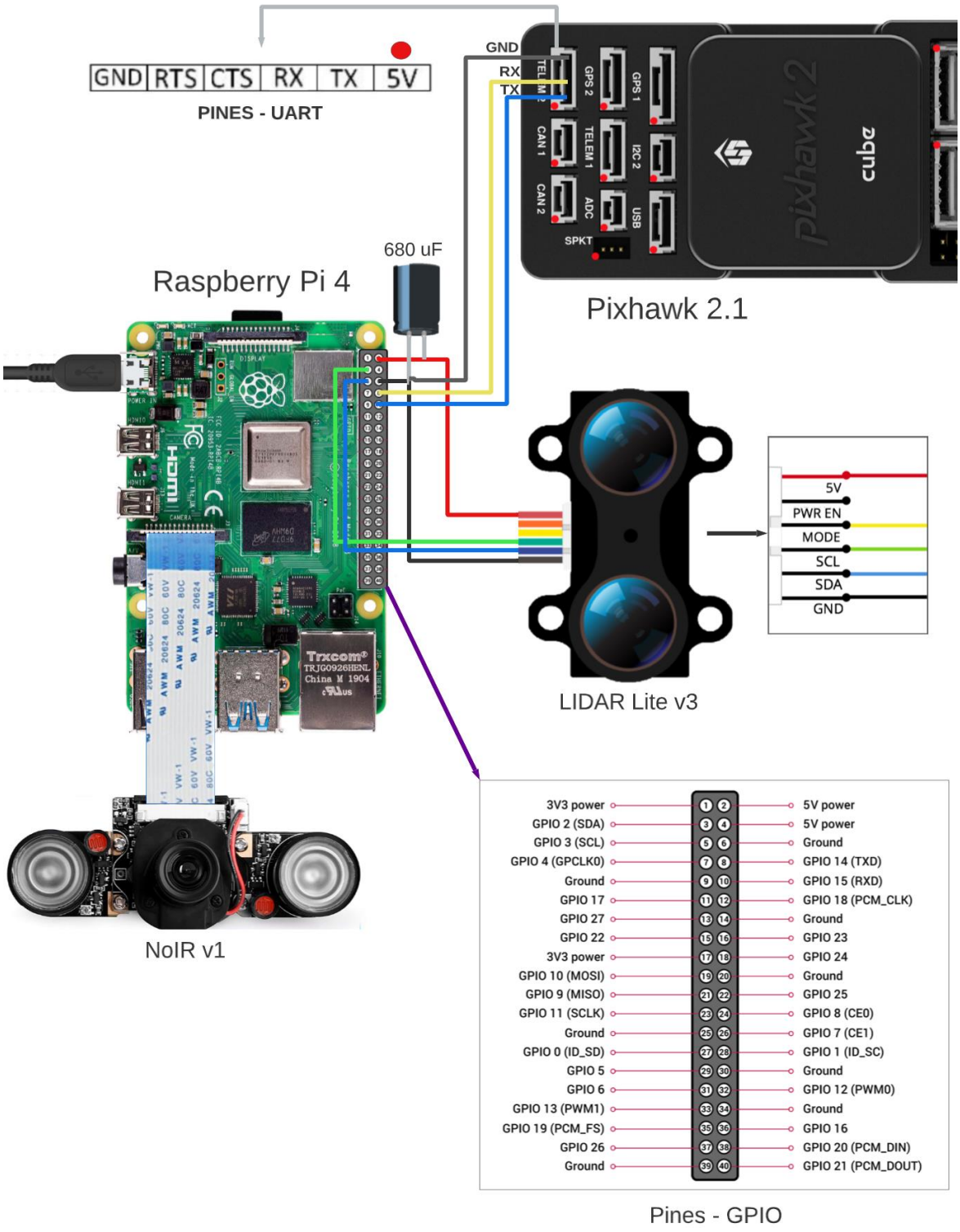


Trat. Térmico	Ninguno		E.P.N	FACULTAD DE INGENIERÍA MECÁNICA	
Recubrimiento	Ninguno				
Material: PLA		Tol. Gral: ±0.01	Escala: 1:2	DIB.	Edwin Amaguaña
				DIS.	Edwin Amaguaña
				REV.	N.A
Base Principal			Plano - 02	Fecha: 22/07/2022	



Trat. Térmico	Ninguno		E.P.N	FACULTAD DE INGENIERÍA MECÁNICA	
Recubrimiento	Ninguno				
Material: PLA		Tol. Gral: ±0.01	Escala: 1:2	DIB.	Edwin Amaguaña
				DIS.	Edwin Amaguaña
				REV.	N.A
Soporte			Plano - 01	Fecha: 22/07/2022	

ANEXO II. Diagrama de conexión del prototipo auxiliar



ANEXO III. Pruebas de funcionamiento

Aquí se presentan los resultados correspondientes a las pruebas de funcionamiento realizadas. Gráficas de trayectorias obtenidas del post procesamiento de los archivos de vuelo y dataset generados en cada una de las pruebas. Se puede revisar el ANEXO IV en donde se encuentran los enlaces que dirigen a la carpeta que contiene dichos datos.

ALTITUD SEGURA

Prueba 1:

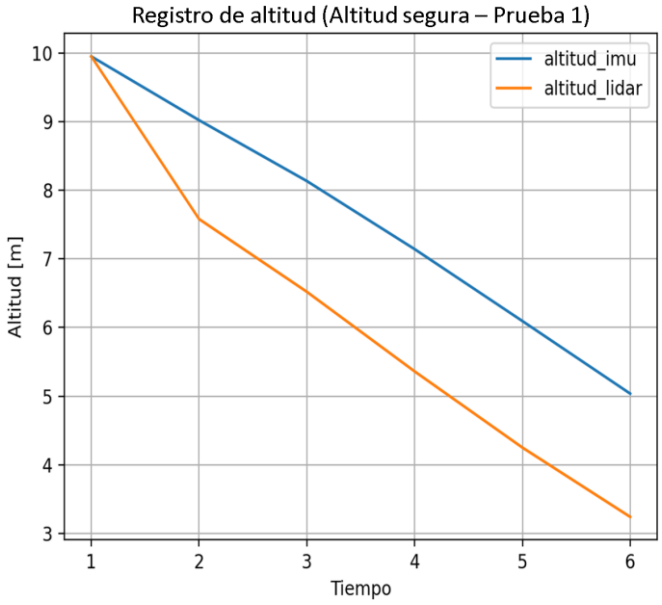


Figura III.1. Registro de altitud durante la prueba 1 – Altitud segura.

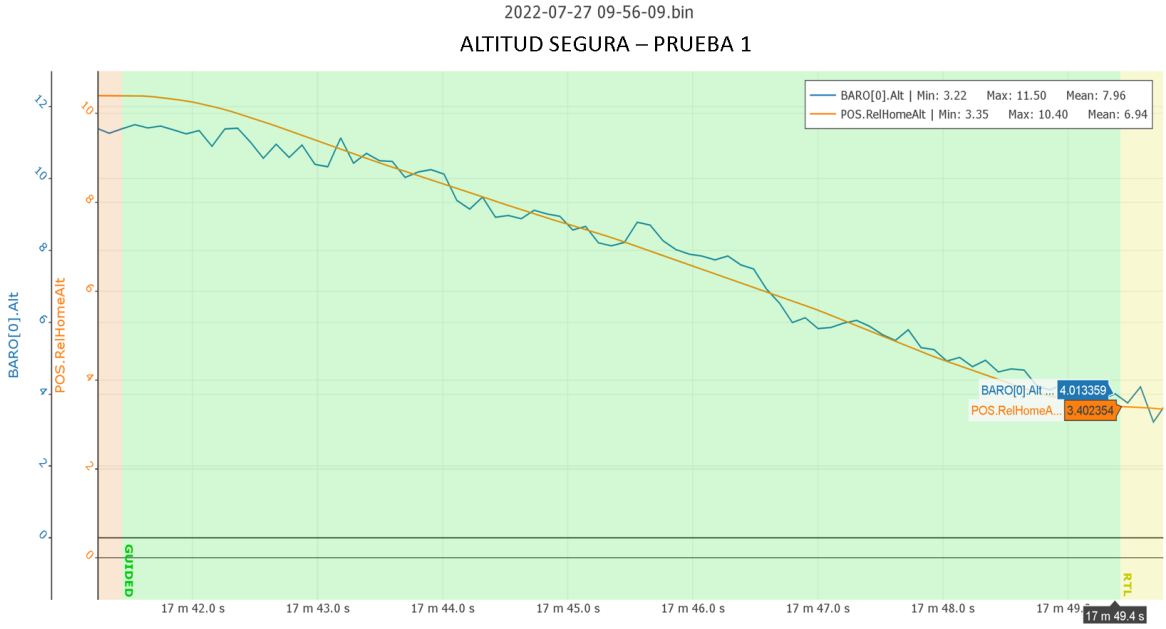


Figura III.2. Registro de altitud del archivo de vuelo durante la prueba 1 – Altitud segura.

Prueba 2:

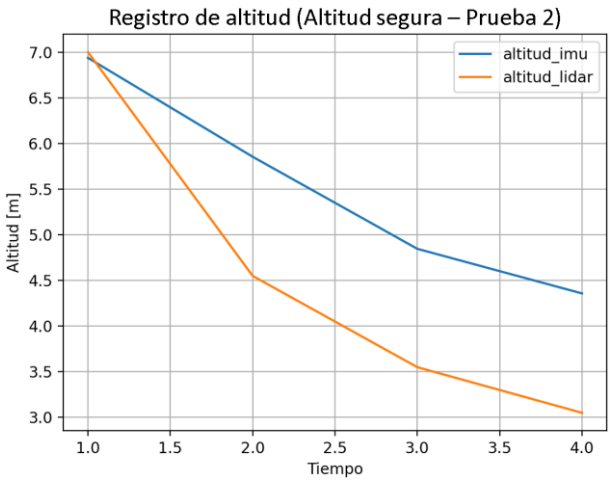


Figura III.3. Registro de altitud durante la prueba 2 – Altitud segura.

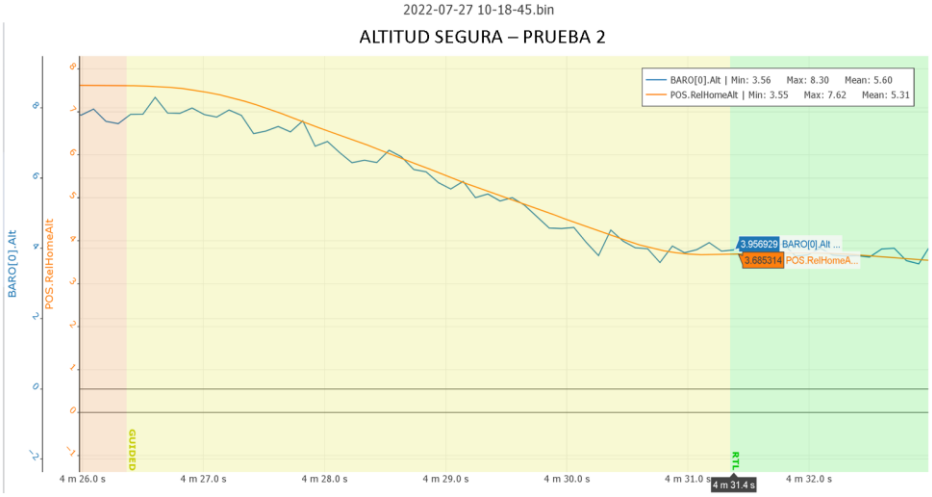


Figura III.4. Registro de altitud del archivo de vuelo durante la prueba 2 – Altitud segura.

Prueba 3:

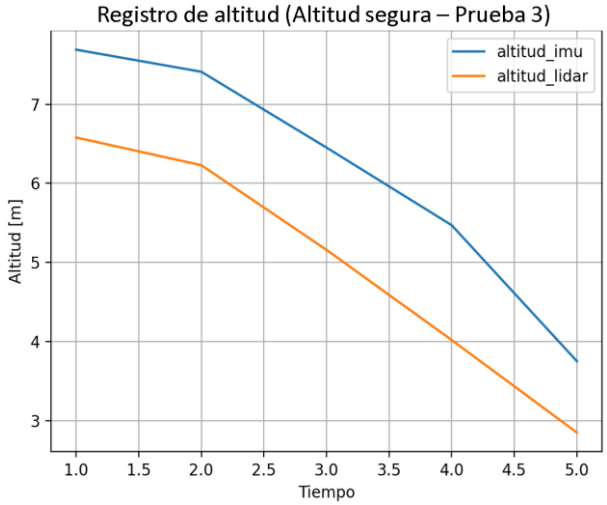


Figura III.5. Registro de altitud durante la prueba 3 – Altitud segura.

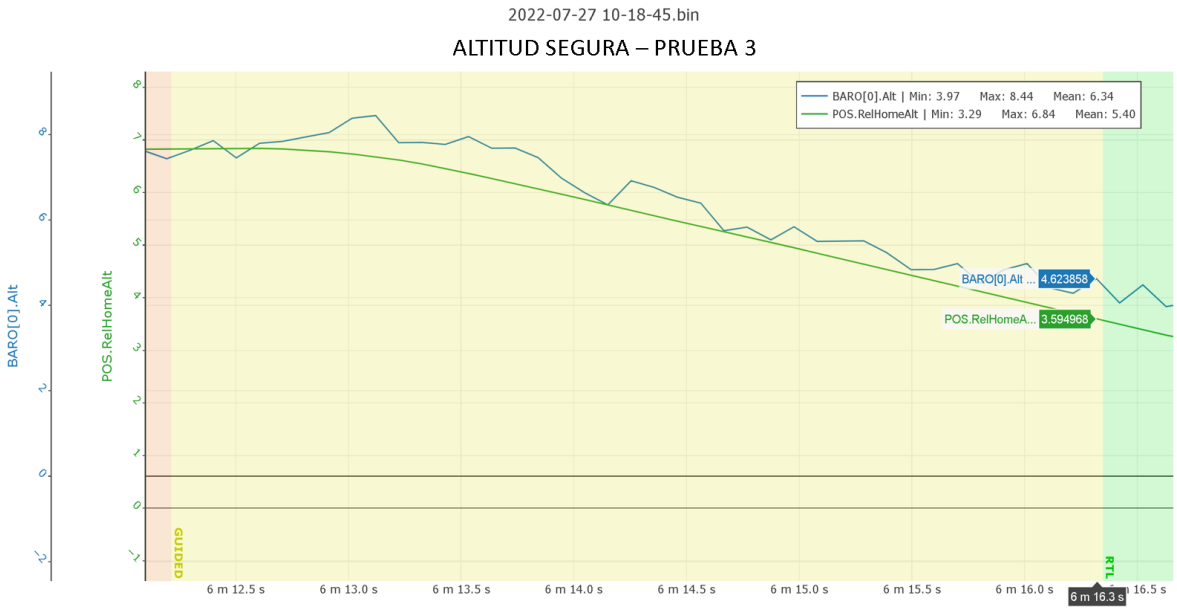


Figura III.6. Registro de altitud del archivo de vuelo durante la prueba 3 – Altitud segura.

Prueba 4:

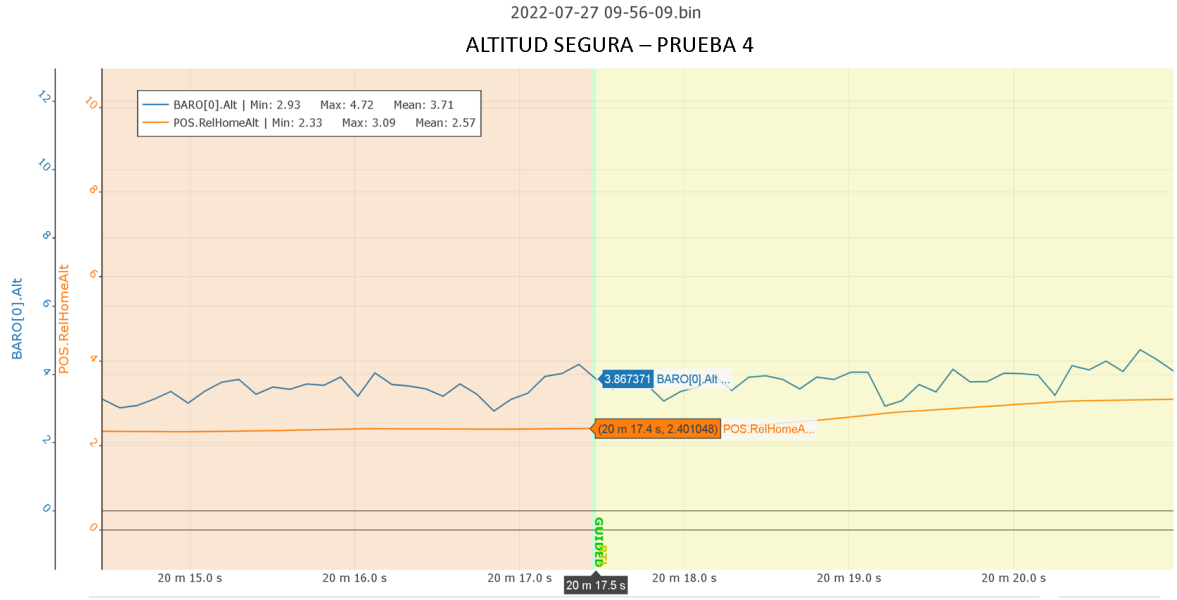


Figura III.7. Registro de altitud del archivo de vuelo durante la prueba 4 – Altitud segura.

SISTEMA DE VISIÓN ARTIFICIAL

Prueba 1:

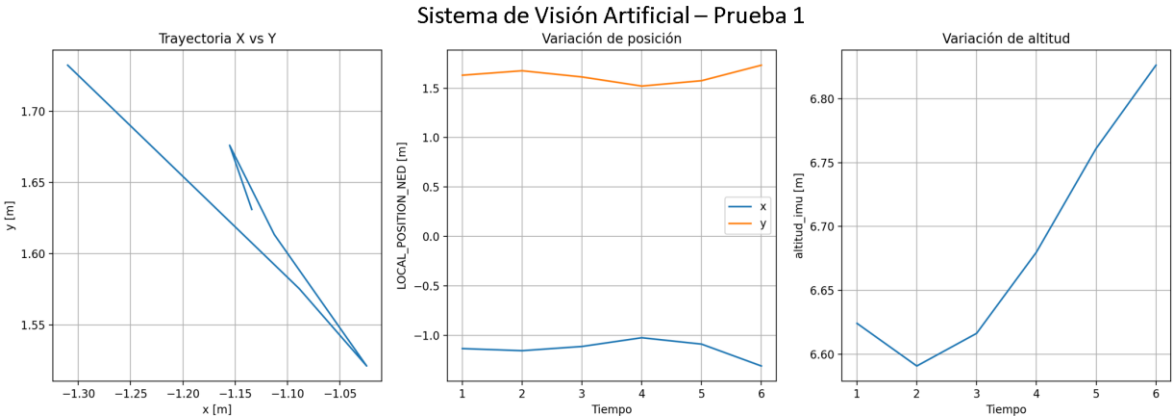


Figura III.8. Trayectorias durante la prueba 1 – Sistema de Visión Artificial.

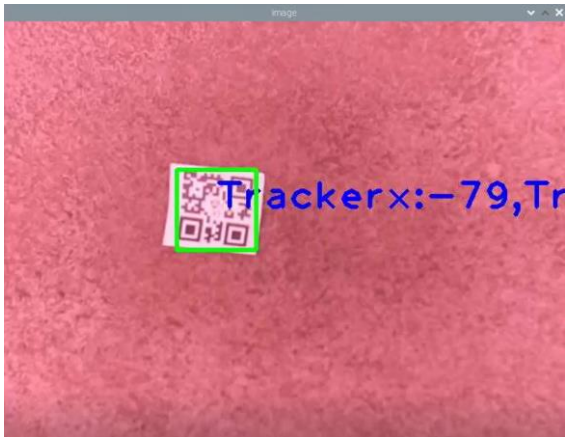


Figura III.9. Seguimiento del helipuerto en la prueba 1 – Sistema de Visión Artificial.

Prueba 2:

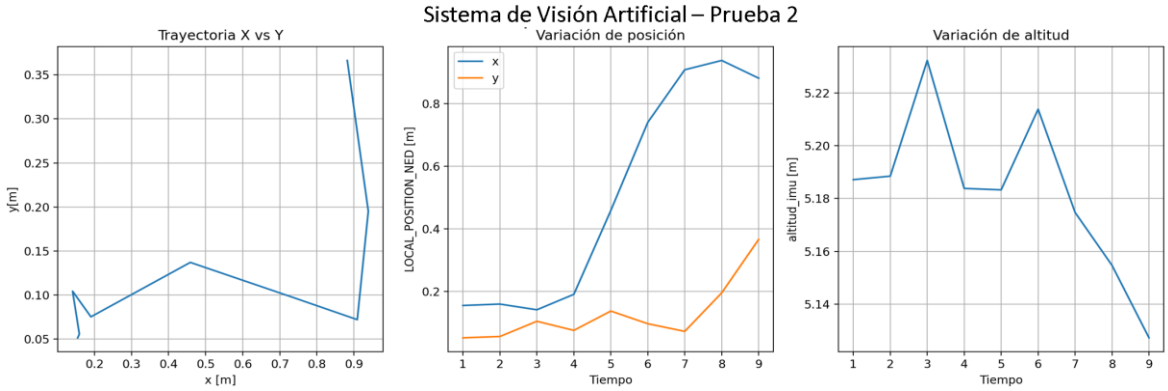


Figura III.10. Trayectorias durante la prueba 2 – Sistema de Visión Artificial.

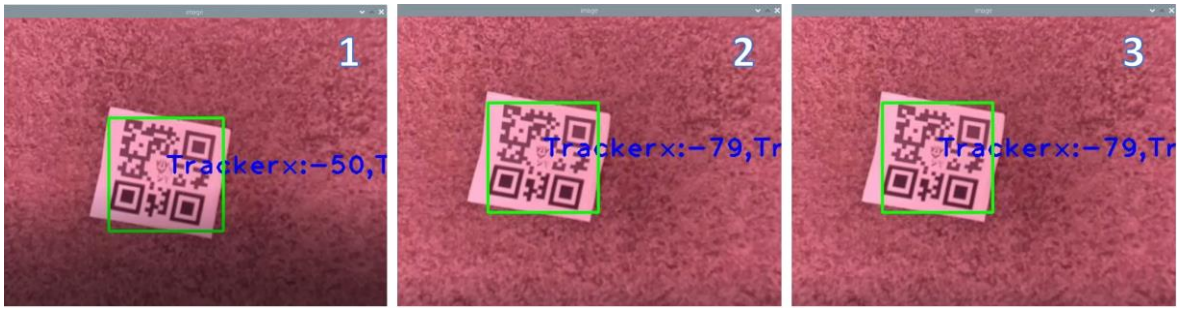


Figura III.11. Seguimiento del helipuerto en la prueba 2 – Sistema de Visión Artificial.

Prueba 3:

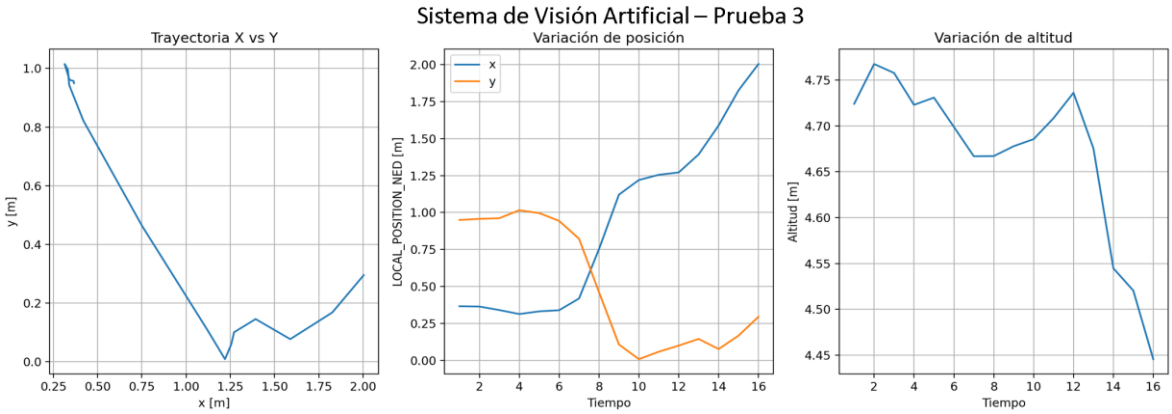


Figura III.12. Trayectorias durante la prueba 3 – Sistema de Visión Artificial.

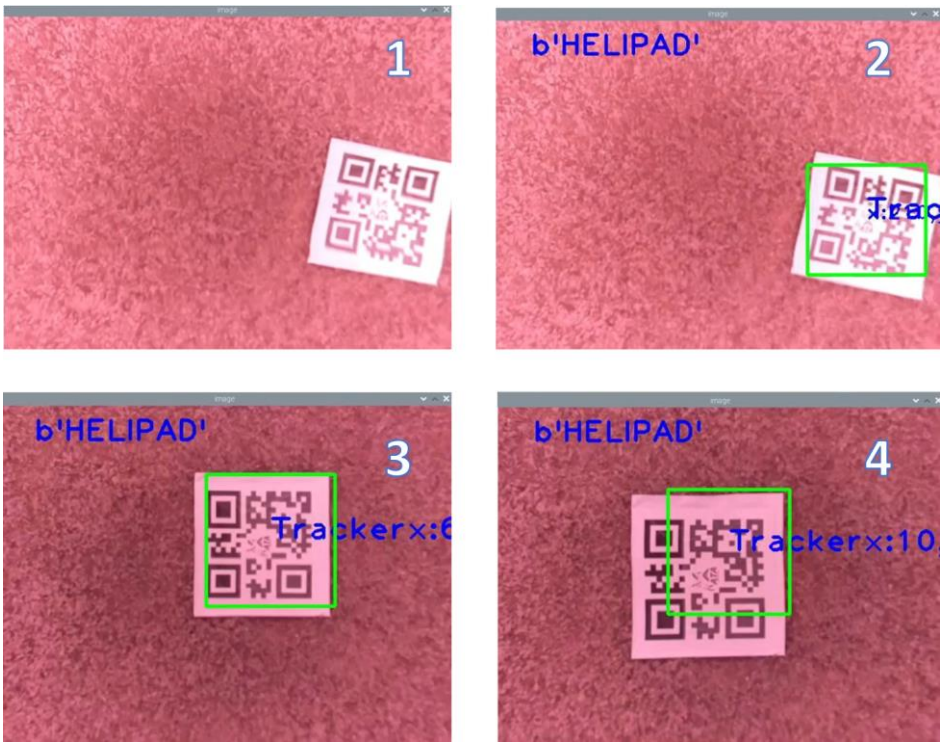


Figura III.13. Seguimiento del helipuerto en la prueba 3 – Sistema de Visión Artificial.

TEST ZONA

Prueba 1:

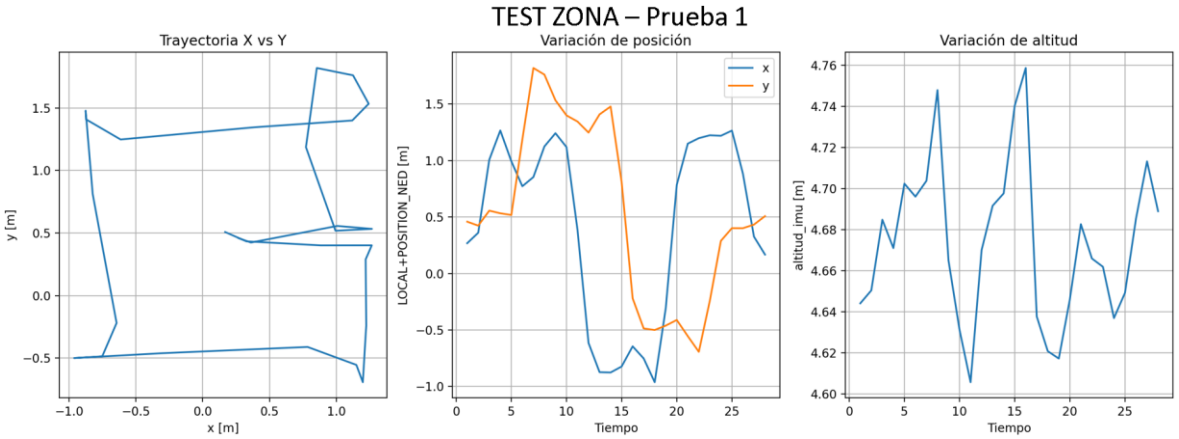


Figura III.14. Trayectorias durante la prueba 1 – Test zona.

Prueba 2:

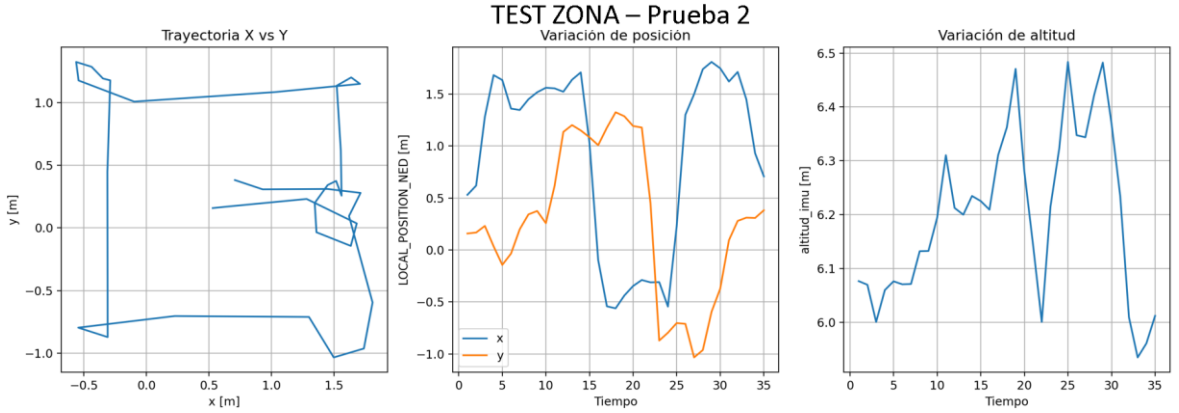


Figura III.15. Trayectorias durante la prueba 2 – Test zona.

Prueba 3:

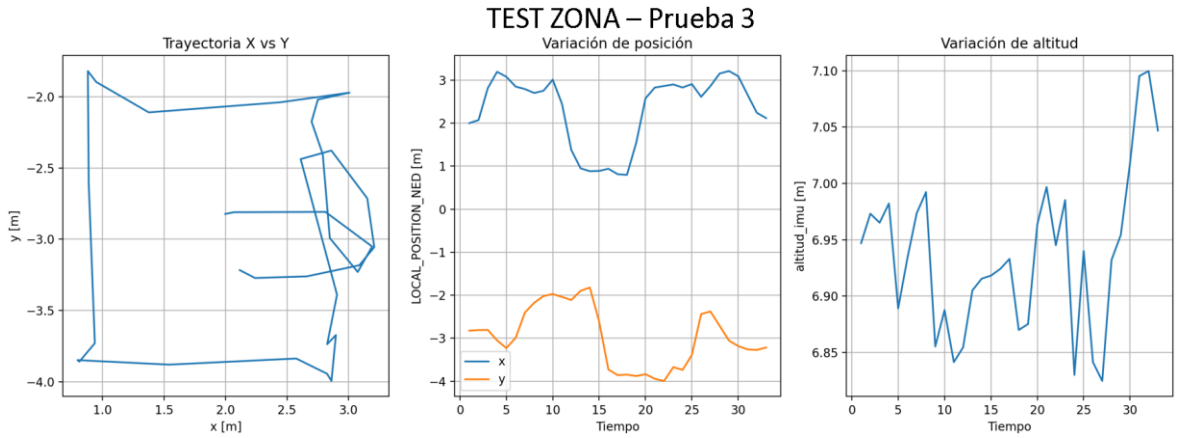


Figura III.16. Trayectorias durante la prueba 3 – Test zona.

ATERRIZAJE CONTROLADO

Prueba 1:

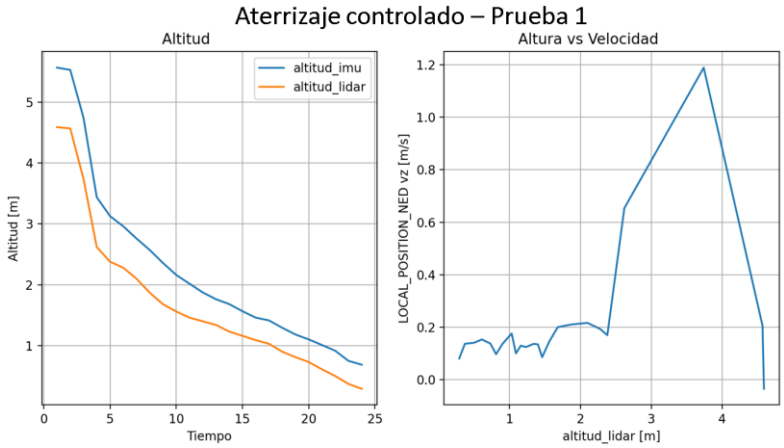


Figura III.17. Trayectorias durante la prueba 1 – Aterrizaje controlado.

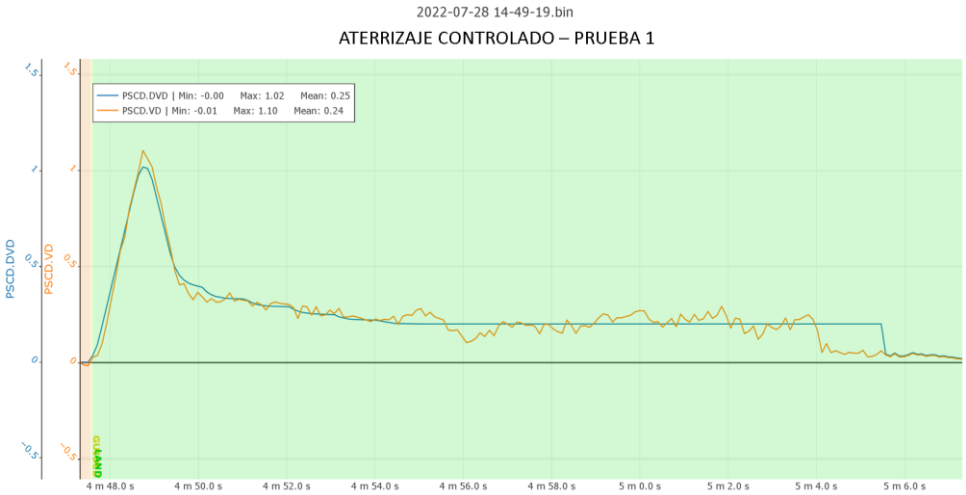


Figura III.18. Registro de velocidad vertical durante la prueba 1 – Aterrizaje controlado.

Prueba 2:

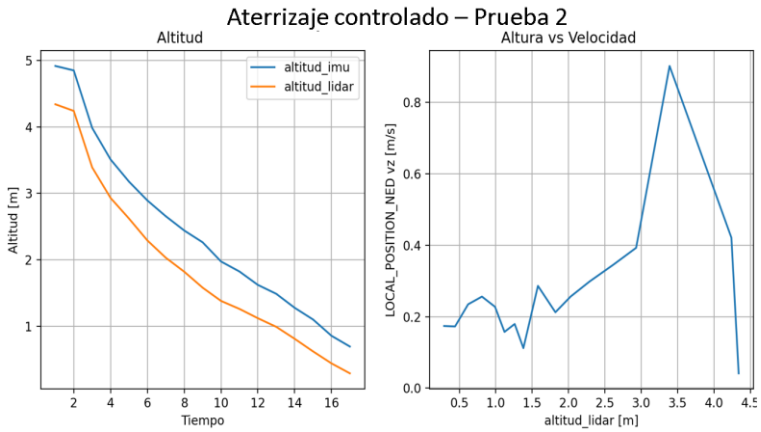


Figura III.19. Trayectorias durante la prueba 2 – Aterrizaje controlado.

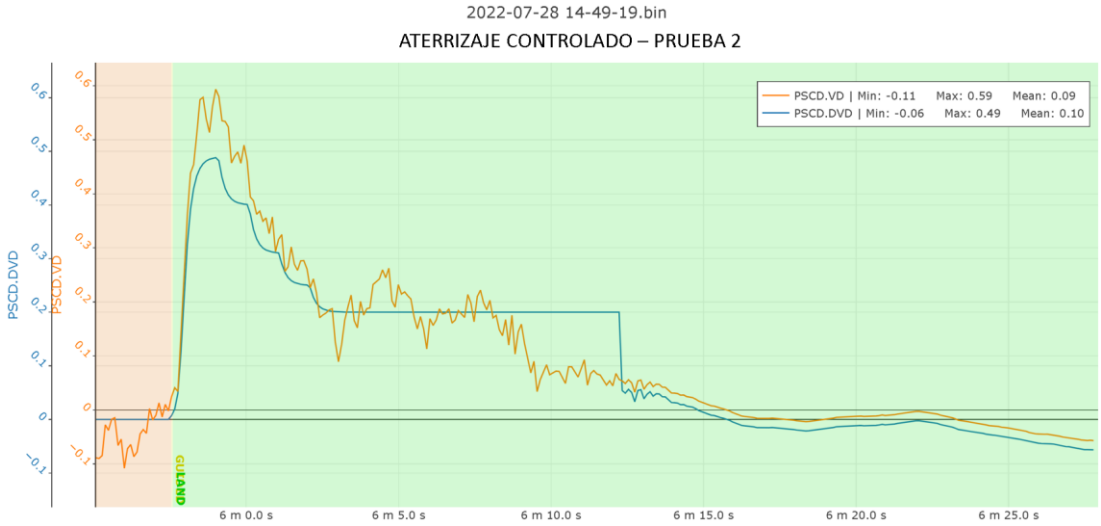


Figura III.20. Registro de velocidad vertical durante la prueba 2 – Aterrizaje controlado.

Prueba 3:

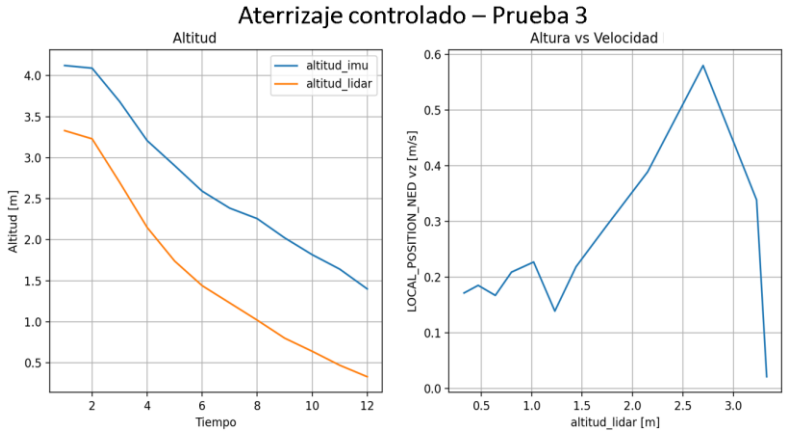


Figura III.21. Trayectorias durante la prueba 3 – Aterrizaje controlado.

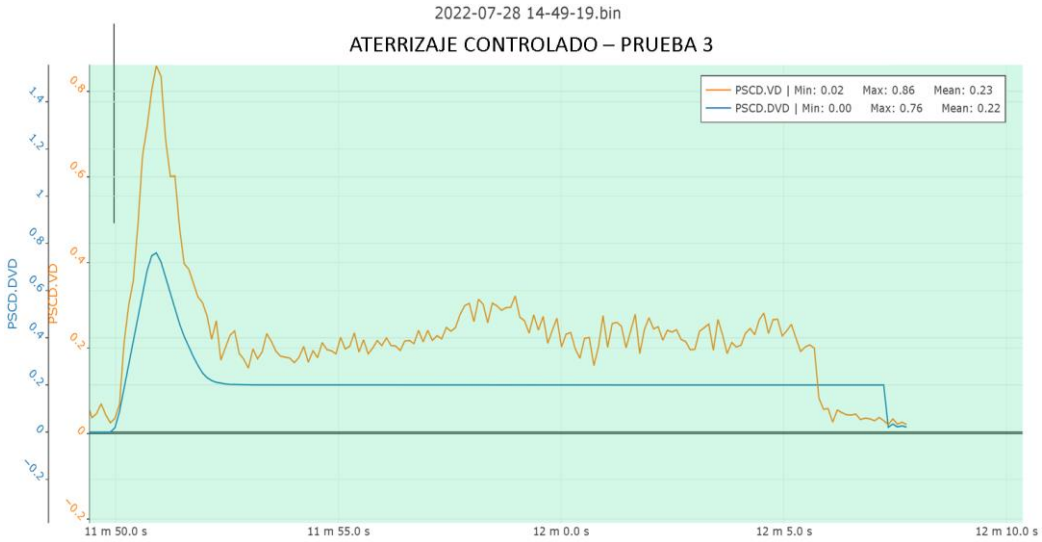


Figura III.22. Registro de velocidad vertical durante la prueba 3 – Aterrizaje controlado.

PRUEBAS COMPLETAS

Prueba 1:

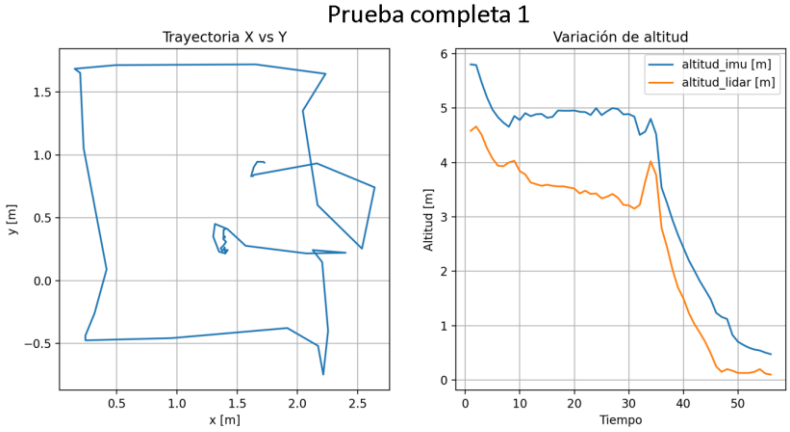


Figura III.23. Trayectoria xy y altitud registrada durante la prueba completa 1.

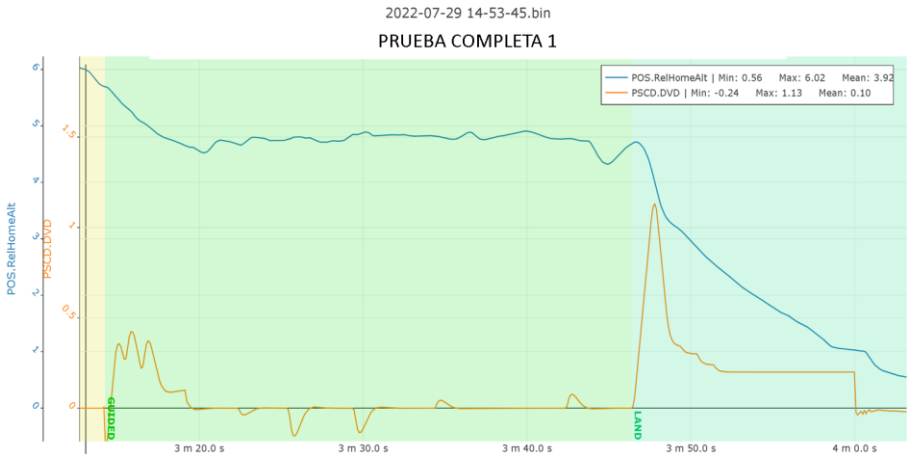


Figura III.24. Altitud y velocidad vertical deseada durante la prueba completa 1.

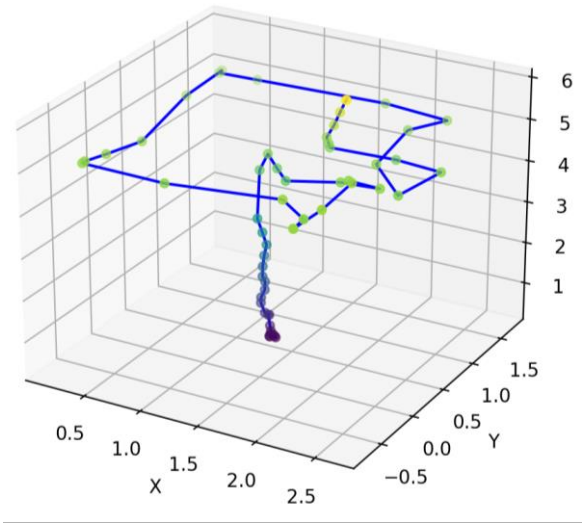


Figura III.25. Trayectoria del cuadricóptero durante la prueba completa 1.

Prueba 2:

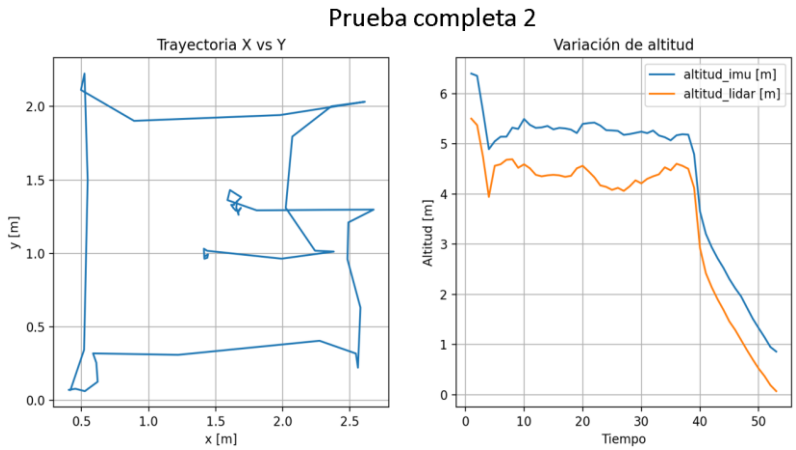


Figura III.26. Trayectoria xy y Altitud registrada durante la prueba completa 2.

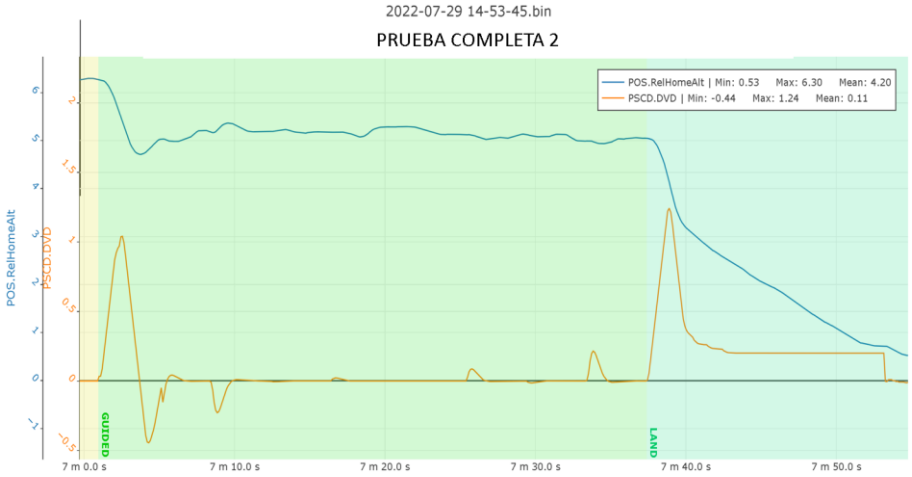


Figura III.27. Altitud y velocidad vertical deseada durante la prueba completa 2.

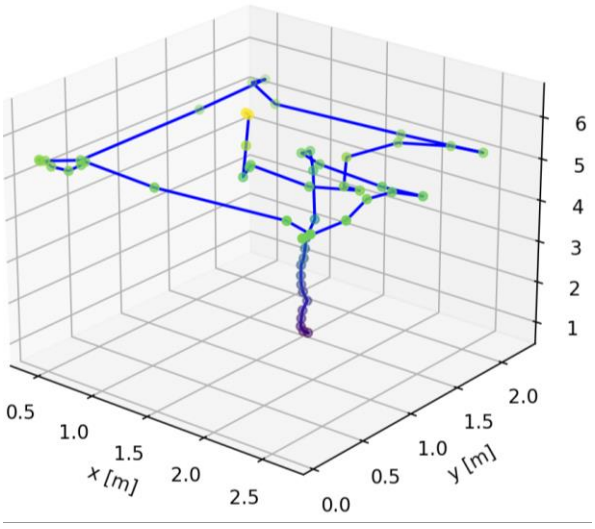


Figura III.28. Trayectoria del cuadricóptero durante la prueba completa 2.

Prueba 3:

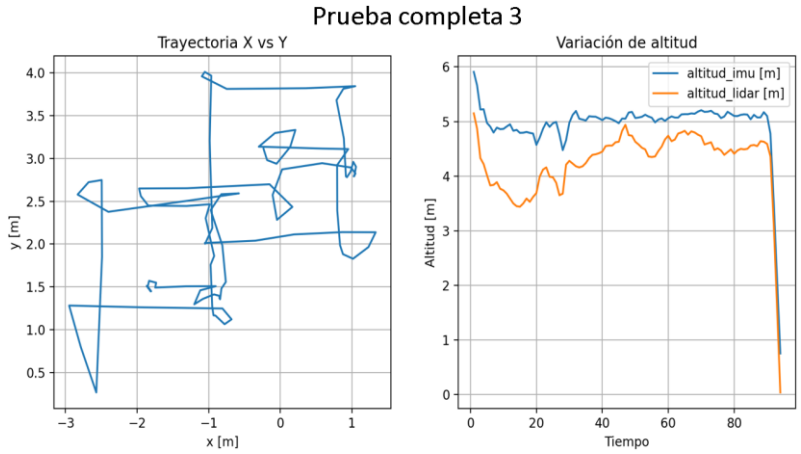


Figura III.29. Trayectoria xy y Altitud registrada durante la prueba completa 3.

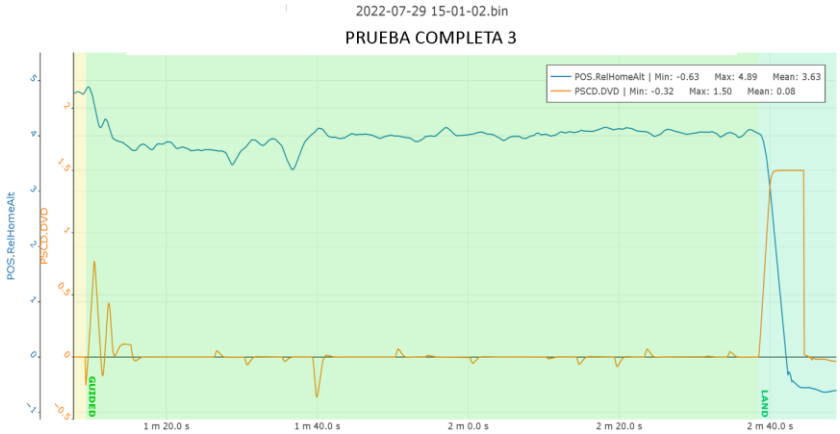


Figura III.30. Altitud y velocidad vertical deseada durante la prueba completa 3.

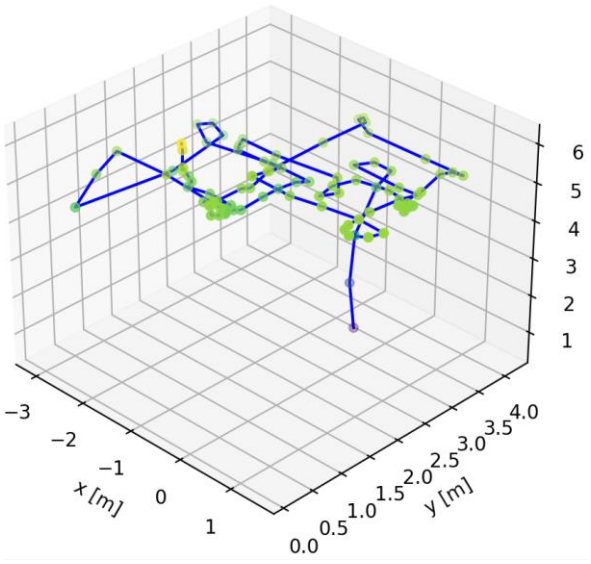


Figura III.31. Trayectoria del cuadricóptero durante la prueba completa 3.

Prueba 4: Detección del QR.

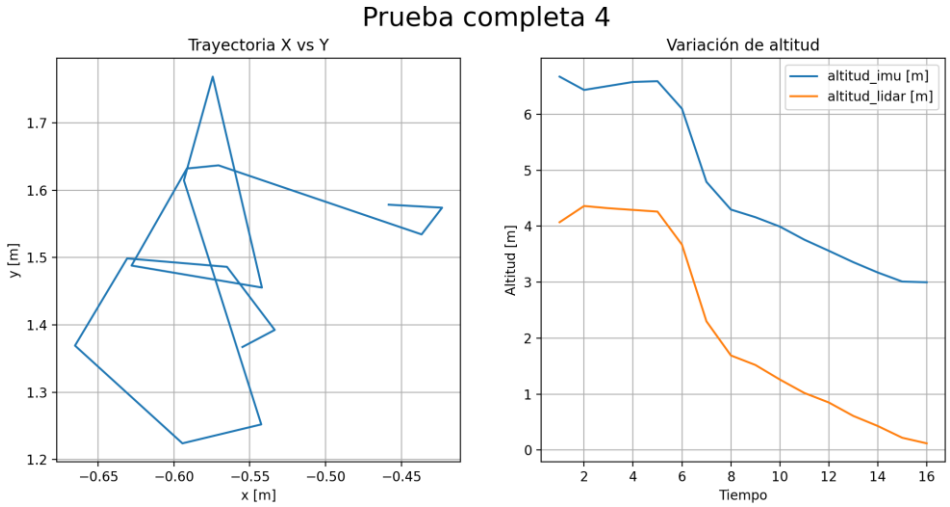


Figura III.32. Trayectoria xy y Altitud registrada durante la prueba completa 4.

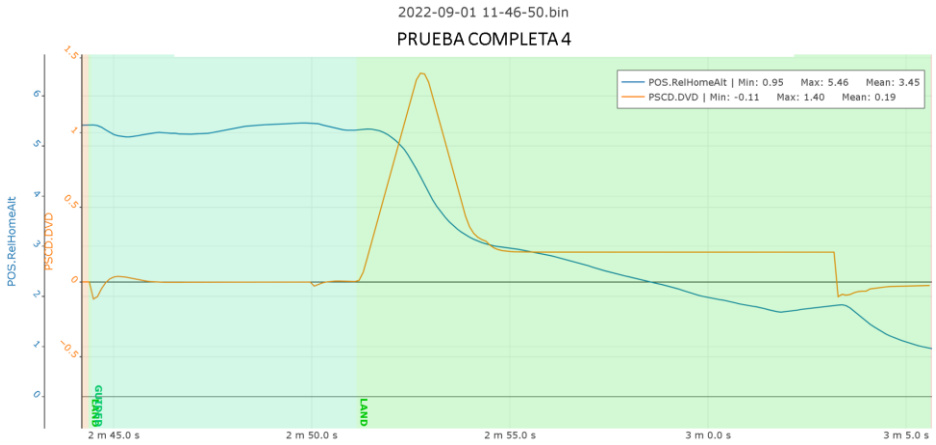


Figura III.33. Altitud y velocidad vertical deseada durante la prueba completa 4.

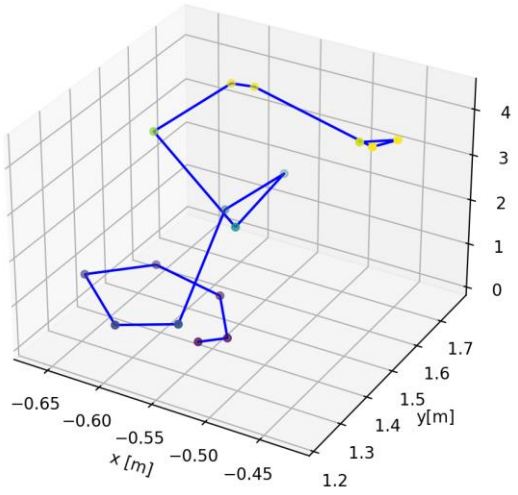


Figura III.34. Trayectoria del cuadricóptero durante la prueba completa 4.

ANEXO IV. Enlaces

Carpeta compartida

LUIS ORTEGA - TIC - TITD201_GR9

Videos demostrativos

https://youtube.com/playlist?list=PLLYK28dOx1CB622AuN3d22e8XX_IiADjZ