UNIVERSIDAD DE CUENCA

Facultad de Ingeniería

Carrera de Electrónica y Telecomunicaciones

Diseño e implementación de algoritmos de visión artificial aplicados en vehículos aéreos no tripulados para aterrizaje autónomo. Caso de estudio: recolección de datos de sensores implementados en agricultura de precisión

Trabajo de titulación previo a la obtención del título de Ingeniero en Electrónica y Telecomunicaciones.

Autores:

Jefferson Santiago Agila Lapo
C.I: 110602723-6
jefferson.agila17@gmail.com

Edisson Paúl Cabrera Tigre
C.I: 010595370-7
eddp29@gmail.com

Director:

Ing. Darwin Fabián Astudillo Salinas, PhD
C.I: 010390703-6

Codirector:

Ing. Luis Ismael Minchala Ávila, PhD
C.I: 030145348-6

Cuenca - Ecuador
13 de octubre de 2020

Resumen

En redes que cubren áreas geográficas amplias, la escalabilidad y distancia entre nodos dificulta la transferencia de datos a puntos de gestión de información. Generalmente, este tipo de redes se destinan al monitoreo y son desplegadas empleando una red de sensores inalámbricos (Wireless Sensor Network (WSN)). La agricultura de precisión es un escenario oportuno y común para el despliegue de este tipo de redes. Estas redes recolectan información del entorno, la transmiten y procesan. Las bajas prestaciones de los nodos de la red precisan el uso de diversos métodos y estructuras que faciliten la transferencia de información. Una red tolerante a retrasos (Delay Tolerant Network (DTN)) se presenta como una solución oportuna para garantizar la transferencia de información a través de almacenamiento y retransmisión. Dentro de este escenario, una DTN puede emplear un vehículo aéreo no tripulado como nodo mula para transportar la información.

El desarrollo e implementación de algoritmos de visión artificial pueden mejorar el rendimiento de las acciones y procesos desarrollados por el vehículo no tripulado. El presente trabajo aborda la etapa de aterrizaje del nodo móvil como alternativa para mejorar la recolección de datos en WSNs empleando una DTN.

El ajuste y personalización que otorga el piloto automático, permite un control total del movimiento y acciones que efectúa el vehículo. La dificultad que se presenta en el desarrollo de algoritmos para este tipo de sistemas radica en los posibles errores que el vehículo experimenta antes y durante el vuelo. Si bien la calibración y configuración del vehículo otorgan cierto grado de confianza en su funcionamiento, *ArduPilot* no garantiza que no se produzcan errores debido ya sea al *software* como al *hardware*. Por tanto, es indispensable tener en cuenta los retos ligados al comportamiento del vehículo y su entorno.

Este trabajo de titulación es una primera aproximación para el desarrollo de aplicaciones que exploten la capacidad de adaptación de un vehículo aéreo no tripulado en aplicaciones que desplieguen WSNs. En esta primera aproximación se logró emplear una cámara para obtener características de un marcador *ArUco*, y usarlas para diseñar algoritmos que controlen el movimiento del dron en la etapa de aterrizaje y la recolección de datos, y evaluar el impacto del entorno en los algoritmos que conforman el sistema de aterrizaje. Además, se consiguió introducir el aterrizaje como alternativa para reducir el consumo energético en la recolección de datos usando drones, evidenciar el funcionamiento de una DTN, contemplarla como una opción valida para la recolección de datos en redes de sensores, y determinar las principales limitaciones presentes en el desarrollo de aplicaciones que involucren vehículos aéreos no tripulados.

Palabras clave : WSN. UAV. OpenCV. DTN. Dron. Ardupilot. Raspberry Pi. MAVLink

Abstract

In networks that cover wide geographic areas, scalability and distance between nodes make it difficult to transfer data to information management points. Generally, these types of networks are used for monitoring and are deployed using a Wireless Sensor Network (WSN). Precision agriculture is a common and timely scenario for the deployment of these types of networks. These networks collect information from the environment, transmit it and process it. The low performance of the network nodes requires the use of various methods and structures that facilitate the transfer of information. A Delay Tolerant Network (DTN) is presented as a timely solution to ensure the transfer of information through store and forward. Within this scenario, a DTN can employ an unmanned aerial vehicle as a mule node to transport the information.

The development and implementation of computer vision algorithms can improve the performance of the actions and processes developed by the unmanned vehicle. The present work addresses the mobile node landing stage as an alternative to improve data collection in WSNs using a DTN.

The adjustment and customization provided by the autopilot allows total control of the movement and actions carried out by the vehicle. The difficulty in developing algorithms for this type of system lies in the possible errors that the vehicle experiences before and during the flight. Although the calibration and configuration of the vehicle provide a degree of confidence in its operation, *ArduPilot* does not guarantee that errors will not occur due to either the software or the hardware. Therefore, it is essential to take into account the challenges linked to the behavior of the vehicle and its environment.

This degree work is a first approach for the development of applications that exploit the adaptability of an unmanned aerial vehicle in applications that deploy WSNs. In this first approach, it was possible to use a camera to obtain characteristics of a marker ArUco, and use them to design algorithms that control the movement of the drone in the landing stage and data collection, and evaluate the impact of the environment on the algorithms that make up the landing system. In addition, it was possible to introduce landing as an alternative to reduce energy consumption in data collection using drones, demonstrate the operation of a DTN, consider it as a valid option for data collection in WSNs, and determine the main limitations present in the development of applications involving unmanned aerial vehicles.

Keywords : WSN. UAV. OpenCV. DTN. Dron. Ardupilot. Raspberry Pi. MAVLink

Índice general

Índice de figuras

Índice de tablas

Cláusula de Propiedad Intelectual

Yo, Jefferson Santiago Agila Lapo, autor de la tesis "Diseño e implementación de algoritmos de visión artificial aplicados en vehículos aéreos no tripulados para aterrizaje autónomo. Caso de estudio: recolección de datos de sensores implementados en agricultura de precisión", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 13 de octubre de 2020

Jefferson Santiago Agila Lapo
110602723-6

Cláusula de Propiedad Intelectual

Yo, Edisson Paúl Cabrera Tigre, autor de la tesis "Diseño e implementación de algoritmos de visión artificial aplicados en vehículos aéreos no tripulados para aterrizaje autónomo. Caso de estudio: recolección de datos de sensores implementados en agricultura de precisión", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 13 de octubre de 2020

Edisson Paúl Cabrera Tigre
010595370-7

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Yo, Jefferson Santiago Agila Lapo en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Diseño e implementación de algoritmos de visión artificial aplicados en vehículos aéreos no tripulados para aterrizaje autónomo. Caso de estudio: recolección de datos de sensores implementados en agricultura de precisión", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos. Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 13 de octubre de 2020

Jefferson Santiago Agila Lapo
110602723-6

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Yo, Edisson Paúl Cabrera Tigre en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Diseño e implementación de algoritmos de visión artificial aplicados en vehículos aéreos no tripulados para aterrizaje autónomo. Caso de estudio: recolección de datos de sensores implementados en agricultura de precisión", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos. Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 13 de octubre de 2020

Edisson Paúl Cabrera Tigre
010595370-7

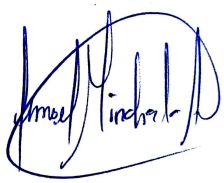
Certifico

Que el presente proyecto de tesis: Diseño e implementación de algoritmos de visión artificial aplicados en vehículos aéreos no tripulados para aterrizaje autónomo. Caso de estudio: recolección de datos de sensores implementados en agricultura de precisión, fue dirigido y revisado por mi persona.

Ing. Darwin Fabián Astudillo Salinas, PhD
Director

Certifico

Que el presente proyecto de tesis: Diseño e implementación de algoritmos de visión artificial aplicados en vehículos aéreos no tripulados para aterrizaje autónomo. Caso de estudio: recolección de datos de sensores implementados en agricultura de precisión, fue dirigido y revisado por mi persona.



Ing. Ismael Minchala, PhD
Co-director

Dedicatoria

A mi familia y amigos

Quiero dedicar esta tesis a mi familia, en especial a mis padres, por ser el ejemplo a seguir del cual aprendí tantas cosas y agradezco hoy en día. Por último, quiero dedicar esta tesis a mis amigos, gracias a su apoyo y palabras de aliento incondicional en el proceso de cumplir mis sueños.

Jefferson Agila

Dedicatoria

A mi familia y amigos

Por su amor, apoyo incondicional, trabajo y sacrificio en todo este tiempo. Gracias a ustedes he alcanzado las metas que me he propuesto.

Edisson Cabrera

Agradecimientos

En primer lugar queremos agradecer a nuestro director, Ingeniero Fabian Astudillo, así como también a nuestro co-director Ingeniero Ismael Minchala, quienes con sus conocimientos y apoyo nos guiaron a través de cada una de las etapas de este proyecto para alcanzar los objetivos planteados.

También queremos agradecer al Ingeniero Santiago Torres por colaborar con los recursos y herramientas necesarias para llevar a cabo el proceso de investigación. No hubiésemos alcanzado estos resultados de no haber sido por su incondicional ayuda.

Un agradecimiento especial al Ingeniero Luis González y al personal del Laboratorio Micro Red Eléctrica, por brindarnos un espacio de trabajo lleno de confianza y apoyo para realizar las distintas pruebas y experimentos durante el desarrollo del proyecto.

LOS AUTORES

Abreviaciones y Acrónimos

ADC Analog-to-Digital Converter. 26
ADU Application Data Unit. 27
AP Agricultura de Precisión. 26
API Application Program Interface. 36, 57, 107
AR Augmented Reality. 40, 41

BP Bundle Protocol. 27, 28

CAD Computer-Aided Design. 45
CGR Contact Graph Routing. 29
CH Cluster Head. 42
CLA Convergence Layer Adapter. 27
CM Cluster Member. 42

DHCP Dynamic Host Configuration Protocol. 102
DNS Domain Nane Server. 101
DOP Dilution of precision. 76
DSP Digital Signal Processing. 26
DTN Delay Tolerant Network. 1, 2, 22–25, 27, 28, 41, 43, 69, 70, 72, 78, 81–84, 114

EEPROM Electrically Erasable Programmable Read-Only Memory. 116
EID Endpoint Identifier. 27, 28, 70, 71
ESC Electronic Speed Controller. 31, 32, 49, 84, 85, 94, 95, 97

FPV First-Person View. 32

GCS Ground Control Station. 32–35, 41, 87, 118
GLONASS Globalnaya Navigatsionnaya Sputnikovaya Sistema. 30, 31
GPIO General Purpose Input/Output. 107–109
GPS Global Positioning System. 26, 30–33, 41, 56, 57, 60, 76, 84, 86, 95, 96, 98, 99, 110
GUI Graphical User Interface. 33, 34, 36, 100, 104

HDOP Horizontal Dilution of Precision. 31, 76, 83
HSV Hue, Saturation, Value. 41

ION-DTN Interplanetary Overlay Network. 28, 29, 69, 70, 78, 79, 83, 100, 110–112, 114
IP Internet Protocol. 28, 102, 103, 113, 118
IR Infrared. 40, 41

JPL Jet Propulsion Laboratory. 28

MAVLink Micro Aerial Vehicle Link. 33–35, 57, 107
MP Mission Planner. 32, 33, 35, 49, 85, 89–91, 95–98, 107, 115

NASA National Aeronautics and Space Administration. 27
NED North-East-Down. 56, 62, 64

OpenCV Open Source Computer Vision Library. 36, 38, 40, 41, 50–52, 54, 55, 64, 104–106

PID Proporcional-Integral-Derivativo. 40, 41, 84
PLA ComPolylactic acid. 46
PnP Plug and Play. 89

RAM Random-Access Memory. 28
RF Radio Frecuencia. 33, 34
ROS Robot Operating System. 40
RPAS Remotely Piloted Aircraft System. 29
RTL Return To Launch. 32

SBC Single Board Computer. 23, 43–45, 49, 69, 75
SDR Simple Data Recorder. 112
SIRGAS Sistema de Referencia Geocéntrico para las Américas. 22
SITL Software In The Loop. 35, 57, 106, 107, 115, 116, 118
SNM Sobre el Nivel del Mar. 30
SNS Sobre el Nivel del Suelo. 30
SSH Secure SHell. 101

TCP Transmission Control Protocol. 35
TSP Travelling Salesman Problem. 42

UART Universal Asynchronous Receiver/Transmitter. 108
UAS Unmanned Air/Aircraft System. 41, 42
UAV Unmanned Aerial Vehicle. 22–25, 29–32, 40–43, 49, 55–58, 60, 62, 64, 69, 70, 75, 78, 82, 83, 85, 88, 91, 92, 94, 98
UDP User Datagram Protocol. 35, 118
USB Universal Serial Bus. 88, 89, 91, 94, 95

VANT Vehículo Aéreo No Tripulado. 29

WSN Wireless Sensor Network. 1, 2, 21, 25, 41, 69

XML Extensible Markup Language. 33

Capítulo 1

Introducción

Este capítulo contextualiza la problemática que presenta el aterrizaje de un vehículo aéreo no tripulado, sus limitantes, y la ventaja que ofrece a la recolección de datos en diferentes redes de sensores el aterrizaje en los nodos. En primer lugar, se describe el problema asociado a la recolección de datos en redes que cubren amplias áreas geográficas (Sección 1.1). A continuación, se presenta una contribución a la solución de la problemática planteada y se enfoca la dirección que tendrá el presente trabajo (Sección 1.2). Luego, se define el alcance que el proyecto tendrá (Sección 1.3) y se formulan los objetivos (Sección 1.4). El capítulo finaliza presentando una visión general de la estructura del documento (Sección 1.5).

1.1. Identificación del problema

En redes que cubren grandes áreas geográficas, la escalabilidad y distancia entre nodos en ocasiones imposibilita la transferencia de datos a puntos de gestión de información [1]. El despliegue de infraestructura para transmisión cableada representa una gran inversión y se encuentra limitada por el tipo de área, su acceso e importancia. Esta infraestructura puede incurrir en desperdicio de recursos de acuerdo al flujo de datos generado en los nodos.

La mayoría de redes que cubren áreas geográficas extensas se destinan al monitoreo. La red se conforma por nodos capaces de obtener, procesar y transmitir información de su entorno, denominados sensores. Generalmente, los nodos son dispositivos de bajo consumo energético y costo reducido. Las redes de sensores inalámbricos, o Wireless Sensor Network (WSN), emplean enlaces inalámbricos para establecer comunicación entre sus nodos sensores.

La agricultura es un escenario oportuno para la integración de tecnologías emergentes que provean herramientas para mejorar la gestión y toma de decisiones. La agricultura de precisión se basa en la observación, medición y respuesta a la variabilidad inter e intra parcela [2]. El objetivo de la agricultura de precisión es la ejecución de acciones para optimizar el rendimiento de insumos agrícolas basado en la información recogida por diversos medios [3].

En este ámbito, es común el despliegue de WSN que acaparen la totalidad de los cultivos desplegados

en diversas áreas. En Ecuador la agricultura de precisión se ha implementado en la producción de caña de azúcar, empleando imágenes multi-espectrales que permiten detectar irregularidades en los cultivos [4].

A medida que la red crece, el área que cubre es mayor. El procesamiento y transmisión de información se vuelve cada vez más problemático en escenarios con redes extensas. No todas las aplicaciones de sensores requieren de procesamiento de información en tiempo real. Por consiguiente, el almacenamiento y retransmisión representa una herramienta para la recolección de datos en redes extensas.

Una Red Tolerante a Retrasos, o Delay Tolerant Network (DTN), permite establecer conexión entre nodos de manera intermitente y la transferencia de datos se realiza gracias a procesos de almacenamiento y retransmisión [5]. Una DTN puede transportar datos en ambientes donde las interrupciones y altas tasas de error son comunes [6], debido a condiciones extremas o movilidad de nodos. Las redes de sensores inalámbricos que cubren amplias áreas geográficas emplean como alternativa de transporte de datos el despliegue de una red DTN.

Las DTN presentan una solución a esta problemática explotando el movimiento de nodos como retransmisores o mulas DTN [7] para transportar datos entre distintos puntos de la red. Distintos esquemas y tipos de nodos brindan la movilidad necesaria para transportar datos dentro de una red DTN. Generalmente, los nodos móviles se adaptan a servicios existentes que comunican parcial o totalmente el área de interés.

Los drones representan una opción de transporte de información en redes de este tipo. Los vehículos aéreos no tripulados, Unmanned Aerial Vehicle (UAV) o drones han tomado protagonismo en la investigación y desarrollo de aplicaciones. Una de las razones de su protagonismo es la economía de escala, que ha permitido que estos dispositivos tengan cada vez un costo menor, por consiguiente, un mayor índice de penetración en el mercado.

1.2. Justificación

Los drones presentan una alternativa para desarrollar procesos como monitoreo de cultivos y fumigación de pesticidas [8], proporcionando mayor precisión en el mapeo y tratamiento de zonas de gestión, gracias al control dinámico de vuelo y descenso del vehículo.

Sin embargo, este tipo de vehículos presentan algunas limitantes como tiempo de vuelo, relacionado directamente con el tipo y capacidad de la batería, precisión de sensores, disponibilidad de vuelo (restricciones), tipo de terreno, entre otras.

La incorporación de drones en diversas tareas y sistemas se debe a su gran capacidad de personalización y adaptación de componentes para desarrollar múltiples funciones [9]. Por otro lado, es posible realizar tareas de manera autónoma estableciendo un plan de vuelo específico, en donde, el individuo detalla la ruta a alcanzar[10]. Además, un UAV brinda una baja complejidad de pilotaje tanto en modos de vuelo asistidos o de forma manual empleando únicamente un control remoto o radio control.

En Ecuador, se han implementado sistemas que emplean UAVs para la transmisión de imágenes térmicas destinadas al control de incendios [11], monitoreo de la degradación de la vegetación en las Islas Galápagos empleando imágenes georreferenciadas [12], y elaboración de cartografía a escalas grandes enlazadas al marco de referencia Sistema de Referencia Geocéntrico para las Américas (SIRGAS) [13]. Además, el país alcanzó un nuevo hito al lanzar en el año 2014 su primer UAV diseñado y construido en su totalidad en territorio ecuatoriano, denominado *UAV-2 "Gavilán"*, destinado para el monitoreo

de las fronteras y áreas inaccesibles [14].

La serie de acciones o fases que realiza un UAV para desplazarse son: despegue (*take off*), vuelo estacionario (*hovering flight*), vuelo guiado/orientado (*forward flight*) y aterrizaje (*landing*) [15].

Una de las etapas de mayor conflicto para un dron es la etapa de aterrizaje [16], debido a que las aspas generan un mayor flujo de aire a bajas alturas produciendo un incremento en la turbulencia del vehículo, desembocando en la necesidad de un proceso de control más exhaustivo. La principal limitante con la que cuenta un UAV es la relación capacidad de batería - tiempo de vuelo. Aumentar la capacidad de la batería, no garantiza que el dron puede aumentar su tiempo de vuelo, debido a que, el peso agregado por la composición física de la batería puede resultar en un requerimiento de potencia mayor para mantener el vehículo en movimiento.

Aterrizar en diferentes puntos de la red reduce el consumo energético del vehículo, permitiendo alcanzar un mayor número de nodos y/o emplear energía para procesos complementarios como transmisión o procesamiento de imágenes.

El presente proyecto propone abordar la etapa de aterrizaje, planteando el diseño e implementación de un algoritmo de aterrizaje usando detección y localización de características de imágenes capturadas con una cámara a bordo utilizando algoritmos de visión artificial.

1.3. Alcance

El trabajo de titulación implementará una prueba de concepto centrada en la agricultura de precisión. El caso de estudio plantea incorporar el uso de drones en la agricultura de precisión para la recolección de datos. Esta recolección sería realizada en los nodos sensores, los cuales junto con el UAV implementarían una red DTN.

En cada nodo se tendrá un puerto, el dron utilizará el sistema de control implementado para aterrizar y descargar los datos, una vez que termina de descargarlos se dirigirá al siguiente puerto, y continuará de esta manera hasta completar el plan de vuelo definido por la ubicación de nodos. La prueba de concepto se implementará y evaluará en un ambiente controlado.

A fin de reducir costos para el despliegue de los nodos sensores, se empleará soluciones de *software* libre en una computadora de placa única o Single Board Computer (SBC). Puntualmente, se utilizará el controlador de vuelo *PixRaptor*, una versión genérica del controlador de *hardware* abierto, *Pixhawk 1*, con sus diferentes periféricos; una placa *Raspberry Pi 3B* como computadora a bordo/complementaria y nodo sensor; y como base de *software* la librería *OpenCV*, y *Dronekit* de *Python*.

1.4. Objetivos

1.4.1. Objetivo general

Diseñar e implementar un sistema de control de aterrizaje basado en visión artificial en un ambiente controlado para la recolección de datos en una red DTN.

1.4.2. Objetivos específicos

El presente trabajo tiene los siguientes objetivos específicos:

- Investigar y analizar algoritmos de aterrizaje autónomo y de recolección eficiente de datos.

- Seleccionar métricas para evaluar los algoritmos de aterrizaje.
- Diseñar un algoritmo de aterrizaje autónomo que implemente algoritmos de visión artificial.
- Implementar un algoritmo de aterrizaje y de recolección de datos para crear la red DTN.
- Evaluar algoritmos empleando las métricas seleccionadas.

1.5. Estructura del documento

En este capítulo se ha contrastado la integración de vehículos aéreos no tripulados provistos con algoritmos de visión artificial como alternativa de transporte de información en redes extensas. El resto del documento se estructura de la siguiente manera:

- **Capítulo 2:** presenta los fundamentos teóricos de los elementos que participan en el desarrollo del sistema de control de aterrizaje. Se presenta una descripción conceptual, y técnica de los principales componentes involucrados tanto en el software como en el hardware del trabajo de titulación.
- **Capítulo 3:** introduce de manera organizada y descriptiva las principales contribuciones de diversos autores para abordar las problemáticas planteadas en este capítulo. Se presenta una breve descripción de la estructura y conceptos involucrados en el desarrollo, y resultados de las diferentes contribuciones. Además, se contrasta la propuesta establecida en este trabajo de titulación con las soluciones planteadas en el capítulo.
- **Capítulo 4:** presenta de forma sistemática el diseño e implementación de los diversos algoritmos que conforman el sistema de aterrizaje y recolección de datos. Se detalla los fundamentos, estructura y funcionamiento de los distintos procesos planteados.
- **Capítulo 5:** introduce los escenarios experimentados empleando el sistema desarrollado, y presenta los resultados obtenidos en los diferentes experimentos de forma organizada, comparando y evaluando el rendimiento de los algoritmos.
- **Capítulo 6:** presenta las contribuciones y experiencia de desarrollo del trabajo de titulación, e introduce trabajos futuros derivados de la temática tratada.
- **Anexo A:** presenta de manera detallada la conexión, actualización, calibración y configuración de los todos elementos que conforman el UAV.
- **Anexo B:** presenta la instalación, y configuración del sistema operativo y paquetes de *software* involucrados en el despliegue tanto de la red DTN, como en la comunicación y control del UAV.
- **Anexo C:** presenta la instalación, configuración y conexión del simulador de vuelo con el piloto automático (dron) y estaciones externas (ordenador).

Capítulo 2

Marco teórico

En este capítulo se presenta la base teórica requerida que sustenta el desarrollo del presente trabajo de titulación. El capítulo inicia con la descripción de una red inalámbrica de sensores (Sección 2.1), continua con un enfoque y aplicaciones de agricultura de precisión (Sección 2.2). A continuación, se introduce y describe una red DTN y el protocolo *Bundle* (Sección 2.3), se presenta la implementación seleccionada para el desarrollo de este trabajo (Sección 2.4), y se detalla brevemente los marcos y clasificación de UAVs (Sección 2.5).

Luego, se introducen los componentes del sistema *ArduPilot* (Sección 2.6), se detalla el protocolo de comunicación para UAVs (Sección 2.7), y se presenta el proyecto *Dronekit*, capaz de crear aplicaciones para computadoras complementarias de drones (Sección 2.8). En seguida, se presenta la estación terrestre minimalista *MAVProxy* (Sección 2.9), y el simulador de vuelo para piloto automático (Sección 2.10). El capítulo finaliza con una descripción de la biblioteca de visión artificial (Sección 2.11).

2.1. Redes inalámbricas de sensores

Una red inalámbrica de sensores (WSN) es un conjunto de sistemas dispersos espacialmente denominados sensores. Estos sensores pueden obtener, procesar y transmitir información capturada a un punto de recolección, generalmente denominado *sink* o sumidero. La transmisión puede efectuarse de manera directa entre nodos o a través de un nodo relé. Emplear un nodo de intermedio permite reducir el consumo de potencia del transmisor inalámbrico, pero requiere mayor gestión en la sincronización y enrutamiento de mensajes [17]. La Figura 2.1 presenta los tipos de transmisión de datos entre nodos en una WSN.

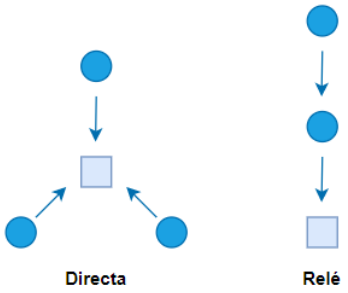


Figura 2.1: Tipos de transmisión de datos entre nodos

Generalmente, los nodos que componen una red de este tipo cuentan con al menos los siguientes elementos:

- **Microcontrolador:** dispositivo pequeño, de baja potencia, encargado del procesamiento de información y control de periféricos del nodo.
- **Sensor:** encargado de medir o detectar una propiedad o magnitud física en su entorno.
- **Unidad Analog-to-Digital Converter (ADC):** encargada de transformar la señal analógica de salida del sensor a una señal digital.
- **Unidad Digital Signal Processing (DSP):** destinada a filtrar, modular y procesar señales digitales.
- **Dispositivo de transmisión inalámbrica:** encargado de transmitir los datos al siguiente punto de la red.
- **Fuente de energía:** regularmente una batería de bajo costo y capacidad.
- **Dispositivo de recepción inalámbrico:** necesario para establecer un protocolo de enrutamiento y retransmitir datos entre nodos de la red.

De acuerdo al tipo de aplicación de la red, un nodo puede incorporar un actuador y una unidad de posicionamiento Global Positioning System (GPS) [18].

El diseño de nodos sensores se ve restringido en peso, volumen y a las condiciones del entorno. Cada nodo debe consumir poca energía, tener bajo costo, ser prescindible, y debe ser autónomo.

Las topologías más empleadas en una red de sensores son: estrella, malla, y árbol. La transmisión de un solo salto con sumidero se emplea en redes de pequeña escala. La eficiencia energética depende la distancia entre el sumidero y el nodo sensor [19]. El nodo sumidero o *sink* es de mayores prestaciones y cuenta con un módulo de almacenamiento.

Para reducir aún más el consumo de energía, algunos nodos cuentan con la capacidad de apagar partes específicas del hardware para ahorrar energía [20].

2.2. Agricultura de precisión

La Agricultura de Precisión (AP) hace referencia al manejo de la agricultura empleando tres factores: observación, cálculo y ejecución de una acción en respuesta a fenómenos que afecten el desarrollo de cultivos entre parcelas. El objetivo de la AP es proporcionar técnicas adecuadas para la toma de decisiones en el manejo de cultivos, que gestionen de manera óptima el rendimiento de la materia prima [2].

2.3. Redes tolerantes a retrasos

Las redes tolerantes a retrasos/interrupciones o DTN nacieron como una solución de la National Aeronautics and Space Administration (NASA) para proveer interconexión confiable en misiones espaciales. Una DTN puede transmitir información en entornos sujetos a interrupciones frecuentes, retrasos largos y altas tasas de error.

Un escenario donde las interrupciones pueden ser recurrentes se presenta en redes que cuentan con limitantes de energía, con nodos que dependen de baterías. Eventualmente, al agotarse las baterías la red se particiona imposibilitando la comunicación.

Una red DTN puede afrontar esta problemática empleando dos mecanismos. El primero es utilizar almacenamiento y reenvío sobre múltiples rutas, y en escalas de tiempo potencialmente largas, mientras que, el segundo es aprovechar el contacto oportunista en la red, explotando el movimiento de los nodos para transmitir información [1].

La arquitectura de una red tolerante a retrasos define una capa denominada *Bundle*, que se ubica sobre la capa de transporte de la red en la que se aloje.

2.3.1. La capa bundle

Un nodo DTN es capaz de implementar la capa *bundle* para usar almacenamiento persistente, que mitigue las interrupciones causadas en la red. El nodo envía mensajes de longitud variable denominada unidades de datos de aplicación o Application Data Unit (ADU). La capa *bundle* transforma un ADU en una o más unidades de información denominadas *bundles*. Cada *bundle* tiene un formato definido que contiene dos o más "bloques". Los bloques contienen información que regularmente se encuentra en el encabezado del protocolo.

Las fuentes y destinos de los *bundles* se identifican a través de un punto final o Endpoint Identifier (EID). Un *bundle* contiene una marca de tiempo, un indicador de vida útil, clase de servicio y una longitud. De esta manera, la capa *bundle* puede enrutar conociendo los requisitos de transferencia de los datos solicitados.

2.3.2. El protocolo bundle

El protocolo *Bundle* o Bundle Protocol (BP) tiene capacidad para:

- Utilizar la movilidad física para la transmisión de datos.
- Aprovechar la conectividad oportunista tanto de forma unidireccional como bidireccional.
- Hacer frente a la conectividad intermitente incluso si el emisor y receptor están inactivos en la red.

La Figura 2.2 presenta la ubicación de BP dentro de la pila de protocolos estándar. La interfaz entre BP y un protocolo subyacente se denomina adaptador de capa de convergencia o Convergence Layer Adapter (CLA). Un CLA utiliza servicios de algún protocolo nativo para enviar y recibir paquetes. El protocolo *bundle* se coloca en la capa de aplicación para ciertas redes.

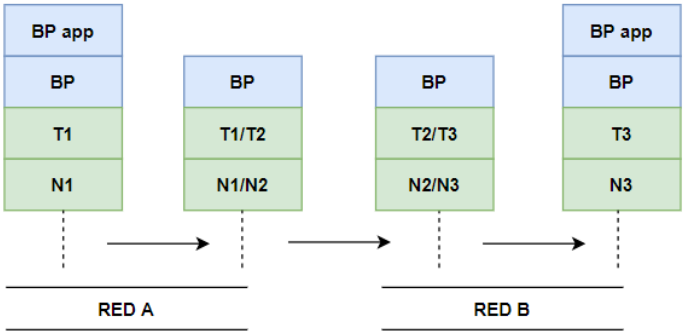


Figura 2.2: Protocolo Bundle en el modelo de pila de protocolos

No hay un método establecido para el enrutamiento en una red DTN. El protocolo de enrutamiento de la red debe aumentar la probabilidad de entrega. Uno de los algoritmos de enrutamiento más usados para DTNs es *Epidemic*. El *epidemic routing protocol* permite la distribución de mensajes dentro de porciones conectadas de la red con consideraciones mínimas de la topología. La conectividad periódica garantiza la entrega eventual de mensajes [21].

2.4. ION-DTN

La red de superposición interplanetaria o Interplanetary Overlay Network (ION-DTN) es una implementación de la arquitectura de redes tolerantes a retrasos descrita en el RFC 4838, desarrollada por el laboratorio de propulsión a reacción, o Jet Propulsion Laboratory (JPL), como alternativa a implementaciones de BP. ION-DTN busca abordar las limitaciones en las operaciones de misiones interplanetarias, reducir el costo, y permitir la operación de redes de comunicación a través de enlaces espaciales, de superficie planetaria y terrestre [22].

ION-DTN optimiza el tamaño de encabezado en las transmisiones estableciendo un esquema de identificador de *endpoint* (EID) exclusivo, que permite abreviarse en pares enteros binarios sin signo. De esta forma, un EID está identificado por un número de nodo y un servicio, de manera análoga a un *host* en red identificado por su dirección Internet Protocol (IP) y un número de puerto.

2.4.1. Recursos de almacenamiento

La operación eficiente de ION-DTN depende de la retención de información de los distintos protocolos por intervalos de tiempo largos. La implementación gestiona dos clases de recursos de almacenamiento: memoria de trabajo (*working memory*) y *heap*. Los dos recursos de almacenamiento se asignan al momento de iniciar la implementación.

La memoria de trabajo es un conjunto de memoria compartida de tamaño fijo asignada desde la Random-Access Memory (RAM) del sistema. El objetivo de esta memoria es almacenar los datos temporales que se producen en la operación de ION-DTN. Por otro lado, el ***heap*** es un grupo de almacenamiento de tamaño fijo, generalmente no volátil que puede ocupar RAM, un archivo, o ambos. Este recurso de almacenamiento se utiliza para almacenar datos como colas de paquetes para cálculo de ruta, transmisión, que deberían conservarse en el caso de que el sistema se reinicie [23].

2.4.2. Enrutamiento

El enrutamiento establece decisiones y operaciones de protocolo acorde a la información conocida (declarada) al iniciar la operación de los nodos. ION-DTN requiere el conocimiento de características de conectividad del nodo local y sus vecinos para efectuar un proceso de enrutamiento exitoso. Esta información que cuenta con fecha y hora en la que los nodos entran en vigencia, se denomina **plan de contacto** (*Contact plan*). Una oportunidad de comunicación o **contacto** (*contact*) es un intervalo de tiempo durante el cual se espera que los datos de un nodo A sean transmitidos y recibidos (parcial o totalmente) por un nodo B. Por otro lado, un **intervalo de rango** (*range interval*) es un intervalo de tiempo en el cual se espera que la distancia entre dos nodos varíe menos de un segundo luz a partir de una distancia anticipada.

El cálculo de ruta de paquetes de unidifusión se realiza utilizando Contact Graph Routing (CGR). Éste es un sistema de enrutamiento dinámico que calcula rutas empleando información de tiempo de contactos de comunicación programada dentro de una topología variable. La información empleada por este sistema se obtiene del *contact plan* [24].

2.5. Vehículos Aéreos no tripulados

El término Vehículo Aéreo No Tripulado (VANT) proviene de la terminología inglesa UAV, pero el término más popular empleado para nombrar a una aeronave de este tipo es dron [25]. Este término es una derivación directa del término en inglés *drone*, que significa zumbido. Este término se le atribuyó debido al sonido característico de un dron durante el vuelo [26].

Debido a que la forma y estructura de este tipo de aeronave no es la misma en todas sus presentaciones, el uso de términos como VANT, UAV o Remotely Piloted Aircraft System (RPAS) es común de acuerdo al contexto en el que se esté empleando. El término RPAS es usado por la organización de aviación civil internacional [27].

Un UAV es una aeronave sin tripulación, controlada remotamente, capaz de ejecutar misiones de vuelo tanto de manera manual como automática. Dicho vehículo es propulsado por diferentes tipos de motores, como: combustión, eléctrico o de reacción. Existe una amplia variedad de diseños de drones tomando como referencia características como: forma, volumen, configuración y funcionalidad. La facilidad en adaptación de un UAV ha permitido la integración de este tipo de vehículos a aplicaciones como misiones de reconocimiento de suelos para gestión de desastres [28], recopilación de noticias en disturbios civiles [29], apoyo en el manejo de incendios forestales [30], levantamientos topográficos para diferentes áreas [31], entre otros.

2.5.1. Marcos para multicopters

El marco o *frame* de un UAV determina el número de unidades de propulsión en el vehículo. De acuerdo al número de unidades de propulsión se definen clases de UAV como *Plane*, *Tricopter*, *Hexacopter* u *Octocopter*. Cada clase posee una configuración de motores, que determinar la ubicación de los mismo respecto al marco. Para el desarrollo del presente trabajo se empleará un *quadcopter* en configuración *X*.

Un *Quadcopter* posee cuatro unidades de propulsión, permitiendo escalar su *frame* para sistemas de mayor tamaño. Algunas de sus características se describen a continuación:

- Es simétrico por lo que presenta una operación más sencilla al momento de controlar el balanceo, cabeceo, desplazamiento y vibración.
- Son capaces de realizar tareas complejas de extracción de información, captura de imágenes, y detección de objetos.
- Su principal desventaja es que al fallar una unidad de propulsión se produce una descompensación que provoca una precipitación fuerte que puede afectar en gran medida la salud del vehículo.

2.5.2. Clasificación de drones

No hay un estándar específico acerca de la clasificación de UAVs. Sin embargo, existen diversos criterios de clasificación con respecto a las características del vehículo como tamaño, alcance, y resistencia. Según el departamento de defensa de los Estados Unidos [32] se puede clasificar un dron de acuerdo a su tamaño en cuatro categorías. La Tabla 2.1 presenta la clasificación de drones de acuerdo a su tamaño. Se caracterizan las categorías por el peso máximo de despegue, expresado en libras, la máxima altitud de funcionamiento, expresada en pies Sobre el Nivel del Mar (SNM), o pies Sobre el Nivel del Suelo (SNS), y la velocidad aérea (expresada en nudos).

Tabla 2.1: Clasificación de drones según el tamaño y peso

Categoría	**Tamaño**	**Peso máximo de despegue (libras)**	**Altitud de funcionamiento normal (Pies)**	**Velocidad aérea (nudos)**
Grupo 1	Pequeño	0-20	<1200 SNS	<100
Grupo 2	Mediano	21-55	<3500 SNS	<250
Grupo 3	Grande	<1320	<18000 SNM	<250
Grupo 4	Extra Grande	>1320	>18000 SNM	Cualquiera

De acuerdo al grado de personalización que se puede otorgar a un dron, se puede clasificar en drones comerciales y drones armables.

Los UAVs comerciales se emplean para tareas de carácter investigativo, entretenimiento o captura de imágenes profesionales. Generalmente este tipo de drones no puede ser modificado y se encuentra altamente optimizado, para ofrecer mejor estabilidad, mayor alcance y tiempo de vuelo. Estos drones ofrecen ventajas como:

- Mejor calidad de posicionamiento GPS.
- Soporte para altas velocidades de desplazamiento.
- Incorporación de sensores anti-impacto.
- Incorporación de modos asistidos.
- Incorporación de modos seguros (en caso de pérdida de señal del radio control o GPS)

En los drones comerciales el sistema de vuelo puede contrarrestar de forma eficiente el efecto causado por problemas como balanceo, sobrecalentamiento de motores y controladores de velocidad, peso, vibración y pandeo del *frame*. Además, presentan un mejor sistema de posicionamiento. Este tipo de drones emplea dos sistemas de posicionamiento: el GPS, propiedad del gobierno de los Estados Unidos y operado por el departamento de fuerza espacial [33], y el sistema de satélite de navegación global de Rusia, o Globalnaya Navigatsionnaya Sputnikovaya Sistema (GLONASS)[34]. La integración de estos dos sistemas proporciona una mejora significativa en el error de posicionamiento, llegando a ser menor a 2 metros [35].

La gran cantidad de aficionados al mundo de los sistemas de radio control ha permitido el desarrollo de sistemas de código abierto que permitan ensamblar y configurar UAVs personalizados, denominados drones armables. De esta manera, un aficionado puede ensamblar en su totalidad un vehículo, empleando diversidad de elementos de diferentes marcas y modelos. Sin embargo, este proceso requiere de un conocimiento técnico sobre la estructura, y los elementos que conforman un dron para su selección, ensamblaje y configuración. Un dron cuenta con nueve elementos fundamentales:

- Cuadro o *frame*
- Controlador de vuelo
- Controlador de velocidad electrónico o Electronic Speed Controller (ESC)
- Motores
- Batería
- Módulo distribuidor de potencia
- Hélices
- Radio Control
- Módulo GPS
- Brújula

Estos elementos son esenciales para que un dron tenga la capacidad de volar. El proceso de selección de elementos debe estar sujeto a especificaciones técnicas, compatibilidad, y tipo de vehículo. Una descripción detallada del proceso de conexión, emsamblaje y configuración de un dron armable se presenta en el Anexo A.

El *firmware* que ofrece *Ardupilot*, principal sistema para drones armables, no cuenta aún con compatibilidad con múltiples sistemas de posicionamiento integrados. Solo tiene acceso al GLONASS por medio de módulos complementarios [36]. El módulo GPS conectado al controlador solo requiere de seis enlaces satelitales para proporcionar un dato seguro [37]. Para garantizar un mejor comportamiento al operar el vehículo, algunos modos asistidos requieren de un valor de dilución de precisión menor a 1. La dilución de precisión hace referencia a los factores que empobrecen la precisión en la aproximación de posición. El tipo de dilución más empleada por *Ardupilot* es la dilución horizontal o Horizontal Dilution of Precision (HDOP).

Los drones comerciales de gama alta optimizan el consumo de energía, empleando baterías inteligentes optimizadas de acuerdo a peso y capacidad, que ofrecen características como descarga automática, carga balanceada, protección de sobrecorriente, modo reposo, entre otras. Los drones armables pueden obtener estas características instalando módulos adicionales al controlador.

2.6. ArduPilot

ArduPilot es un sistema de piloto automático de código abierto, capaz de controlar una gran variedad de sistemas de vehículos aéreos, terrestres y acuáticos. El código fuente se encuentra en constante desarrollo a cargo de una gran comunidad de profesionales y fanáticos [38].

El piloto automático funciona en una amplia variedad de hardware y permite la integración de diversos periféricos que proveen funcionalidades avanzadas como comunicación en tiempo real, adquisición de datos de telemetría, e integración de estaciones terrestres y computadoras de placa única. Además, cuenta con documentación exhaustiva de las funciones disponibles y un foro directo con una

comunidad en crecimiento y con gran experiencia. *ArduPilot* está conformado por tres componentes: *hardware*, *firmware* y *software*.

2.6.1. Hardware

El *hardware* del *ArduPilot* consiste del piloto automático, denominado controlador. El controlador es el cerebro del vehículo, se encarga de usar las entradas de información provista por los sensores, procesarlas y enviar señales de salida a actuadores para ejecutar movimientos en el vehículo de forma controlada. Una configuración básica para un vehículo que integre *ArduPilot* cuenta con: controlador, control electrónico de velocidad o ESC, motor, batería, brújula, radio control, y módulo GPS. Una descripción más detallada de la configuración de un vehículo aéreo se presenta en el Anexo A.

2.6.2. Firmware

El firmware es el código que ejecuta el controlador y se adapta al tipo de vehículo que se pretende desplegar. El proyecto está compuesto por cinco códigos diseñados para diferentes tipos de vehículos [39], estos son: *ArduCopter*, *ArduPlane*, *ArduRover*, y *Antenna Tracker*.

En el proyecto se trabaja con el código de *ArduCopter*, éste está destinado a vehículos de rotor en varias configuraciones. Incluye nivelación automática y retención de altitud, misiones autónomas, monitoreo de estado del sistema, First-Person View (FPV), regreso al lanzamiento o Return To Launch (RTL), y vuelo asistido para mantener posición usando GPS, acelerómetro y barómetro.

2.6.3. Software

La interfaz para el controlador se denomina estación de control en tierra o Ground Control Station (GCS). Los vehículos de *ArduPilot* son gestionados por el *software* para estación de control en tierra, Mission Planner (MP). La estación terrestre permite configurar, modificar, y calibrar el vehículo desde una computadora. Algunas GCSs avanzadas permiten la planificación de misiones autónomas, operación, monitoreo y análisis de telemetría del vehículo durante el tiempo de ejecución de movimientos. Las estaciones terrestres más conocidas son MP y *QGround control*. En el presente proyecto usamos *Mission Planner* debido a su compatibilidad con el controlador *PixRaptor*.

El MP es un *software* de estación terrestre para vehículos tipo *Plane*, *Copter* y *Rover*. Éste permite acceder a todas las funciones del proyecto *ArduPilot*. Esta MP es compatible con *Windows*, y se emplea para configurar los periféricos del vehículo, actualizar e instalar *firmware* del controlador de vuelo [40].

MP permite configurar los parámetros de versión del *firmware*, proporcionando un control total de los periféricos del vehículo. De esta manera, el vehículo puede ser personalizado para adaptarse a diversos escenarios. MP permite planificar, guardar, cargar, y analizar misiones automáticas en la interfaz del controlador de vuelo.

En el caso de presentarse algún fallo en el UAV, la estación terrestre ofrece la posibilidad de descargar registros de operación desde el vehículo. Los registros de operación pueden ser:

- ***Dataflash:*** registros de datos que se almacenan en la tarjeta de memoria complementaria a bordo del piloto automático. Generalmente, se emplea una tarjeta microSD para este propósito.
- **Telemetría:** estos registros se graban cuando se conecta el piloto automático a la computadora donde se ejecuta MP, y almacenan datos obtenidos mediante un enlace de telemetría.

Analizar los datos dentro de los registros de operación permite diagnosticar problemas en el sistema antes, durante y después del vuelo.

MP permite establecer un enlace de telemetría empleando sistemas de radio como *SiK 3DR*, una plataforma de radio de fuente abierta, versátil y ligera, que permite rangos superiores a 300 m, funciona con paquetes Micro Aerial Vehicle Link (MAVLink) y se integra con diversas GCSs [41]. Algunas características que posee este kit de telemetría son las siguientes:

- Control del vehículo mientras ejecuta una misión de vuelo.
- Grabado de registros de vuelo en la computadora de la estación terrestre.
- Visualización en tiempo real de diferentes parámetros como: posición, número de satélites GPS, calidad de señal GPS, nivel de batería, errores, entre otros.

La Figura 2.3 presenta la interfaz gráfica de usuario o Graphical User Interface (GUI) de MP.

Figura 2.3: Visión general de la estación terrestre *Mission Planer*

2.7. Micro aerial vehicle link

MAVLink es un protocolo de mensajería liviano empleado por vehículos y estaciones terrestres para enviar flujos de datos de telemetría mediante multidifusión, y comandos utilizando comunicación punto a punto con entrega garantizada [42].

Los mensajes se encuentran definidos en archivos Extensible Markup Language (XML). Este archivo determina el conjunto de mensajes admitidos por un sistema MAVLink. Se emplean estas definiciones de mensajes para generar bibliotecas para lenguajes de programación compatibles, como *C*, *C++*, *Python*, *Java*, entre otros. MAVLink establece un mecanismo de serialización de mensajes en la capa de aplicación. Soporta diferentes tipos de capas de transporte y medios de transmisión.

Los mensajes pueden enviarse por varios tipos de conexiones serie sin importar la tecnología subyacente. Pueden usarse enlaces Radio Frecuencia (RF) a frecuencias menores a 1 GHz, alcanzando un rango máximo de 500 m y una tasa de datos no superior a 250 kbps [43].

Al establecer una conexión GCS - vehículo, cada dispositivo envía un mensaje denominado `HEARTBEAT` una vez por segundo (1 Hz). El mensaje `HEARTBEAT` anuncia la existencia de un sistema, y puede usarse para descubrir sistemas conectados a la red. Si un sistema deja de enviar el mensaje `HEARTBEAT`, se asume que ese sistema ya no se encuentra disponible. `HEARTBEAT` cuenta con seis campos:

- `MAV_TYPE:` tipo de componente o vehículo.
- `MAV_AUTOPILOT:` clase o tipo de piloto automático.
- `MAV_MODE_FLAG:` mapa de bits del modo del sistema.
- `Custom mode:` campo reservado para banderas específicas del piloto automático.
- `MAV_STATE:` estado del sistema (no inicializado, inicializado, calibrado pero no listo para el vuelo, activo, en espera, crítico, emergencia, apagado).
- `MAVLink version:` versión del protocolo.

La estación terrestre o computadora complementaria puede solicitar datos al vehículo usando el mensaje `SET_MESSAGE_INTERVAL`, que establece el intervalo entre mensajes para un identificador de mensaje específico. Además, la GCS puede enviar comandos para planificar, ejecutar misiones o modificar parámetros del vehículo usando el mensaje `COMMAND_LONG`. Este mensaje contiene 7 parámetros en función del comando específico que se utilice. Cuenta además con cuatro campos, que se detallan a continuación:

- `target_system:` sistema que ejecuta el comando.
- `target_component:` componente que ejecuta el comando.
- `command:` identificador del comando.
- `confirmation:` número de transmisión del comando.

Los mensajes pueden ser transmitidos en multidifusión, si se tratan de mensajes de estado como `HEARTBEAT`, o punto a punto si se tratan de mensajes de comando como `COMMAND_LONG` [44].

2.8. Dronekit

DroneKit-Python es un proyecto de código abierto, que permite crear aplicaciones que se ejecutan en una computadora a bordo y establecen comunicación con el controlador de vuelo utilizando un enlace de baja latencia. De este modo, agrega mayor capacidad de procesamiento, sensibilidad e inteligencia a las aplicaciones implementadas. *DroneKit-Python* también puede comunicarse con vehículos y estaciones terrestres a través de un enlace RF de mayor latencia.

El protocolo usado para la comunicación con el controlador de vuelo es MAVLink, que brinda acceso a información de telemetría, parámetros del vehículo, y permite la gestión y control del movimiento y operación del vehículo [45]. La instalación y configuración de *Dronekit* en *Ubuntu* y *Raspbian* se presenta en el Anexo B, Sección B.3.

2.9. MAVProxy

MAVProxy es un paquete de *software* de GCS para sistemas basados en MAVLink. Se trata de una estación terrestre minimalista, portátil y adaptable a vehículos que utilicen el protocolo MAVLink. Además, cuenta con complementos para proporcionar una GUI básica. La aplicación se basa en una consola de línea de comandos [46]. La Figura 2.4 presenta la consola y el mapa de *MAVProxy*.

Figura 2.4: Consola y Mapa de MAVProxy

La instalación y configuración de *MavProxy* en *Raspbian* y *Ubuntu* se presenta en el Anexo B, Sección B.3 y C, respectivamente.

2.10. SITL

Software In The Loop (SITL) es un simulador que permite ejecutar el código fuente de un piloto automático para diferentes vehículos (*Plane*, *Copter*, o *Rover*) sin necesidad de contar con algún tipo de *hardware*. SITL permite ejecutar *ArduPilot* directamente en una computadora empleando un compilador *C++*, ofreciendo un ejecutable nativo para experimentar con el comportamiento del vehículo.

Los datos de los sensores se originan en un modelo de dinámica de vuelo alojado en un simulador de vuelo. *ArduPilot* cuenta con una amplia gama de simuladores de vehículos. La Figura 2.5 presenta la arquitectura de SITL. El simulador requiere para su funcionamiento *software* adicional como *MAVProxy* o MP. *MAVProxy* permite establecer comunicación con el usuario a través de instrucciones en línea de comando. De esta manera, puede modificar parámetros, modo de vuelo, crear e iniciar misiones, o establecer entradas de radio control. Esta GCS se comunica con las librerías de *ArduPilot* a través del protocolo MAVLink empleando protocolos de transporte como User Datagram Protocol (UDP) o Transmission Control Protocol (TCP).

ArduCopter consiste en una serie de funciones y librerías, que engloba todos los procesos necesarios para simular el comportamiento del controlador de vuelo en un vehículo multirotor. La simulación física del vehículo se encuentra establecida mediante un modelo dinámico en el archivo `sim_multicopter.py`. El modelo dinámico se basa en cuatro parámetros: masa, aceleración, velocidad límite y velocidad límite rotacional [47]. La instalación y configuración de SITL en *Ubuntu* se presenta en el Anexo C.

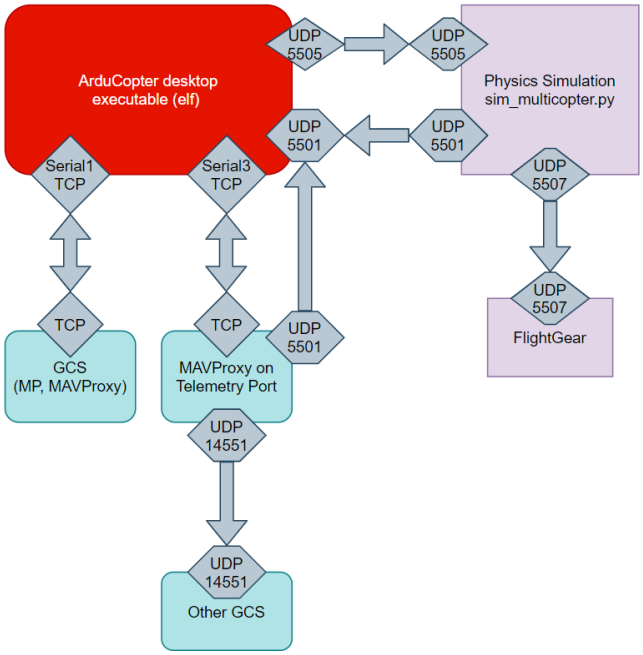


Figura 2.5: Arquitectura SITL

2.11. OpenCV

Open Source Computer Vision Library (OpenCV) es una biblioteca de código abierto formada por cientos de algoritmos de visión por computadora. OpenCV es modular, un paquete incluye varias bibliotecas. La Application Program Interface (API) de esta librería está basada en *C++* para las versiones 2.x.

Entre los módulos disponibles en OpenCV se encuentra un módulo de procesamiento de imágenes, filtrado, transformaciones geométricas, conversión de espacios de color, histogramas, entre otros. Además, cuenta con módulos para análisis de video, calibración de cámaras, descriptores de características, detección de objetos, entrada/salida de video y GUI de alto nivel [48].

2.11.1. Marcadores fiduciales

El término fiducial proviene del latín fiducia que significa confianza [49]. En inteligencia artificial se denomina como imagen fiducial al objeto o conjunto de objetos capturados por medio de instrumentos ópticos, que son empleados como punto de referencia para obtener información confiable [50]. Los objetos referenciales son también empleados para producir marcadores que representen alguna información objetiva, como un código de barras.

Este tipo de marcadores puede dividirse o segmentarse empleando técnicas sencillas para distinguir puntos, líneas, figuras o manchas. La técnica de segmentación permite la creación de marcadores binarios que albergan información, donde los bits son representados por medio de figuras cuadradas y el objeto fiducial referencial es el marco que los aloja.

2.11.1.1. Marcadores ArUco

Un marcador *ArUco* es un derivado de marcadores fiduciales de base cuadrada, compuesto por un marco cuyo interior aloja información expresada en una matriz de datos binarios codificados $(K \times K)$ de tal manera que el sistema de reconocimiento permita detectar y corregir errores [51]. La decodificación de estos datos binarios representa un número entero que cumple la función de identificador. La dimensión del marcador tiene relación con el valor del identificador, a mayor valor, mayor tamaño del marcador. La base de los marcadores *ArUco* es su marco cuadrado, que es el elemento referencial cuya segmentación permite adquirir la matriz interna, como muestra la Figura 2.6.

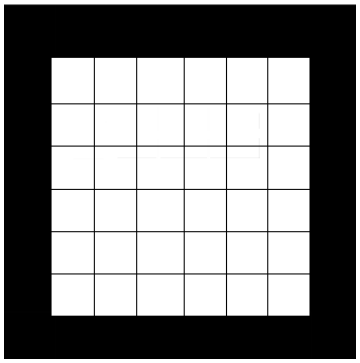


Figura 2.6: Marco de marcador de 6x6

La Figura 2.6 ilustra el marco referencial, donde la segmentación ecuánime deriva en la matriz de 6×6, cuyos segmentos internos son iguales y representan la matriz de datos binarios. Los marcadores *ArUco* se clasifican por el tamaño de la matriz que alojan. Sin embargo, la dimensión de estos está representado por el cuerpo del marco, cuya longitud es empleada para la segmentación, como se muestra en la Figura 2.7.

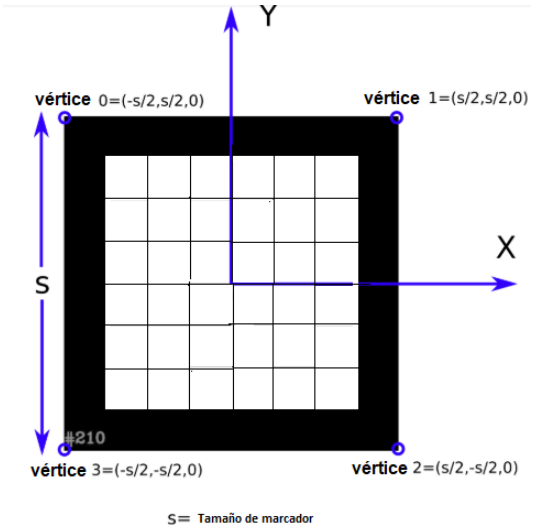


Figura 2.7: Distribución de bits en matriz interna de un un marcador *ArUco*

2.11.1.2. Generación de marcadores

OpenCV cuenta con la librería *ArUco*, que ofrece funciones para crear marcadores de N número de bits. Esta función permite elegir entre matrices de 4×4, 5×5, 6×6 , 7×7, donde un mayor número de bits permite tener mayor cantidad de identificadores [52].

El proceso de generación comienza con la declaración del diccionario contenedor de los marcadores, todos los diccionarios poseen contenedores de 50, 100, 250 y 1000 marcadores, que se diferencian en la codificación para evitar errores. En otras palabras, mientras mayor sea el número de marcadores que contiene el contenedor menor probabilidad de detección y corrección de errores tendrá el marcador.

Una vez que se declara el diccionario, se hace uso de `drawMarker(dictionary, N, M, markerImage, 1))`, que cuenta con los siguientes parámetros:

- **`dictionary:`** diccionario declarado (`aruco.DICT_KxK_NumeroMarcadores`).
- **`N:`** identificador del marcador, depende del tamaño del contenedor, puede ser un valor entre 0 y 999 acorde al diccionario.
- **`M:`** tamaño de la imagen del marcador generado. El tamaño de un marcador es $M \times M$ píxeles. Se debe tener presente que el número de píxeles debe ser proporcional al número de bits empleados, por tal motivo se recomienda, que `M` sea al menos dos veces el número de bits.
- **`markerImage:`** imagen de salida.
- **`1:`** parámetro opcional que establece el ancho del borde negro del marcador, proporcional al número de bits.

Al emplear el diccionario de 6×6 de 250 valores, `N=5`, `M = 200` y ancho de borde por defecto, genera como resultado el marcador de la Figura 2.8.

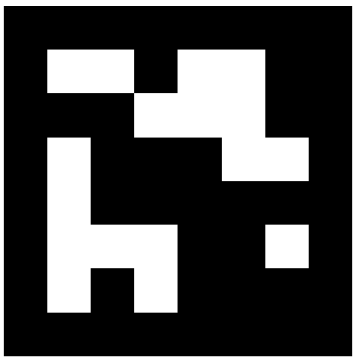


Figura 2.8: Marcador *ArUco* 6×6 N=5

2.11.1.3. Detección de marcadores *ArUco*

La detección de marcadores principalmente se basa en dos etapas encargadas de procesar y extraer datos [52]. En la primera etapa se aborda la detección de marcadores empleando un análisis de imágenes, que encuentra formas cuadradas que puedan ser interpretadas como el marco del marcador. Este proceso se encarga de filtrar, eliminar perspectiva, descartar formas convexas, y descartar contornos demasiado pequeños y demasiado grandes.

Por otro lado, la segunda etapa abarca el análisis de codificación de la matriz interna. Para poder obtener la información de manera correcta se requiere de una técnica de extracción de bits efectuando procesos para descartar, clasificar y seleccionar los bits blancos y negros de acuerdo a un umbral. Esta

selección emplea un subproceso de división de marcador en celdas iguales utilizando como medida el cuerpo del marco, y realiza un conteo de píxeles. El número de píxeles blancos y negros determina si la división realizada pertenece a un bit blanco (1) o negro (0).

2.11.1.4. Parámetros para detección de marcadores

- **Parámetros de contorno:** el proceso de detección de marcadores opta por mantener un equilibrio entre la capacidad de detección y el rendimiento. Empleando parámetros de contorno para descartar marcadores candidatos que no cumplen con un marco/contorno válido y clasificación de formas, encargada de filtrar formas semejantes a figuras cuadradas. Con el fin de que en etapas posteriores de interpretación de datos no influyan en el rendimiento del proceso.
- **Parámetros de extracción de bits:** una vez que se realiza la detección de contornos y se descarta marcadores erróneos, el algoritmo procede a la fase de extracción de información por medio de dos procesos: eliminación de perspectiva y extracción de bits.
  Para eliminar la perspectiva se separa el marcador del entorno que lo rodea empleando un umbral de *Otsu*. El método de *Otsu* se emplea para definir umbrales automáticos consecuente a la calidad de imágenes empleadas[53], es decir, este método retorna un umbral exclusivo que separa los píxeles en dos clases: contenido ilustrado y fondo [54].
  Luego del proceso de eliminación de perspectiva, se realiza una fragmentación acorde al número de celdas indicadas en el diccionario electo. El conteo de píxeles dentro de cada celda determina el tipo de bit idóneo, sea blanco o negro.

Capítulo 3

Trabajos Relacionados

En este capítulo se presentan los trabajos relacionados a la temática que se plantea en el presente trabajo de titulación. La Sección 3.1 presenta una descripción de diferentes trabajos centrados en el aterrizaje de precisión, y su aplicación en redes tolerantes a retrasos. A continuación, se presenta algunas contribuciones para la recolección eficiente de datos en redes de sensores usando vehículos aéreos no tripulados (Sección 3.2). El capítulo finaliza presentando algunas conclusiones sobre las contribuciones analizadas (Sección 3.3).

3.1. Aterrizaje de precisión

El protagonismo que los UAVs han alcanzado en el desarrollo e investigación de aplicaciones, se debe a su gran capacidad de adaptación a entornos hostiles. Tareas como la inspección de líneas de transmisión usando imágenes capturadas por una cámara a bordo [55] se logran gracias a la gran estabilidad, y retención de altitud del vehículo.

Dentro de este ámbito, existen diversas soluciones desarrolladas para lograr aterrizajes de UAVs de gran precisión. *Zhao* y *Jiang* [56] establecen una conexión *WiFi* entre UAV y un ordenador para desarrollar detección de bordes en las imágenes capturadas por la cámara a bordo, y utilizar herramientas de Robot Operating System (ROS) para el reconocimiento de marcas de realidad aumentada o Augmented Reality (AR), que determinen el error de posición en dos ejes para controlar el descenso del dron usando un control Proporcional-Integral-Derivativo (PID).

Sudevan y colaboradores [57] implementan un algoritmo de búsqueda semi-autónoma, que detecta puntos clave y computa descriptores para definir marcas de aterrizaje. Los algoritmos usados por los autores buscan coincidencias de los descriptores en los *frames* capturados por una cámara a bordo, y se generan señales de velocidad para el controlador PID de vuelo. *Smyczynski* y colaboradores [58] emplean una *Raspberry Pi 3*, en donde se combina ROS con la librería OpenCV para realizar detección de bordes (algoritmo *Canny Edge*), y el cálculo de los momentos invariantes de *Hu* para detectar la plataforma de aterrizaje (marca) e intercambiar parámetros de vuelo con el controlador *Pixhawk*.

Janousek y *Marcon* [59] utilizan una cámara de infrarrojos o Infrared (IR), para detectar una

plataforma de aterrizaje (objeto marcado) especialmente diseñada con diodos IR, cuya lectura es capturada por la cámara a bordo y procesada para establecer señales de velocidad para el controlador de vuelo. De forma similar, *Putra* y colaboradores [60] utilizan una cámara para detección de objetos basados en umbral de color Hue, Saturation, Value (HSV) y un control PID. Sin embargo, introducen un filtro complementario para determinar altitud, posición del objeto (plataforma de aterrizaje) e inhibir los efectos del canal.

Sani y *Karimian* [61] utilizan una cámara frontal para obtener dos estimaciones de posición del UAV. La estimación por visión, detecta la posición de un marcador AR utilizando la librería *ArUco* basada en OpenCV, y la estimación por sensores utiliza un filtro de *Kalman* para mejorar las mediciones obtenidas por los sensores de velocidad, aceleración y ángulo del dron. El proceso de aterrizaje se efectúa usando retroalimentación del conjunto marcador-cámara vertical. El aterrizaje de precisión en un UAV puede emplearse para lograr tareas como reemplazar la batería de forma automática [62].

El control preciso de maniobra que los drones ofrecen al ser dirigidos por un computador complementario o una GCS en ambientes hostiles, ha posibilitado la integración con redes tolerantes a retrasos. Un esquema de red DTN, que integra el uso de UAVs se plantea en [63] para la retransmisión de información de sensores, y para la búsqueda de nodos en desastres naturales en [64].

3.2. Recolección de datos

Las WSNs presentan un escenario oportuno para el desarrollo e implementación de algoritmos de recolección eficiente de datos. Un algoritmo eficiente implica optimización de algún proceso o recurso en un sistema o función.

Al emplear vehículos aéreos no tripulados para la recolección de datos en redes que cubren amplias áreas geográficas, determinar la ruta o trayectoria de vuelo más corta, eficiente en términos de consumo de energía y cantidad de nodos alcanzados es el propósito final. *Caillout* y colaboradores presentan en [65] un método basado en algoritmos heurísticos para resolver el problema de recolección aérea de datos. Se plantea un escenario donde existe una red de drones, que cuentan con dos tipos de comunicación: aire-aire (*air-to-air*) y aire-tierra (*air-to-ground*), nodos objetivo móviles, y una estación base. Aprovechan la movilidad de los objetivos para minimizar la ruta de saltos calculada hacia la estación base empleando el algoritmo *Dijkstra*. Se garantiza la comunicación aire-aire manteniendo la distancia entre pares de UAVs dentro de un rango de comunicación establecido. Asimismo, se garantiza la comunicación aire-tierra minimizando la altitud del vehículo.

Wang y colaboradores establecen que una red de sensores eficiente es aquella que sigue reglas para distribuir sus nodos [66]. Definen un marco de referencia que incluye el despliegue de red, posicionamiento de nodos, búsqueda de puntos de anclaje, planificación de ruta rápida, y recolección de datos. La red cuenta con dos tipos de nodos: *beacon* y *unknown*. El nodo guía (*beacon*) cuenta con un módulo GPS, mientras que los nodos *unknown* se ubican empleando el criterio de intensidad de señal para transmisión de un solo salto. La ruta rápida se calcula en base a una planificación basada en división de cuadrículas, donde se define una ruta primaria y se minimiza la distancia que pasa por todos los nodos guía. La recolección de datos se realiza a través de los nodos guía considerando el enrutamiento de un salto.

La operación de la WSN y un sistema aéreo no tripulado o Unmanned Air/Aircraft System (UAS) se emplean en [67] para mejorar el rendimiento en la recolección de datos. La red de sensores está

organizada siguiendo el principio de grupos. Se definen clúster de nodos, cada clúster asigna un nodo principal Cluster Head (CH) encargado de transmitir información al UAS. El resto de nodos en el clúster se denominan Cluster Member (CM). Se opera con la premisa de que cada nodo puede ajustar su potencia de transmisión, por consiguiente, puede aumentar o reducir su rango de comunicación. La eficiencia de la operación de la red se alcanza con el proceso de partición y rotación de la red. La partición de la red permite que ésta se organice de manera autónoma. Se selecciona el nodo CH empleando un esquema de pesos en función al porcentaje de batería, y una proporción entre la potencia de señal transmitida y la potencia de señal recibida entre pares de nodos dentro del clúster. El nodo CH candidato con mayor peso será seleccionado como nodo principal del clúster.

La planificación de vuelo del UAS a través de la red se define como el problema del vendedor o Travelling Salesman Problem (TSP). La ruta obtenida reduce la distancia total en una trayectoria definida con un clúster inicial y final. La ruta se actualiza cuando un nodo CH desencadena una rotación CH. Este proceso se produce cuando la batería del nodo CH es menor a un umbral establecido. La selección evalúa la distancia que la trayectoria actual del vehículo atraviesa en la zona de cobertura del nodo candidato. Una distancia mayor implica cambios menores en el plan de vuelo.

3.3. Conclusiones

Una de las principales limitantes que se encuentra implícita en la optimización de recursos para la recolección de datos, en las contribuciones presentadas, es la trayectoria o ruta de vuelo. Al minimizar la distancia que cubre el vehículo al atravesar los nodos de red, el tiempo de vuelo, la distancia total, y por ende, el número total de nodos visitados es el máximo alcanzable con las prestaciones del vehículo. De una u otra forma, la recolección eficiente se basa en optimizar el consumo de batería, minimizando la distancia total recorrida por el vehículo.

La incorporación de drones en la recolección de datos en redes amplias puede abordarse desde diversos puntos de vista. El propósito inicial de la recolección es emplear de manera eficiente el dron para establecer un sistema de recolección que cubra toda la red. Si bien la recolección de datos abordada en la literatura investigada en este documento pretende optimizar la asignación y consumo de recursos en toda la red, ninguna aplicación implementa el aterrizaje como opción de optimización de consumo de batería en el UAV. Este ahorro en el consumo de energía resulta significativo cuando el número de nodos a visitar es grande. Además, aterrizar puede complementar procesos como reemplazo y/o carga de la batería incorporada en el vehículo.

Capítulo 4

Diseño e Implementación

En este capítulo se presenta el diseño e implementación del sistema de control de aterrizaje para recolección de datos empleando un UAV. El capítulo inicia definiendo el montaje y configuración del vehículo empleado (Sección 4.1). A continuación, se presentan los algoritmos involucrados en la calibración de la cámara y en la detección del marcador *ArUco* (Sección 4.2). Luego, se presentan los procesos y parámetros implicados en los algoritmos que controlan el movimiento del dron (Sección 4.3). El capítulo finaliza presentando los componentes y el proceso de transmisión y recepción de datos en la red DTN desplegada (Sección 4.4).

4.1. Montaje y configuración del vehículo

En esta sección se detalla el *hardware* que se utilizó para el montaje del vehículo y la placa SBC empleada (Subsección 4.1.1). Además, se presentan las versiones de *firmware* con las que se experimentó durante la implementación y algunas observaciones de rendimiento (Subsección 4.1.2).

4.1.1. Hardware

El UAV empleado en el desarrollo de este trabajo es un drone armable, que cuenta con un controlador *PixRaptor*, una versión genérica del controlador *Pixhawk*, compatible con *Ardupilot* y *PX4*. Los distintos componentes de *hardware* pueden ser personalizados cuando sea necesario. Se optó por un vehículo con cuatro unidades de propulsión o *quadcopter* en configuración *X*. La Figura 4.1 presenta un diagrama de conexión de todos los elementos que conforman el vehículo.

Cada elemento especifica el puerto al que se conecta en el controlador de vuelo. Además, se presenta la computadora complementaria, y su conexión con el controlador de vuelo y la cámara. El consumo de potencia requerido por la computadora complementaria al efectuar procesamiento de imágenes precisa el uso de dos baterías en el vehículo, una destinada para el consumo exclusivo de la cámara y la SBC, y otra para el resto de elementos en el vehículo.

Las líneas de color rojo y negro representan la distribución principal de potencia en el dron. El resto

de elementos obtiene energía a través del controlador. Un descripción más detallada de la conexión de los componentes del *quadcopter* se presenta en el Anexo A.

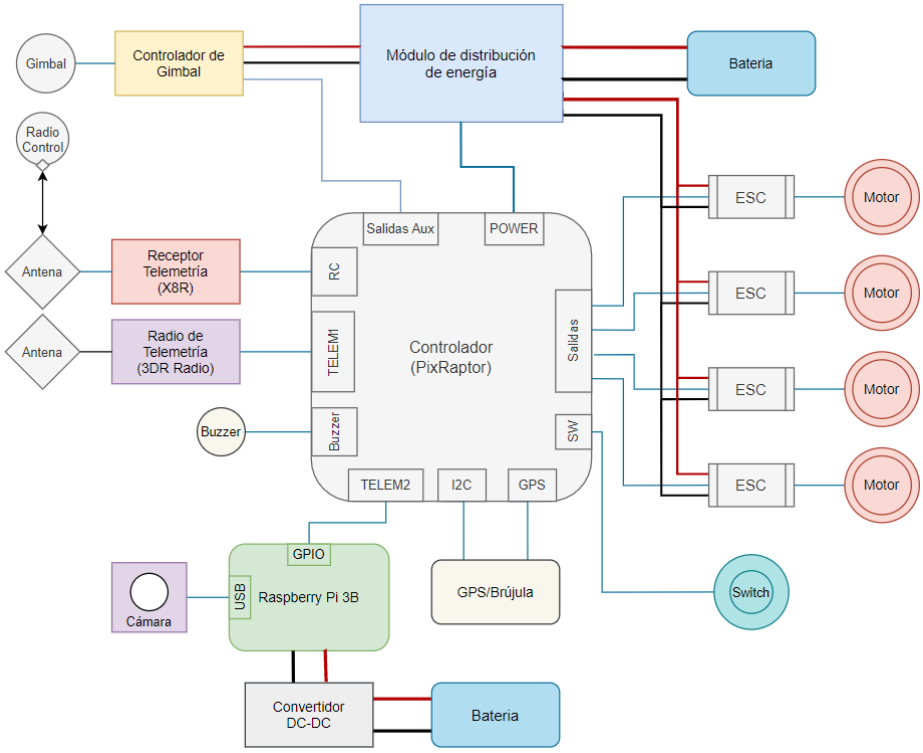


Figura 4.1: Diagrama de bloques del sistema

La Tabla 4.1 presenta las especificaciones de los elementos que conforman el *quadcopter*.

La SBC empleada en el desarrollo de este proyecto es una *Raspberry Pi 3B*. Los algoritmos desarrollados están diseñados para que se ejecuten de forma directa sobre la SBC. Por ende, es necesario que la placa se encuentre sujeta al vehículo durante todo el proceso. Para ello, se optó por diseñar una estructura de sujeción al vehículo para la SBC.

Tabla 4.1: Especificación de componentes del dron

Elemento	Tipo	Descripción
Cuadro	*Carbon HJ-H4 reptile*	Marco plegable de fibra de carbono con tren de aterrizaje
Controlador de vuelo	*PixRaptor*	Control de vuelo alterno de *PixHawk*
GPS+Brujula	*Ublox M8N*	Sistema de posicionamiento y orientación
Radio de telemetría *Sik*	*3DR* Radio V2	Conexión de telemetría entre el controlador y una estación terrestre
Receptor de telemetría	*X FrSky X8R*	Receptor compatible con controles *Taranis*
ESC	*Mystery 30A M-30*	Controladores compatibles con baterías LiPo 2s y 3s
Motores	*Multistar Elite 3508*	Motor eléctrico sin escobillas de 268 RPM/V
Buzzer + Switch	Alarma *TEAEGG BIP* con interruptor	Indicar de forma audible los cambios de estado del vehículo y bloqueo de motores
Módulo de alimentación	*3DR FCMODEL*	Distribuidor de energía
Control remoto	Transmisor *Taranis X9D*	Transmisor compatible con *FrSky Series X*
Gimbal	*Tarot T-2D*	Estabilizador de cámara de 2 ejes (*roll, tilt*)
Cámara Dron	*Turnigy HD WiFi*	Cámara compacta , ligera y resistente
Hélices	*RCtimer TM 12x4.0”*	Propelas de fibra de carbono
Batería Dron	*Wild Scorpion NANO*	Batería 3S 11.1 voltios 4200mAh
Raspberry Pi 3	Modelo B, 1GB RAM	Ordenador de placa reducida
Batería *Raspberry Pi*	*Turngy Nano-Tech*	Batería 2S 7.4 voltios 3500 mAh
Convertidor DC/DC	*XL4015 5A*	DC/DC Reductor ajustable
Cámara *Raspberry Pi*	Módulo versión 2	Módulo optimizado para la SBC

Al emplear una placa *Raspberry Pi*, se consideró en primera instancia emplear como dispositivo de captura un módulo de cámara versión 2 de la misma fundación, puesto que, se encuentra optimizado para el uso exclusivo con este tipo de placas. Por otro lado, como segunda opción se consideró la cámara *Turnigy HD Wifi Action*, debido a su compatibilidad con el *gimbal* del dron. La Sección 5.1 presenta una comparación de rendimiento de las dos cámaras.

Empleando software Computer-Aided Design (CAD) se diseñó una caja y una base de sujeción. La Figura 4.2 presenta la estructura de sujeción diseñada. Cada una de las piezas que conforman

la estructura se construyeron en una impresora 3D con filamento ComPolylactic acid (PLA) como material plástico.

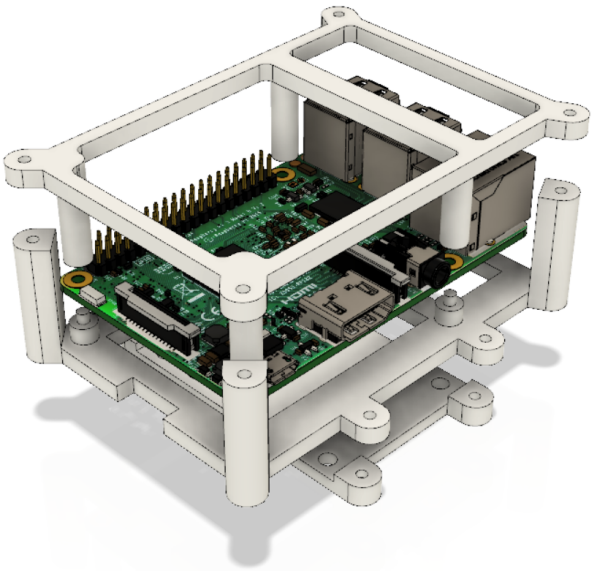


Figura 4.2: Estructura de sujeción para *Raspberry Pi 3B*

En la parte inferior de la Figura 4.2 se encuentra la base de sujeción para la caja de la *Raspberry Pi*. Esta pieza tiene el propósito de anclar la caja a la parte superior del *frame* del dron. Las vistas y medidas de esta pieza se presentan en la Figura 4.3.

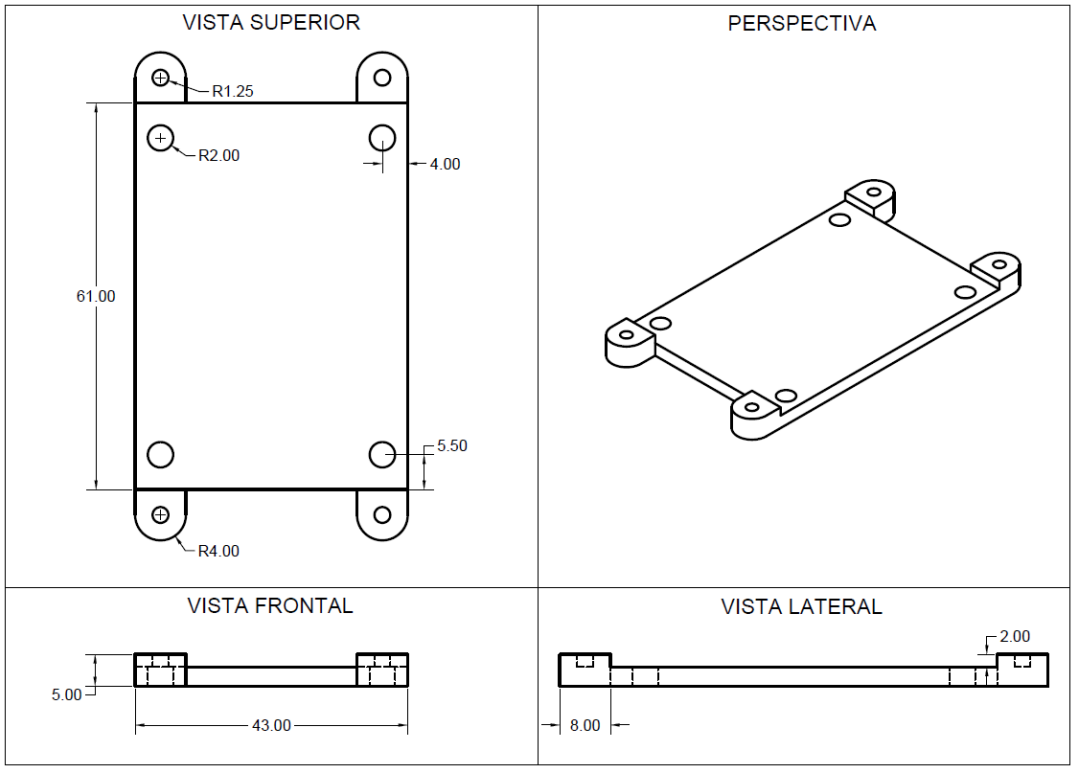


Figura 4.3: Base de sujeción al UAV

La caja de la *Raspberry Pi* consta de dos piezas. La parte inferior sirve de soporte/base para la placa y se une a la base de sujeción en el *frame* mediante cuatro tornillos M3 dispuestos en los laterales de la placa. Por otro lado, la parte superior se encarga de mantener fija la placa mediante cuatro tornillos M3 dispuestos en los extremos de la pieza, que se usen a la parte inferior de la caja. Las vistas y medidas de la pieza superior e inferior se presentan en las Figuras 4.4 y 4.5, respectivamente.

Los diseños se basaron en las medidas expuestas en los archivos de diseño mecánico de las placas de la Fundación *Rasperry Pi* y del diseño 3D de *Mechatronics Art*.

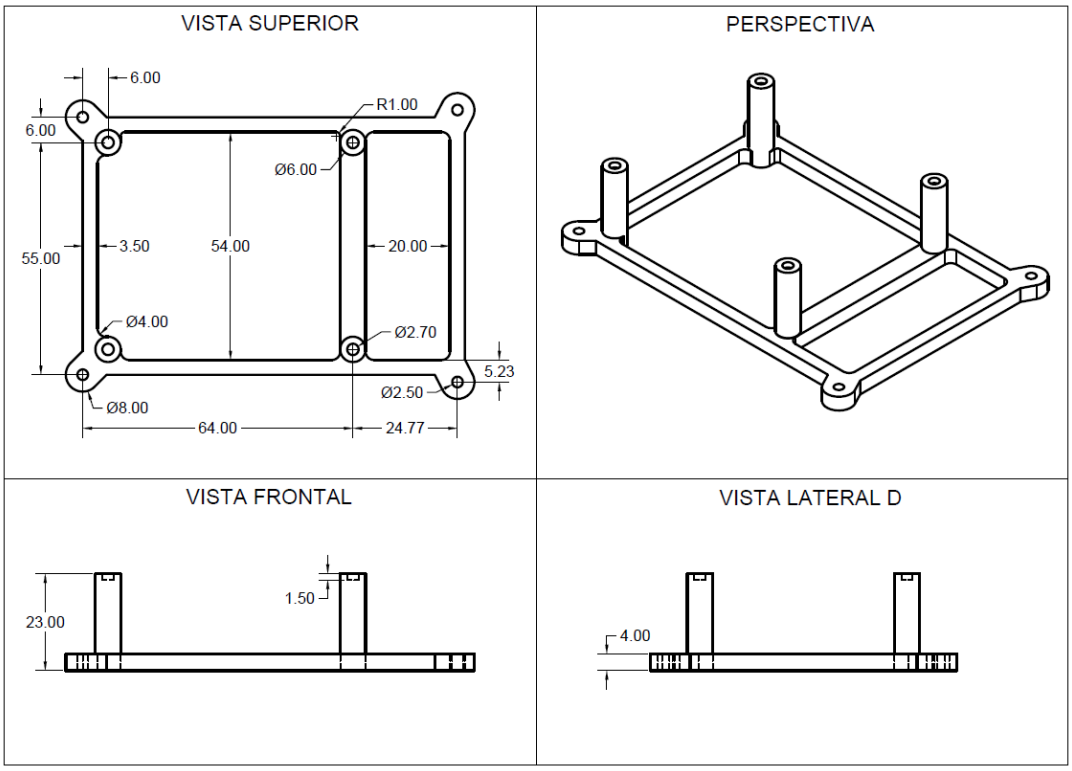


Figura 4.4: Parte superior de la caja para *Raspberry Pi 3B*

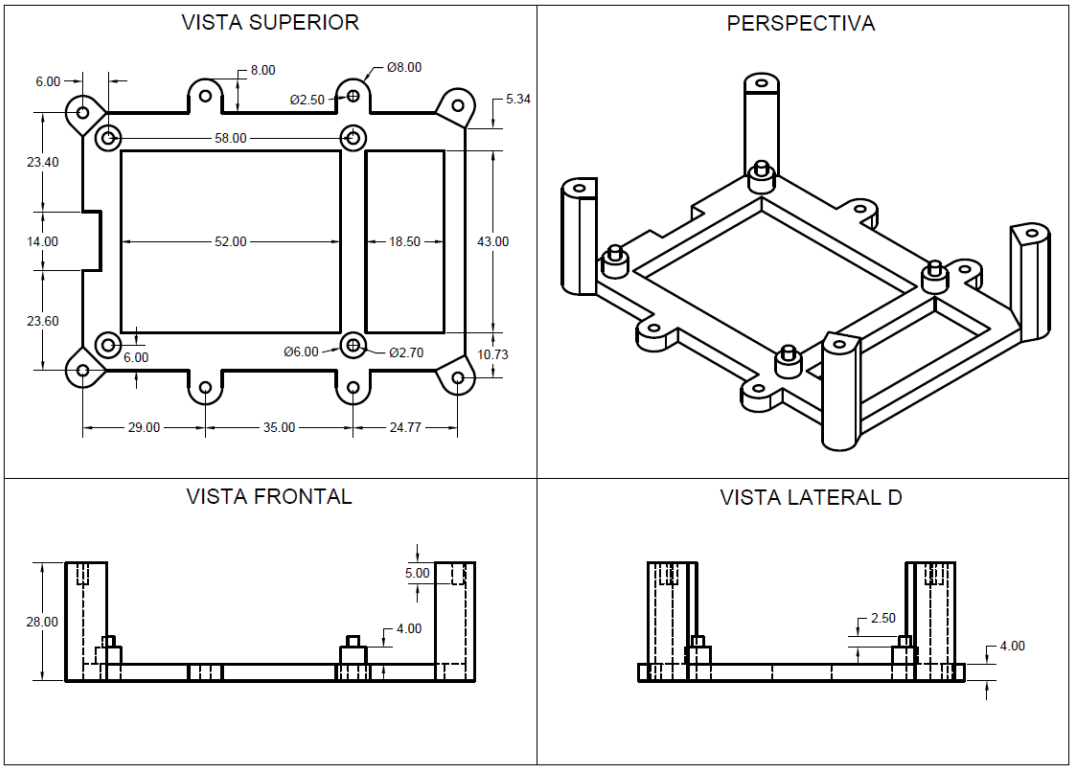


Figura 4.5: Parte inferior de la caja para *Raspberry Pi 3B*

4.1.2. Firmware

Para el desarrollo de este trabajo, se experimentó con cuatro diferentes versiones de *firmware* tanto de *Ardupilot* como de *PX4*. La Tabla 4.2 presenta las distintas versiones de *firmware* instaladas en el controlador, además de observaciones de rendimiento y compatibilidad con los diferentes dispositivos que conforman el vehículo.

Tabla 4.2: Rendimiento y compatibilidad de *firmware* en el vehículo

Firmware	Versión	Observaciones
PX4	1.19	Ofrece diferentes modos de vuelo que *Ardupilot* No permite calibrar algunos sensores Solo cuenta con soporte básico con *Dronekit*
ArduCopter	3.4.6	No ofrece soporte para ESC *BLHeli*
ArduCopter	3.6.11	El radio de telemetría no se conecta con MP
ArduCopter	3.5.5	Versión más actual que permite la conexión sin errores del radio de telemetría con MP

Todas las versiones de *firmware* son estables. *ArduCopter* v3.5.5 es el *firmware* más actual, que se pudo cargar en el controlador *PixRaptor*. Esta versión no produce errores de conexión ni de compatibilidad con los elementos del vehículo.

4.2. Detección de plataforma de aterrizaje

Una SBC es empleada para adquirir, procesar y analizar imágenes digitales con la finalidad de extraer información y producir datos objetivos empleados en la toma de decisiones. En este proyecto se hará uso de esta herramienta con el fin de proporcionar una retroalimentación de posicionamiento al momento de descender el UAV. Además, la SBC se configuró como punto de acceso destinado al monitoreo de procesos en el desarrollo de la implementación.

4.2.1. Calibración de cámara

La gran variedad de tipos de cámara pueden capturar imágenes de diversas maneras. Sin embargo, no todas proporcionan la misma forma y calidad de imagen. En algunos casos, las imágenes presentan distorsión radial y tangencial alterando la forma de la escena capturada. En la distorsión radial las líneas rectas aparecen curvadas a medida que se alejan del foco central de la imagen. En este caso se puede definir dos tipos de distorsión radial:

- *Distorsión radial positiva* conocida como distorsión de barril (*barrel distortion*), donde las líneas presentan una curvatura de manera convexa, es decir, las líneas presentan una curvatura desde el centro hacia fuera.
- *Distorsión radial negativa* conocida como distorsión de alfiler (*pincushion distortion*), donde las líneas presentan una curvatura de manera cóncava, es decir, las líneas presentan una curvatura desde afuera hacia el centro.

Por otra parte, la distorsión tangencial ocurre cuando la captura de imágenes no se encuentra alineada paralelamente con el plano de captura, debido a esto, ciertas áreas de una imagen pueden verse más cercanas que otras.

Una cámara estenopeica describe la relación matemática entre las coordenadas de un punto en el espacio tridimensional y su proyección en el plano de captura de la cámara, el uso de este tipo de cámaras evita distorsiones geométricas o desenfoque de objetos provocado por lentes y aperturas de tamaño finito[68]. El modelo de una cámara estenopeica es usado como una aproximación del mapeo de una escena 3D en una imagen 2D mediante el uso de parámetros intrínsecos y extrínsecos[69][70], que toman la forma de la Ecuación (4.1).

$$\vec{m}' = \bar{A}[\bar{R}|\vec{t}]\vec{M}' \tag{4.1}$$

La matriz de parámetros intrínsecos expuesta en la Ecuación (4.2) no depende de la escena. Por lo tanto, cuando se evalúa una vez, se puede reutilizar para diferentes escenas. Los parámetros que conforman esta matriz son:

- (f_x, f_y): distancias focales expresadas en píxeles
- (c_x, c_y): punto central de la imagen

$$\bar{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

La matriz de parámetros extrínsecos o matriz de rotación-traslación $\bar{R}|\vec{t}$ se emplea para describir el movimiento de la cámara en una escena estática o describir el movimiento de un objeto frente a una cámara fija. Por lo tanto, la matriz de parámetros extrínsecos es empleada como diccionario, cuya función es traducir las coordenadas (X, Y, Z) de un punto M de una escena, a un punto m en el sistema de coordenadas (x, y) de la cámara empleada. La Ecuación (4.3) presenta la matriz de parámetros extrínseco, el vector M y el vector m.

$$\begin{aligned} \bar{R}|\vec{t} &= \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \\ \vec{M}' &= \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \\ \vec{m}' &= \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \end{aligned} \tag{4.3}$$

OpenCV ofrece funciones que permiten determinar la matriz de parámetros intrínsecos en conjunto con el vector de coeficientes de distorsión radial y tangencial de cualquier cámara compatible con la librería. Para corregir la distorsión radial emplea la Ecuación (4.4) para el eje x y la Ecuación (4.5) para el eje y.

$$x_{\text{corregido}} = x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{4.4}$$

$$y_{\text{corregido}} = y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{4.5}$$

Por otro lado, para corregir la distorsión tangencial utiliza la Ecuación (4.7) para el eje x y la Ecuación (4.7) para el eje y.

$$x_{\text{corregido}} = x' + [2p_1 x' y' + p_2(r^2 + 2x'^2)] \tag{4.6}$$

$$y_{\text{corregido}} = y' + [2p_2 x' y' + p_1(r^2 + 2y'^2)] \tag{4.7}$$

Donde, intervienen las siguientes variables:

- x, y: coordenadas de píxeles no distorsionados
- x', y': coordenadas de imagen normalizadas, calculadas a partir de la Ecuación (4.8) y (4.9), donde se divide las coordenadas de píxeles desde el centro óptico entre la distancia focal en píxeles. Por tanto, estas coordenadas son adimensionales
- k_1, k_2, k_3: coeficientes de distorsión radial de la cámara
- p_1, p_2: coeficientes de distorsión tangencial de la cámara
- $r^2 = x^2 + y^2$

$$x' = \frac{x}{z} \tag{4.8}$$

$$y' = \frac{y}{z} \tag{4.9}$$

Algunos lentes presentan distorsión radial y una leve distorsión tangencial. OpenCV corrige la distorsión radial y tangencial empleando el modelo general expresado en las Ecuaciones (4.10) y (4.11).

$$x'' = x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2) \tag{4.10}$$

$$y'' = y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_2 x' y' + p_1(r^2 + 2y^2) \tag{4.11}$$

Las coordenadas del punto de proyección (u,v) de la Ecuación (4.1) se determinan empleando la Ecuación (4.12) y la Ecuación (4.13).

$$u = f_x * x'' + c_x \tag{4.12}$$

$$v = f_y * y'' + c_y \tag{4.13}$$

4.2.1.1. Captura de imágenes

El objetivo de la calibración es obtener una imagen plana. Este proceso requiere de un número de imágenes de prueba que permitan extraer características de la cámara. Estas imágenes deben ser tomadas desde diferentes lugares y orientaciones con la cámara estática. Se requiere el uso de al menos diez imágenes para obtener una matriz confiable de coeficientes intrínsecos[71].

Las imágenes empleadas para el proceso de calibración requieren de un elemento de referencia que facilite la extracción de características. Para este propósito, es común el uso de patrones cuadriculados similares a un tablero de ajedrez. La Figura 4.6 presenta el tablero empleado en la calibración.

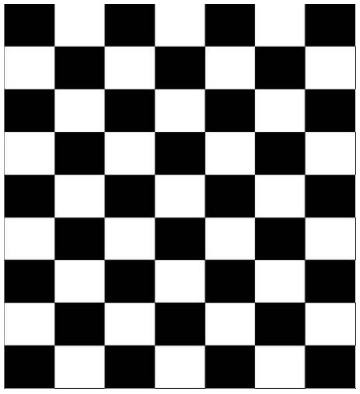


Figura 4.6: Tablero de calibración

El Código 4.1 se utiliza para capturar un fotograma desde una cámara y almacenarlo en el disco empleando OpenCV. El parámetro `0` empleado por el método `VideoCapture` es la entrada de video por defecto del sistema. Se usan los métodos `imshow` y `imwrite` para mostrar el fotograma en pantalla y escribirlo en el disco.

```
cap = cv2.VideoCapture(0)
ret, frame = cap.read()
cv2.imshow('camara', frame)
cv2.imwrite('captura.png', frame)
cap.release()
cv2.destroyAllWindows()
```

Código 4.1: Captura de imágenes usando *OpenCV*

Colocar el tablero de calibración en diferentes disposiciones, permite determinar los puntos en donde se presenta mayor distorsión asociados a un objetivo específico. La Figura 4.7 presenta algunas posturas de referencia para la captura de imágenes de calibración. Estás posturas describen la ubicación del marcador dentro del fotograma capturado (posición), relativo al dispositivo de captura (profundidad) y relativo al plano de captura (inclinación).

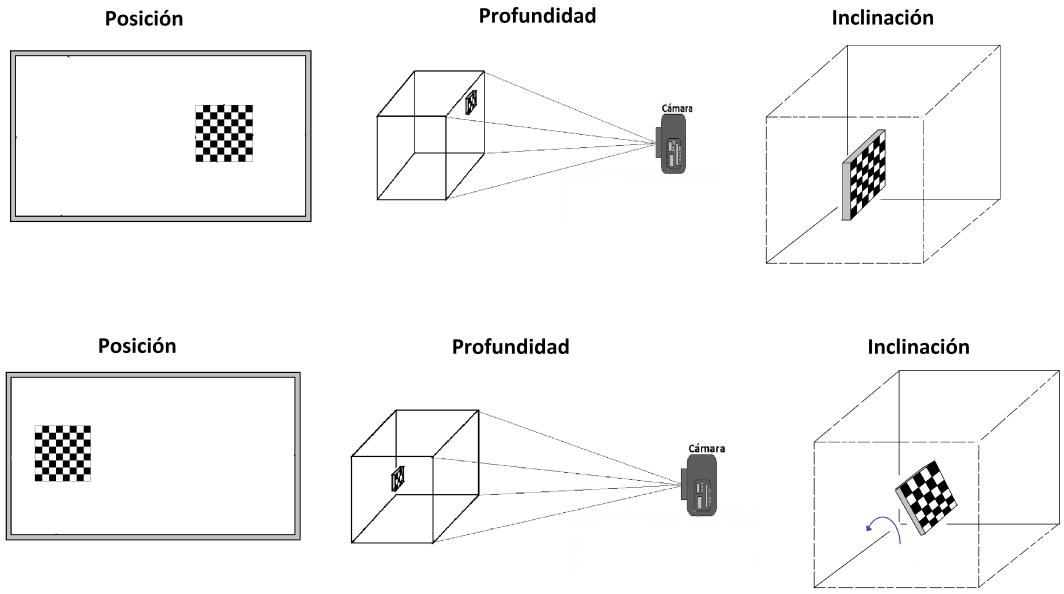


Figura 4.7: Posturas de referencia para captura de imágenes de calibración

La Figura 4.8 presenta algunas capturas del tablero de calibración modificando su posición, profundidad e inclinación.

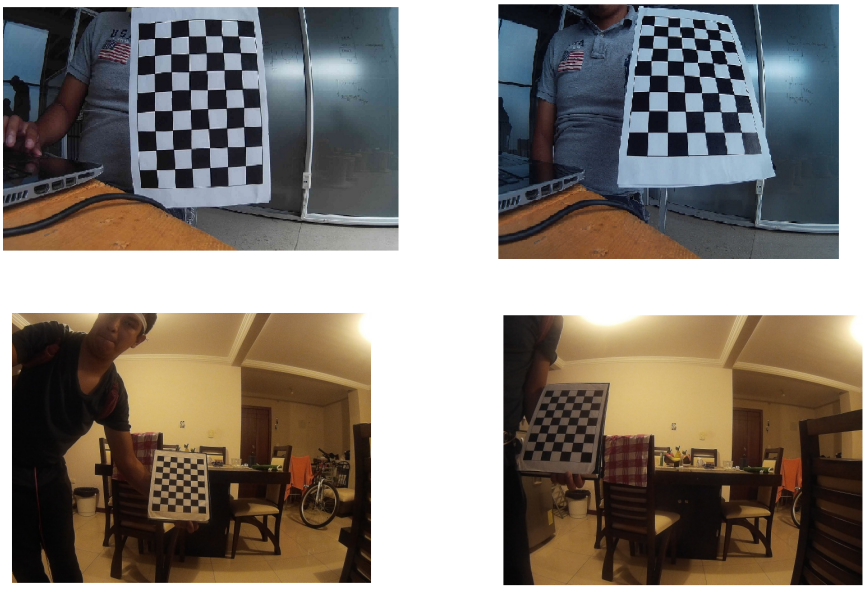


Figura 4.8: Capturas del tablero de calibración

4.2.1.2. Extracción de matriz característica y vector de distorsión de la cámara

El proceso de calibración tiene como objetivo encontrar la matriz de propiedades intrínsecas así como el vector de distorsión de la cámara. Este proceso se basa en la obtención de vértices en la imagen

distorsionada, y la reubicación de estos para aproximar la forma del patrón de referencia. De este modo, se puede obtener una estimación de la distorsión presente en un área específica.

El Código 4.2 presenta el proceso para la obtención de la matriz intrínseca y el vector de distorsión. El código establece un bucle que inicia con la lectura de una captura del tablero de calibración (`fname`), y la coloca en escala de grises. Luego, busca los vértices de los cuadrados dentro del tablero de $N \times M$ cuadrados (`findChessboardCorners`) y los almacena (`corners`). A continuación, intenta redefinir los vértices obtenidos de manera que se ubiquen equidistantes entre sí (`cornerSubPix`). La reubicación de vértices se desarrolla en un proceso iterativo que obedece a un criterio de finalización basado en un valor de exactitud, *epsilon* (`EPS`) o número máximo de iteraciones (`MAX_ITER`). Para finalizar, la matriz intrínseca y el vector de distorsión se obtienen empleando el método `calibrateCamera` basado en el trabajo desarrollado por *Zhang* [70].

```
criterio = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, MAX_ITER,
        EPS)
objp = np.zeros((m*n,3), np.float32)
objp[:,:2] = np.mgrid[0:m,0:n].T.reshape(-1,2)
puntosdeobjeto = []
puntosdeimagen = []
images = glob.glob('*.jpg')
for fname in images:
 imagen = cv2.imread(fname)
 gris = cv2.cvtColor(imagen,cv2.COLOR_BGR2GRAY)
 ret, corners = cv2.findChessboardCorners(gris, (m,n),None)
 if ret == True:
  puntosdeobjeto.append(objp)
  corners2 = cv2.cornerSubPix(gris,corners,(11,11),(-1,-1),criterio)
  puntosdeimagen.append(corners2,ret)
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(puntosdeobjeto,
        puntosdeimagen,gris.shape[::-1],None,None)
```

Código 4.2: Cálculo de matriz intrínseca y vector de distorsión

4.2.2. Detección de marcador *ArUco*

Para el desarrollo de este trabajo se recurrió a diferentes dispositivos de captura de imágenes. Algunos de estos dispositivos cuentan con un lente gran angular que permiten tener un mayor ángulo de visión en las capturas realizadas. Por ende, es necesario eliminar la distorsión provocada por este tipo de dispositivos previo al proceso de detección del marcador *ArUco*. OpenCV emplea funciones desarrolladas por la librería de Garrido y Jurado [52] para la detección de marcadores *ArUco*. Para la detección de marcadores *ArUco* empleando OpenCV se usa el Código 4.3.

```
cap = cv2.VideoCapture(0)
ret, img =cap.read()
image = eliminar_distorsion(img, mtx, dist)
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters = aruco.DetectorParameters_create()
```

```
corners, ids, rejectedImgPoints = aruco.detectMarkers(image, aruco_dict,
        parameters=parameters)
image = aruco.drawDetectedMarkers(image, corners, ids)
cv2.imshow('frame',image)
cap.release()
cv2.destroyAllWindows()
```

Código 4.3: Detección de marcador *ArUco*

El proceso de detección requiere la declaración de un diccionario acorde al tamaño de la matriz interna del marcador y el número de identificadores que almacena (`DICT_6X6_250`). En este caso, se emplea un diccionario para una matriz de 6×6 que almacena 250 identificadores. La decodificación de la matriz binaria contenida en el marcador se logra a través de parámetros generados por el diccionario establecido (`DetectorParameters_create`).

Basado en la información del diccionario y los parámetros de decodificación establecidos, OpenCV es capaz de detectar (`detectMarkers`) uno o varios marcadores en una imagen. Cada marcador está asociado a un *ID* y las coordenadas de sus vértices.

La distorsión radial y tangencial presente en la imagen se puede reducir reubicando píxeles. El Código 4.4 se usa para reducir la distorsión de una imagen empleando la matriz intrínseca, el vector de distorsión (Código 4.2) y el modelo general de corrección expuesto en las Ecuaciones (4.10) y (4.11) (`undistor`). Además, se determina el área en donde la imagen no presenta distorsión (`roi`), que será empleada para estimar las coordenadas del punto original a partir de las coordenadas del punto distorsionado (`calibrada`).

```
def eliminar_distorsion(frame, matriz, vector):
  h, w = frame.shape[:2]
  newcammtx, roi=cv2.getOptimalNewCameraMatrix(matriz,vector,(w,h),1,(w,h))
  calibrada = cv2.undistort(frame, matriz, vector, None, newcammtx)
  x,y,w,h = roi
  calibrada = calibrada[y:y+h, x:x+w]
  return calibrada
```

Código 4.4: Eliminar distorsion de una imagen

4.3. Algoritmos de vuelo

La orientación de un UAV respecto a su centro de masa se define empleando tres ángulos: cabeceo (*pitch*), balanceo (*roll*), y guiñada (*yaw*). La Figura 4.9 ilustra la ubicación de estos ángulos en un *quadcopter* en configuración X, donde la nariz (*nose*) es la parte frontal del vehículo. La combinación resultante de modificar la velocidad de los rotores, se traduce en traslación del UAV.

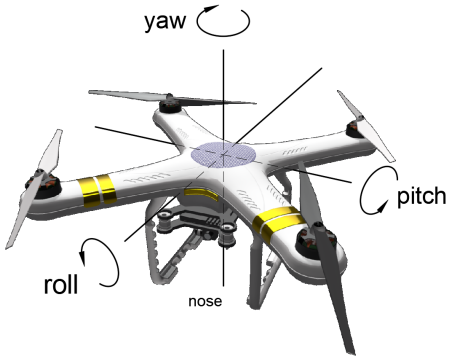


Figura 4.9: Ángulo *pitch*, *roll* y *yaw* en un *quadcopter*

La posición y movimiento del vehículo en el espacio se puede describir empleando dos sistemas de coordenadas.

- ***XYZ*:** definido sobre el cuerpo del *quadcopter*. El eje X es el eje longitudinal que apunta hacia la nariz del vehículo, el eje Y es el eje lateral que apunta hacia la parte derecha del dron, y el eje Z es el eje vertical que apunta hacia abajo. Este sistema se mueve y gira junto con el vehículo.
- ***Norte-Este-Abajo*:** o North-East-Down (NED), definido respecto a tierra [72].

La Figura 4.10 ilustra los dos sistemas de coordenadas.

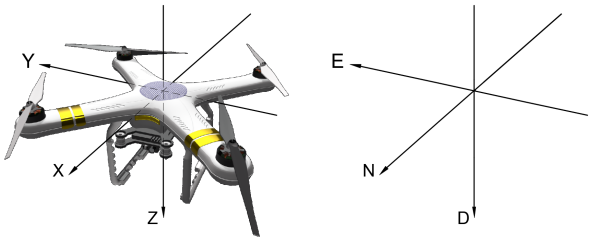


Figura 4.10: Sistema de coordenadas XYZ y NED

En el presente trabajo, el proceso de vuelo se encuentra conformado por varios algoritmos, que efectúan procedimientos específicos para desplegar un plan de vuelo capaz de atravesar por toda la red de sensores, y regresar al lugar de despegue efectuando tanto el aterrizaje como la recolección de datos en cada nodo.

4.3.1. Recolección de datos

El algoritmo de recolección de datos se encarga de establecer el plan de vuelo del vehículo a través de la red. Un nodo dentro de la red puede ser distinguido usando un identificador de red, o empleando el identificador del marcador *ArUco* asociado. Sin embargo, estos indicadores se encuentran definidos de forma externa al *firmware* del controlador de vuelo.

El vehículo emplea la información provista por la brújula y el módulo GPS para dirigirse hacia un punto. Un punto de interés o *waypoint* está definido por sus valores angulares (latitud y longitud) del sistema de coordenadas geográficas. En consecuencia, para definir el plan de vuelo del UAV, es necesario conocer la información de referencia de los puntos de interés que se desea alcanzar.

El Algoritmo 1 presenta la secuencia de procesos que el piloto automático debe realizar para alcanzar la ubicación de los nodos, recolectar datos y regresar al punto de partida. Inicialmente, el algoritmo obtiene la ubicación actual del vehículo (Algoritmo 1, línea 1). Este punto será tomado como `HOME`, punto de despegue o ubicación del servidor. A continuación, para un nodo específico dentro de la lista de puntos de referencia o ubicaciones de nodos (`ListaWayPoints`), el vehículo debe completar el proceso de `ArmadoyDespegue` (Algoritmo 1, línea 3). Al alcanzar una altura definida (`AlturaVuelo`), el piloto automático emplea las coordenadas geográficas de un *waypoint* para iniciar su recorrido empleando la función `DirigirVehiculo` (Algoritmo 1, línea 4). El vehículo no debe realizar otro proceso mientras no alcance la ubicación de interés. Debido a la precisión del sistema GPS es imposible referenciar un punto exacto, por ello, existe un margen de error permitido (`MargenErrorWP`) al estimar la distancia entre el punto de ubicación actual del vehículo y el *waypoint* al que se dirige (Algoritmo 1, líneas 5 a 9).

Cuando el vehículo alcanza el punto de interés, se inician los algoritmos complementarios, que garantizan aterrizaje de precisión y recolección de datos en un nodo (Algoritmo 1, línea 11). Finalmente, al completar el proceso antes descrito para cada nodo de la red, el UAV inicia el proceso de `ArmadoyDespegue`, emplea el *waypoint* `HOME` y regresa al lugar de despegue, donde `Aterriza` y efectúa el proceso de `Desarmado` (Algoritmo 1, líneas 13 a 16).

Algoritmo 1: Recolección de datos

```
   Datos: AlturaVuelo, ListaWayPoints, MargenErrorWP
 1 HOME = UbicacionActual()
 2 para WP en ListaWayPoints hacer
 3 |   ArmadoYDespegue(AlturaVuelo)
 4 |   DirigirVehiculo(WP)
 5 |   mientras VERDADERO hacer
 6 |   |   dist=ObtenerDistancia(UbicacionActual, WP)
 7 |   |   si dist <= MargenErrorWp entonces
 8 |   |   |   salir
 9 |   |   fin
10 |   fin
11 |   Siguiente algoritmo...
12 fin
13 ArmadoYDespegue(AlturaVuelo)
14 DirigirVehiculo(HOME)
15 Aterrizar()
16 DesarmarVehículo()
```

La implementación se desarrolla empleando la API de *Dronekit-Python*. La clase principal de este proyecto es `vehicle`. La conexión eléctrica entre la computadora complementaria y el controlador de vuelo, además de la comunicación con el piloto automático usando *Dronekit* se detallan en el Anexo B, Sección B.4. Las definiciones y mensajes comunes que gestiona MAVLink se definen en *commom.xlm* [44].

La ubicación de `HOME` del vehículo está definida en el atributo `home_location` de la clase principal. Este atributo puede establecerse de forma directa usando programación (en el caso de SITL), o al armar el vehículo previo a cualquier proceso.

El Código 4.5 presenta la función `ArmadoYDespegue`. Un vehículo puede *armar* su sistema de propulsión solo si ha completado todas las comprobaciones de seguridad. Estas comprobaciones buscan

prevenir accidentes debido a errores de calibración, y/o configuración incorrecta de sensores. El atributo `is_armable` denota la posibilidad del vehículo para iniciar el proceso de *armado*. *Ardupilot* permite controlar el movimiento de un UAV en el modo guiado (*GUIDED*). Solo cuando el vehículo se encuentre en este modo, se puede establecer el atributo `armed` en `TRUE`. Este proceso al igual que la comprobación de seguridad puede tomar algún tiempo en ser completado.

```
1  def armado_y_despegue(alturaobjetivo):
2    while not vehicle.is_armable:
3      time.sleep(1)
4    vehicle.mode = VehicleMode("GUIDED")
5    vehicle.armed = True
6    while not vehicle.armed:
7      time.sleep(1)
8    vehicle._master.mav.command_long_send(0, 0,mavutil.mavlink.
         MAV_CMD_NAV_TAKEOFF, 0, 0, 0, 0, 0, 0, 0,alturaobjetivo)
9    while True:
10     if vehicle.location.global_relative_frame.alt >= alturaobjetivo * 0.9:
11       break
12     time.sleep(1)
```

Código 4.5: Función ArmadoYDespegue

El proceso de despegue se realiza empleando el mensaje `command_long`, para enviar el comando `MAV_CMD_NAV_TAKEOFF` y establecer el valor de altitud `alturaobjetivo` del vehículo. La Tabla 4.3 presenta los campos del mensaje `command_long`.

Tabla 4.3: Descripción de campos del mensaje *command_long*

Campo	Tipo	Valor	Descripción
target_system	uint8_t		Sistema en el que debe ejecutarse el comando
target_componet	uint8_t		Componente en el que debe ejecutarse el comando
command	uint16_t	MAV_CMD	Identificador de comando
confirmation	uint8_t		0: Primera transmisión 1-255: transmisiones de confirmación
param1	float		Parámetro 1 del comando
param2	float		Parámetro 2 del comando
param3	float		Parámetro 3 del comando
param4	float		Parámetro 4 del comando
param5	float		Parámetro 5 del comando
param6	float		Parámetro 6 del comando
param7	float		Parámetro 7 del comando

Al enviar un comando específico (`MAV_CMD`) hacia el controlador de vuelo, los campos `param` se asignan a los parámetros del comando. En este caso, el comando empleado en el despegue es `MAV_CMD_NAV_TAKEOFF`, sus parámetros se describen a continuación:

1. **pitch:** mínimo valor de *pitch* expresado en grados
2. Vacío
3. Vacío
4. **yaw:** ángulo de *yaw* expresado en grados
5. **latitude:** latitud
6. **longitude:** longitud
7. **altitude:** altitud expresada en metros

La altura a la que se encuentra el vehículo relativa a la altitud de `HOME`, se encuentra definida en el atributo `alt` dentro de la propiedad `global_relative_frame` de `location`. Es necesario validar la altura alcanzada, incluyendo un margen de error.

Por otra parte, el Código 4.6 implementa la función `DirigirVehiculo`. En este caso, es necesario definir un marco de referencia de coordenadas (`frame`), altitud (`alt`) y velocidad en el aire (`airspeed`) del vehículo. Se establece `MAV_FRAME_GLOBAL_RELATIVE_ALT` como marco de referencia global, la altitud se define con respecto a la posición inicial o `HOME` establecido.

```
1 def dirigir_vehiculo(ubicacion):
2   frame = mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT
3   alt = ubicacion.alt
4   vehicle.airspeed = 1
5   vehicle._master.mav.mission_item_send(0, 0, 0,frame,mavutil.mavlink.
        MAV_CMD_NAV_WAYPOINT, 2, 0, 0, 0, 0, 0,ubicacion.lat,ubicacion.lon,
        alt)
```

Código 4.6: Función DirigirVehiculo

El mensaje empleado para establecer la posición del *waypoint* a alcanzar es `mission_item`. La Tabla 4.4 describe cada uno de los campos de este mensaje. Al igual que el mensaje `command_long` los campos `param` se reservan para el comando específico.

Tabla 4.4: Descripción de campos del mensaje *mission_item*

Campo	Tipo	Valor	Descripción
target_system	uint8_t		Sistema en el que debe ejecutarse el comando
target_componet	uint8_t		Componente en el que debe ejecutarse el comando
seq	uint16_t		Secuencia
frame	uint8_t	MAV_FRAME	Sistema de coordenadas del waypoint
command	uint16_t	MAV_CMD	Identificador de comando
current	uint8_t		0: falso, 1: verdadero
autocontinue	uint8_t		Continuar automáticamente al siguiente waypoint
param1	float		Parámetro 1 del comando
param2	float		Parámetro 2 del comando
param3	float		Parámetro 3 del comando
param4	float		Parámetro 4 del comando
x	float		Parámetro 5 del comando: latitud
y	float		Parámetro 6 del comando: longitud
z	float		Parámetro 7 del comando: altitud

El campo `MAV_CMD` utilizado por esta función es `MAV_CMD_NAV_WAYPOINT`, cuyos parámetros se presentan a continuación:

1. **`hold`:** tiempo en segundos que el vehículo debe mantenerse en el *waypoint*
2. **`accept radius`:** radio de aceptación para alcanzar un punto de referencia, expresado en metros
3. **`pass radius`:** permite control de trayectoria (horario o antihorario sobre el *waypoint*)
4. **`yaw`:** ángulo de *yaw* en el *waypoint*, expresado en grados
5. **`latitude`:** latitud
6. **`longitude`:** longitud
7. **`altitude`:** altitud expresada en metros

Un *waypoint* debe estar definido como una ubicación global relativa (`LocationGlobalRelative`), donde se especifica latitud y longitud expresada en grados decimales, y altitud relativa a `HOME`, expresada en metros.

La función `ObtenerDistancia` hace uso de una aproximación de la formula de *Haversine* para obtener la distancia entre dos puntos, X y Y, en la superficie terrestre. Los valores de latitud y longitud expuestos en la Ecuación (4.14) están expresados en grados decimales.

$$d = \sqrt{(LatY - LatX)^2 + (LonY - LonX)^2} \times 1{,}113195 \times 10^5 \tag{4.14}$$

Por último, los procesos `Aterrizar` y `DesarmarVehiculo` se logran colocando el vehículo en modo *LAND*. Este modo aterriza el vehículo desde cualquier altura a la que se encuentre y desarma el sistema de propulsión al llegar a tierra.

4.3.2. Localización y detección de marcador

El error de precisión del módulo GPS y el `MargenErrorWP` introducido en el Algoritmo 1, sumado a la altura con respecto a tierra a la que se encuentra el vehículo, no garantizan que el marcador en la plataforma de aterrizaje se encuentre dentro del campo de visión (*field of view*) de la cámara incorporada. El algoritmo de localización y detección es el encargado de garantizar que la cámara incorporada en el vehículo pueda detectar el marcador ubicado en la plataforma de aterrizaje.

El Algoritmo 2 presenta el procedimiento de localización del marcador *ArUco*. El algoritmo inicia cuando el vehículo se encuentra en el aire, a una altura definida en el plan de vuelo. En este punto, inicialmente emplea la función `DetectarMarcador` (Código 4.3) para descubrir si un marcador se encuentra dentro del *field of view* actual de la cámara (Algoritmo 2, línea 1). Si no se encuentra un marcador, inicia un proceso de búsqueda. En primer lugar, emplea la función `UbicacionActual` para establecer una ubicación de referencia (`ubicacionReferencia`), que será usada para controlar el movimiento del vehículo (Algoritmo 2, línea 5). El proceso de búsqueda está definido en una área específica descrita mediante distancias (`DistAreaLoc`). El UAV se mueve en una sola dirección a la vez empleando la función `MoverVehiculo` acorde a vectores de velocidad en dos ejes definidos en `VelLocX` y `VelLocY` (Algoritmo 2, líneas 7 a 10). El punto `ubicacionReferencia` se emplea para controlar la distancia que el vehículo se mueve en el área definida en `DistAreaLoc` (Algoritmo 2, líneas 11 a 13). Al finalizar cada movimiento, se evalúa el *field of view* actual en busca de un marcador. Si lo encuentra, el Algoritmo termina (Algoritmo 2, líneas 16 a 18), caso contrario, actualiza banderas y continua con el proceso de búsqueda.

Algoritmo 2: Localización y detección de marcador

```
Datos: VelLocX, VelLocY, DistAreaLoc, MargenErrorPRef
marcador = DetectarMarcador()
mientras marcador == 0 hacer
    distRes = 0
    contador = 0
    ubicacionReferencia = UbicacionActual()
    para distLoc en DistAreaLoc hacer
        velx = VelLocX[contador]
        vely = VelLocY[contador]
        mientras distRes <distLoc hacer
            MoverVehiculo(velx, vely, 0)
            distRes=ObtenerDistancia(UbicacionActual, ubicacionReferencia)
            si distRes <= MargenErrorPRef entonces
                salir
            fin
        fin
        marcador = DetectarMarcador()
        si marcador == 1 entonces
            salir
        fin
        ubicacionReferencia = UbicacionActual()
        contador = contador + 1
        distRes = 0
    fin
fin
```

El *field of view* de una cámara es el ángulo de visión que es capaz de captar el sensor del dispositivo. El área de detección que el *field of view* puede abarcar depende de la distancia de separación entre la cámara y el plano objetivo. La Figura 4.11 muestra el área de detección a una distancia d desde la cámara. En un *quadcopter* con la cámara apuntando hacia abajo, esta distancia de separación, es la altura a la que se encuentra el vehículo con respecto a tierra.

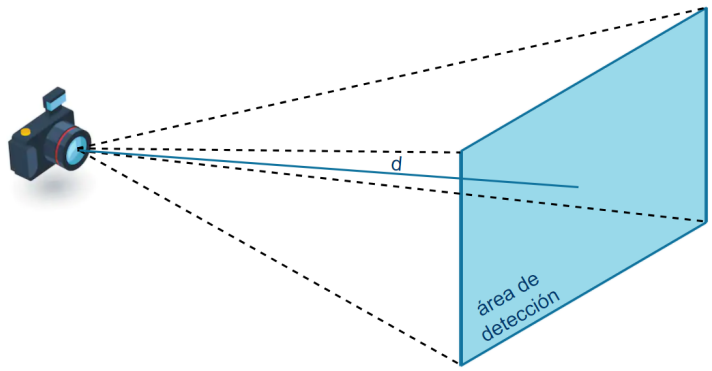


Figura 4.11: Área de detección

El Algoritmo 2 define un área de localización en función de áreas de detección. La Figura 4.12

presenta un área de localización formada por seis subáreas de detección a una altura de vuelo, h. El UAV debe seguir la trayectoria definida por las distancias $dist1$, $dist2$, $dist3$ y $dist4$ para abarcar toda la superficie definida. El movimiento se realiza empleando el sistema de referencia XYZ.

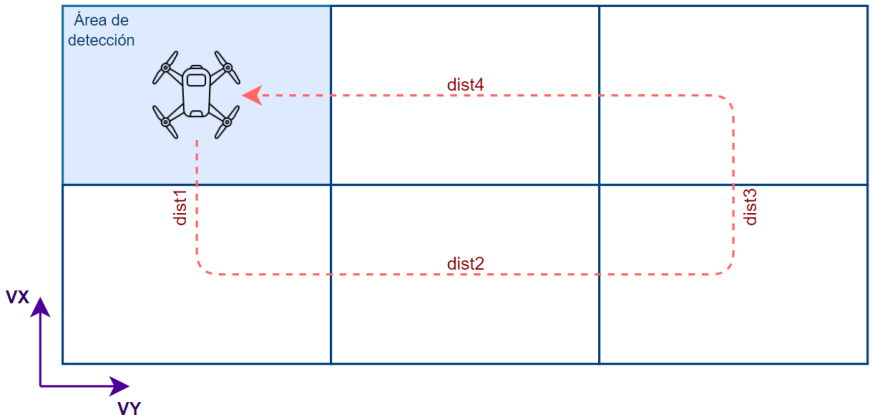


Figura 4.12: Área de localización

El Código 4.7 define la función `MoverVehiculo`. Esta función utiliza el mensaje `SET_POSITION_TARGET_LOCAL_NED` para establecer la posición del vehículo en el marco de referencia NED. La Tabla 4.5 presenta los campos con los que cuenta este mensaje.

```
1 def mover_vehiculo(velx, vely, velz):
2   msg = vehicle.message_factory.set_position_target_local_ned_encode(0, 0,
        0, mavutil.mavlink.MAV_FRAME_BODY_OFFSET_NED, 0b0000111111000111,
        0, 0, 0, velx, vely, velz, 0, 0, 0, 0, 0)
3   vehicle.send_mavlink(msg)
```

Código 4.7: Función MoverVehiculo

Tabla 4.5: Descripción de campos del mensaje

Campo	Tipo	Valor	Descripción
time_boot_ms	uint32_t		Marca de tiempo
target_system	uint8_t		Sistema en el que debe ejecutarse el comando
target_componet	uint8_t		Componente en el que debe ejecutarse el comando
coordinate_frame	uint8_t	MAV_FRAME	Marco de referencia
type_mask	uint16_t	POSITION_TARGET_TYPEMASK	Mapa de bits para indicar que campos debe ignorar el vehículo
x	float		Posición en X (m)
y	float		Posición en Y (m)
z	float		Posición en Z (m)
vx	float		Velocidad en X (m/s)
vy	float		Velocidad en Y (m/s)
vz	float		Velocidad en Z (m/s)
afx	float		Aceleración en X (m/s^2)
afy	float		Aceleración en Y (m/s^2)
afz	float		Aceleración en Z (m/s^2)
yaw	float		Ajuste de *yaw* (rad)
yaw_rate	float		Velocidad de ajuste de yaw (rad/s)

El marco de referencia seleccionado para el movimiento del vehículo es `MAV_FRAME_BODY_OFFSET_NED`, donde las posiciones son relativas a la posición y rumbo (*heading*) actual del vehículo. La velocidad toma un valor positivo en X hacia adelante, en Y hacia la derecha, y en Z hacia abajo, como se muestra en la Figura 4.13. El valor para el campo `type_mask` está definido de tal forma que lea únicamente los campos de velocidad e ignore todos los campos de posición, aceleración y *yaw*. Los campos de aceleración (`afx`, `afy`, `afz`) y *yaw* (`yaw`, `yaw_rate`) no son soportados aún por el *firmware* del vehículo.

El método `send_mavlink` envía mensajes sin procesar al vehículo en cualquier momento y sin importar el modo en el que se encuentre. Para asegurar un formato adecuado hay que usar el contenedor `MESSAGE_FACTORY`.

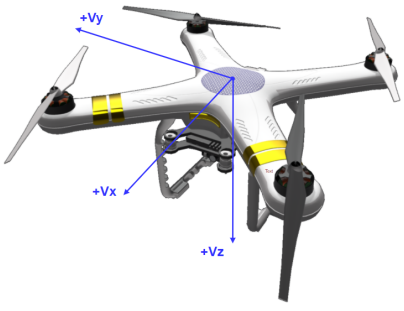


Figura 4.13: Velocidades en el sistema de coordenadas XYZ

4.3.3. Rotación

Un vehículo establece su rumbo relativo al Norte (0°) de acuerdo a la posición del punto de interés al que se dirige, es decir, apunta en dirección al *waypoint* en cuestión. En este escenario, la orientación del marcador ubicado en la plataforma de aterrizaje puede no ser igual al rumbo que el vehículo tome al avanzar a su ubicación. El algoritmo de rotación permite obtener una aproximación de orientación del marcador relativo al marco de referencia, y modificar el rumbo del vehículo para alinearlo con el marcador.

El Algoritmo 3 ilustra los procesos involucrados en la corrección de orientación del vehículo. Inicialmente, el algoritmo emplea la función `ObtenerYawActual` para obtener el *yaw* actual del UAV, expresado en grados (Algoritmo 3, línea 1), que será empleado por la función `ObtenerCorreccionYaw` para calcular un ángulo de orientación del marcador (Algoritmo 3, línea 2). Luego, establece un bucle para garantizar el proceso de corrección, que inicia al establecer un nuevo valor de *yaw* (`yawNuevo`) usando la función `establecerYaw` (Algoritmo 3, línea 5). La corrección de rumbo se establece respetando un criterio de tolerancia acorde al valor calculado. Por tanto, se establece un rango inferior (`limiteYawInferior`) y superior (`limiteYawSuperior`) donde el valor de *yaw* del vehículo se considere corregido (Algoritmo 3, líneas 7 a 10).

Algoritmo 3: Corrección de orientación

```
   Datos: yawTolerancia
 1 yawReferencia = ObtenerYawActual()
 2 yawNuevo = ObtenerCorreccionYaw()
 3 orientacion = 1
 4 mientras orientacion == 1 hacer
 5 |   establecerYaw(yawNuevo)
 6 |   yawactual = ObtenerYawActual()
 7 |   limiteYawSuperior = yawNuevo + yawTolerancia
 8 |   limiteYawInperior = yawNuevo - yawTolerancia
 9 |   si limiteYawSuperior >= yawactual >= limiteYawInferior entonces
10 |   |   orientacion = 0
11 |   fin
12 fin
```

La resolución de la cámara establece la dimensión en píxeles que tendrá un *frame* de datos. OpenCV establece un sistema de referencia de dos ejes, XY, cuyo origen se ubica en la parte superior izquierda del *frame* de datos. La Figura 4.14 presenta un marcador detectado, su orientación relativa al *frame* de datos (β), y la orientación del *frame* de datos relativa al marco de referencia NED. Cuando un marcador *ArUco* es detectado, el vértice superior izquierdo o vértice guía, es marcado para indicar la orientación del mismo. Los vértices se enlistan iniciando por el vértice guía (P_1) y siguiendo un sentido horario (P_2, P_3 y P_4).

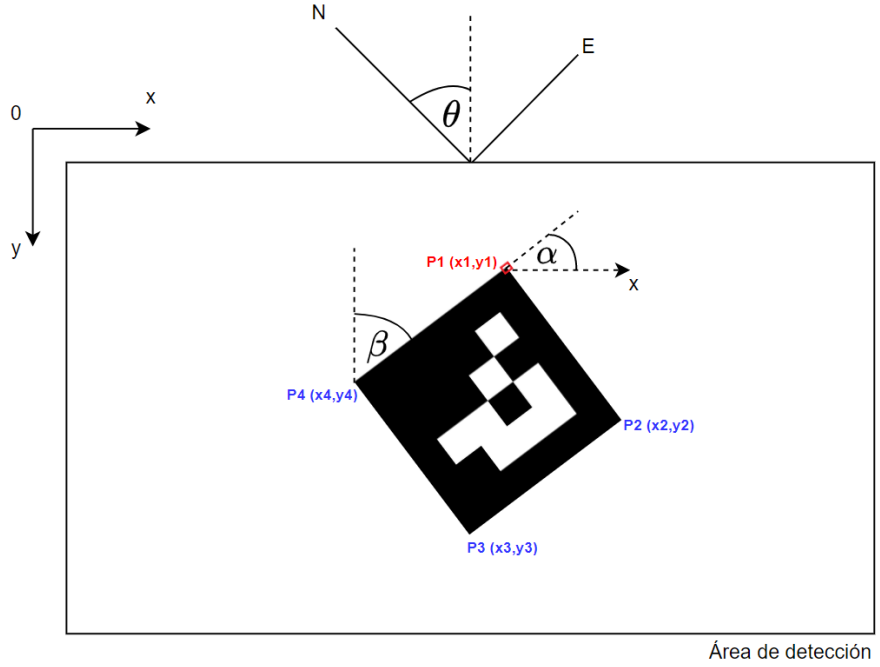


Figura 4.14: Orientación de marcador relativo al *frame* de datos y al marco NED

La orientación del marcador relativa al *frame* de datos (β) puede determinarse usando el ángulo de inclinación de la recta (P_1, P_4, α). El valor de α se obtiene usando la Ecuación (4.15). β se obtiene al realizar un ajuste al valor del ángulo de inclinación, acorde a su signo y la posición del vértice guía.

$$\alpha = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{4.15}$$

La posición del vértice guía se puede determinar comparando las distancias entre cada vértice y el origen. El Algoritmo 4 presenta el procedimiento seguido para el cálculo de β. Inicialmente, el algoritmo encuentra el vértice más cercano al origen (`minposicion`) determinando el índice de la menor distancia en la lista de distancias vértice-origen, `ListaDistOr` (Algoritmo 4, línea 1). Luego, evalúa el signo de α y la posición del vértice más cercano al origen para obtener β sumando 90° o 270° (Algoritmo 4, líneas 2 a 14).

Algoritmo 4: Cálculo de β

Datos: α, ListaDistOr

```
minposicion = indice(minimo(ListaDistOr))
si α <0 entonces
    si minposicion == 1 || minposicion == 4 entonces
        β = 90+α
    fin
    si minposicion == 2 || minposicion == 3 entonces
        β = 270+α
    fin
en otro caso
    si minposicion == 1 || minposicion == 2 entonces
        β = 270+α
    fin
    si minposicion == 3 || minposicion == 4 entonces
        β = 90+α
    fin
fin
```

El ángulo θ o `yawNuevo` es el valor de *yaw* necesario para alinear el vehículo y el marcador. La función `ObtenerCorreccionYaw` realiza todos los cálculos previamente presentados, emplea el valor de β y el rumbo actual del vehículo (`yawReferencia`) para determinar θ, este ángulo es la suma de `yawReferencia` y β.

El valor de *yaw* del vehículo, expresado en radianes, se encuentra definido dentro del atributo `attitude.alt` de la clase principal. El Código 4.8 presenta la función `establecerYaw`.

```
def establecer_yaw(rumbo):
  msg = vehicle.message_factory.command_long_encode(
    0, 0, mavutil.mavlink.MAV_CMD_CONDITION_YAW, 0, rumbo,0, direccion, 0,
        0, 0, 0)
  vehicle.send_mavlink(msg)
```

Código 4.8: Función EstablecerYaw

La función utiliza el mensaje `command_long` para empaquetar el comando `MAV_CMD_CONDITION_YAW`, empleado para alcanzar un ángulo de rumbo objetivo. Los parámetros del comando se describen a continuación:

1. `heading`: valor de *yaw*, expresado en grados
2. `angular speed`: velocidad angular expresada en deg/s
3. `direction`: dirección de giro, -1 para sentido antihorario, y 1 para sentido horario
4. `relative`: 0: ángulo absoluto, 1: desplazamiento relativo
5. Vacío
6. Vacío
7. Vacío

4.3.4. Aterrizaje

El descenso del vehículo se ve afectado principalmente por la velocidad del viento. El algoritmo de aterrizaje es el encargado de determinar la ubicación del marcador dentro del *frame* de datos, y calcular vectores de velocidad que posicionen el vehículo sobre el marcador para descender de manera controlada hasta finalizar el aterrizaje.

Un *frame* de datos posee una resolución vertical (`vres`) y horizontal (`hres`) definidas. Estos parámetros son empleados para determinar el punto central (p) de la captura, definir una ventana o subárea de tolerancia de ancho $2 \times m$ píxeles, y cuatro franjas de aproximación (f), como se muestra en la Figura 4.15. Dicha ventana permite al algoritmo mantener un rango de variación al momento de posicionar el centro del marcador.

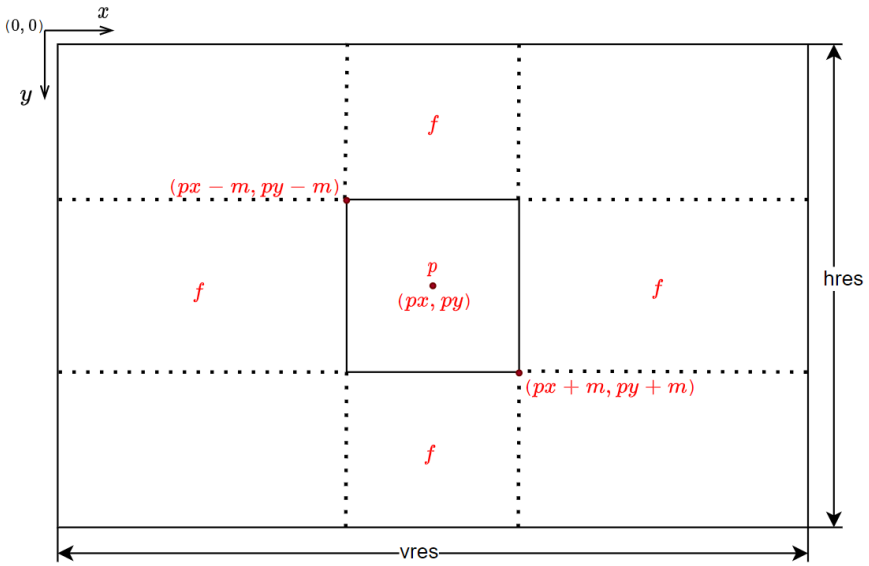


Figura 4.15: Ventana de tolerancia

El Algoritmo 5 presenta el procedimiento seguido para posicionar y descender el vehículo sobre el marcador detectado. En primer lugar, se establecen vectores de retorno (`velXAnt`, `velYAnt`), en caso de que el vehículo perdiese de vista al marcador (Algoritmo 5, líneas 1 y 2). A continuación, se emplea la función `ObtenerCentroMarcador` y `ObtenerVectorVelocidad` para obtener el centro del marcador y vectores de velocidad unidimensional (`velX`, `velY`), calculados en función de la ubicación del marcador (Algoritmo 5, líneas 4 y 5). Estas variables permiten determinar si el centro del marcador se encuentra en la ventana de tolerancia o en alguna franja de aproximación, y emplear la función `MoverVehiculo` para efectuar movimientos en el eje X o Y (Algoritmo 5, líneas 13 y 16), o descender (Algoritmo 5, línea 11). Si la coordenada del centro del marcador es $(0, 0)$, el marcador se encuentra fuera del área de detección de la cámara. En este caso, el vehículo debe utilizar los vectores velocidad de retorno para revertir el último movimiento efectuado (Algoritmo 5, líneas 6 y 7). El proceso descrito se repite hasta alcanzar una altitud relativa al suelo determinada, `AlturaAterrizaje`. Para finalizar, el vehículo inicia el proceso `Aterrizar` y `DesarmarVehiculo` (Algoritmo 5).

Algoritmo 5: Centrado y descenso

```
Datos: AlturaAterrizaje, velDescenso
velXAnt = 0
velYAnt = 0
mientras ObtenerAlturaVuelo() >= AlturaAterrizaje hacer
    cx, cy = ObtenerCentroMarcador()
    velX, velY = ObtenerVectorVelocidad()
    si cx == 0 & cy == 0 entonces
        MoverVehiculo(velXAnt, velYAnt, 0)
    fin
    si (Py) + m) >= cy >= (Py -) entonces
        si (Px + m) >= cx >= (Px - m) entonces
            MoverVehiculo(0, 0, velDescenso)
        en otro caso
            MoverVehiculo(0, velY, 0)
        fin
    en otro caso
        MoverVehiculo(velX, 0, 0)
    fin
    velXAnt = velX
    velYAnt = velY
fin
Aterrizar()
DesarmarVehiculo()
```

El centro del marcador se puede determinar calculando el punto medio entre dos vértices opuestos, como se ilustra en la Figura 4.16.

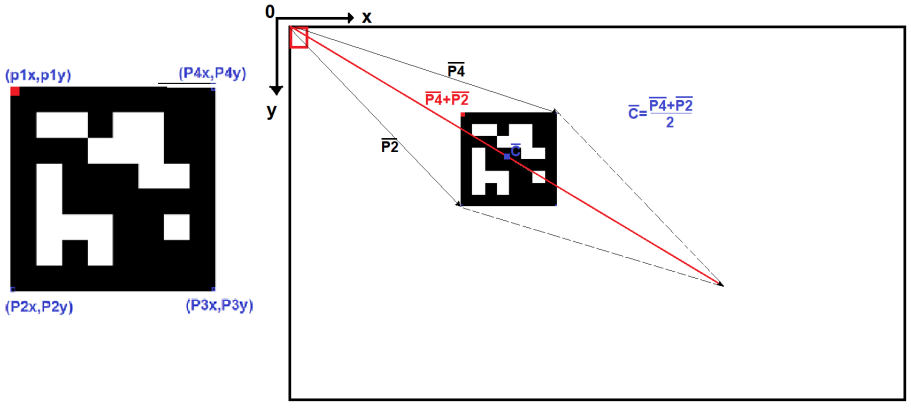


Figura 4.16: Cálculo del centro de marcador.

El proceso de cálculo de vectores de velocidad en los ejes X y Y segmenta el *frame* de datos en cuatro cuadrantes, y establece la dirección (signo) del vector velocidad en el marco de referencia del vehículo (XYZ). La dirección de los vectores velocidad del vehículo es opuesta a la dirección que el marcador debe seguir para llegar al centro del *frame* de datos, como lo ilustra la Figura 4.17.

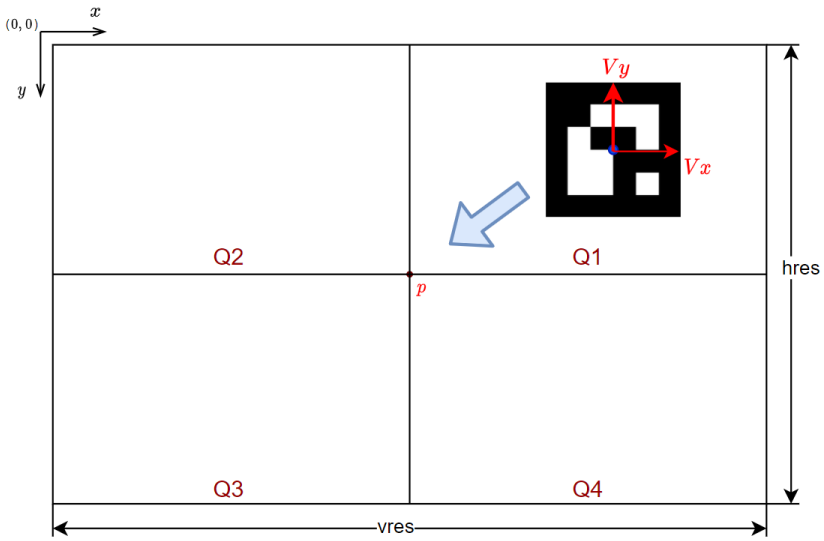


Figura 4.17: Cuadrantes y cálculo de vector de velocidad en el área de detección

4.4. Recolección de datos

El proceso de recolección de datos en los nodos de la red se desarrolla usando ION-DTN. La instalación y configuración de este paquete de *software* se presenta en el Anexo B, Sección B.5. La Figura 4.18 presenta un esquema reducido de los componentes que conforman la red DTN desplegada. Se definen tres componentes en la red:

- **Servidor:** nodo DTN destinado a ser el punto final o receptor de los datos recolectados por el dron. La ubicación del servidor se establece en el atributo *HOME* o punto de despegue del UAV.
- **Nodo estático:** es una placa SBC, *Raspberry Pi*, usada como sensor de la red WSN. Este elemento establece comunicación con el nodo móvil a través de un enlace inalámbrico para transmitir datos. Asociado a cada nodo estático se encuentra un marcador *ArUco*, empleado por los algoritmos de visión artificial, y de vuelo para lograr un aterrizaje de precisión.
- **Nodo móvil:** es el elemento central, efectúa el trabajo de mula o retransmisor para los demás nodos en la red. Sujeto al *frame* del vehículo (*RaspiUAV*) se encuentra una placa *Raspberry Pi 3B*, configurada como punto de acceso de la red `192.168.0.X`, para establecer comunicación inalámbrica con los demás nodos. Además, este nodo es el encargado de ejecutar todos los algoritmos presentados en las secciones anteriores. La configuración de la placa SBC como punto de acceso se presenta en el Anexo B, Sección B.1.

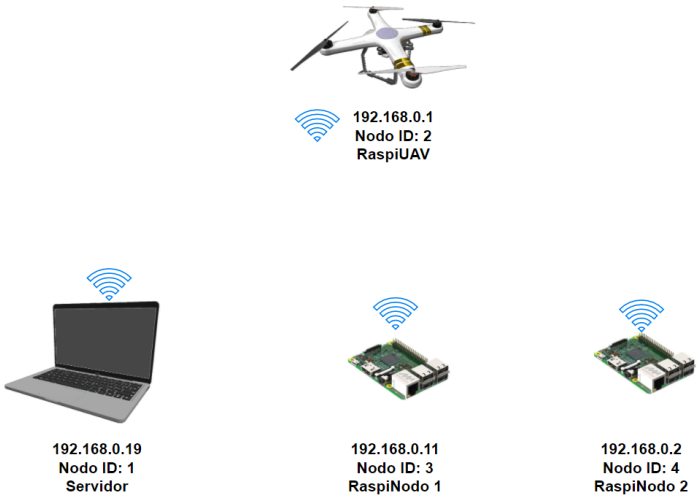


Figura 4.18: Esquema de red DTN

La transmisión de datos entre un nodo estático y el UAV inicia cuando el vehículo ha terminado de ejecutar todos los algoritmos de vuelo para el nodo específico. La configuración de ION-DTN requiere la definición de contactos con el nodo móvil.

La implementación se centra además en *pyion*, una extensión de *Python* para ION-DTN. La instalación y configuración de este paquete de *software* conjuntamente con la configuración DTN se presenta en el Anexo B. *pyion* proporciona una interfaz para enviar/recibir datos a través de ION-DTN, y funciones para modificar la configuración de la red en tiempo de ejecución. El Código 4.9 se usa para enviar una cadena de texto del nodo 1 (`node_ID`) hacia un nodo 2.

```
1 node_id = 1
2 EID = 'ipn:1.1'
3 proxy = pyion.get_bp_proxy(node_id)
4 proxy.bp_attach()
5 with proxy.bp_open(EID) as eid:
6   eid.bp_send('ipn:2.1', b'Hola')
```

Código 4.9: Transmisor DTN

El proceso de envío de datos se realiza a través de un *proxy* conectado al motor de ION-DTN en un nodo específico (Código 4.9, líneas 2 a 5). Al conectarse al *proxy*, el envío de datos se realiza empleando un EID del nodo transmisor (Código 4.9, línea 6). Finalmente, el método `bp_send` es el encargado de enviar datos hacia el EID destino (Código 4.9, línea 7). Por otra parte, el Código 4.10 se usa para recepción de paquetes.

```
1 node_id = 2
2 EID = 'ipn:2.1'
3 proxy = pyion.get_bp_proxy(node_id)
4 proxy.bp_attach()
5 with proxy.bp_open(EID) as eid:
6   while eid.is_open:
```

```
7     try:
8       print('Received:', eid.bp_receive())
9     except InterruptedError:
10      break
```

Código 4.10: Transmisor DTN

En este caso, se conecta al *proxy*, se abre un EID en el nodo local y se escucha, en espera de un mensaje utilizando el método `bp_receive` (4.10, líneas 6 a 11).

Capítulo 5

Resultados

Este capítulo presenta los resultados obtenidos durante la experimentación con los algoritmos que conforman el sistema de control de aterrizaje para recolección de datos. El capítulo inicia planteando el análisis de dispositivos de captura de video en el proceso de detección de marcadores *ArUco* (Sección 5.1). A continuación, se presentan las pruebas y resultados obtenidos al evaluar los algoritmos de vuelo (Sección 5.2). El capítulo finaliza presentando los escenarios de experimentación planteados para analizar el rendimiento en la recolección de datos empleando una red DTN (Sección 5.3).

5.1. Detección de marcador

El módulo cámara para *Raspberry Pi* cuenta con un lente fijo, que no genera distorsión en las capturas realizadas, mientras que, la cámara *Turnigy HD* cuenta con un lente gran angular que provoca distorsión en las capturas. Si bien el módulo para *Raspberry Pi* no genera distorsión, la cámara *Turnigy* posee un *field of view* mayor. Por tanto, captura una mayor área de detección a la misma distancia que el módulo cámara. Se realizaron varias pruebas para reducir la distorsión de capturas en la cámara *Turnigy* usando los Códigos 4.2 y 4.4. Los códigos se utilizaron para calcular la matriz y el vector de distorsión. La Figura 5.1 presenta la captura antes y después del proceso de corrección de distorsión para la cámara *Turnigy*. El proceso de corrección de distorsión desarrollado reduce la resolución vertical de la imagen y cuenta con un porcentaje de error en los extremos de la imagen.

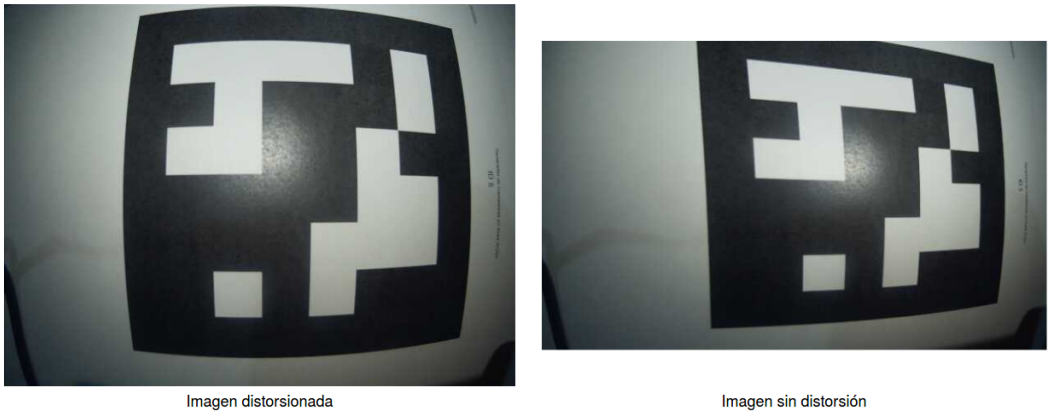


Figura 5.1: Captura antes y después de eliminar la distorsión

A fin de comparar el rendimiento de los dispositivos de captura de video se realizaron distintas pruebas con el Código 4.3. Se contemplaron tres escenarios para el análisis de detección con los dispositivos.

- **Altura máxima de detección:** se colocó el dispositivo de captura a diferentes alturas de un marcador de tamaño determinado, para obtener la altura máxima de detección por tamaño de marcador.
- **Altura mínima de aterrizaje:** se colocó el dispositivo a diferentes alturas del marcador, a fin de determinar la mínima distancia para que el marcador se encuentre dentro del área de detección de la cámara.
- **Iluminación**: se colocó el marcador en entornos con diferente iluminación, para determinar su influencia en la detección.

Los resultados muestran que el módulo de *Raspberry Pi* puede detectar un marcador a mayor altura que la cámara *Turnigy*, como lo muestra la Figura 5.2.

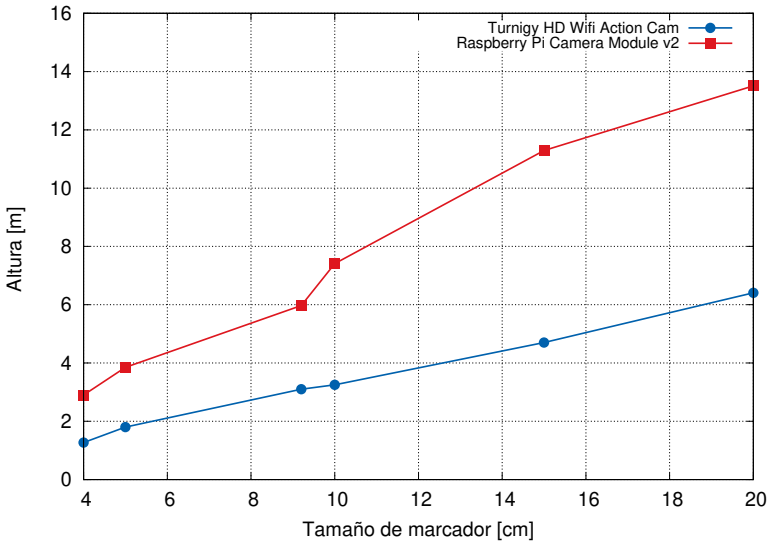


Figura 5.2: Máxima altura de detección de marcador *ArUco*

La Figura 5.3 presenta la distancia mínima para una detección exitosa por tamaño de marcador. Esta distancia determina la altura mínima de aterrizaje definida en el Algoritmo 5. La cámara *Turnigy* permite una detección exitosa de un marcador a menor distancia que el módulo de cámara, debido a su lente gran angular. Esta distancia influye en la precisión del aterrizaje, puesto que, los movimientos del vehículo a altitudes bajas permiten una mejor aproximación al marcador.

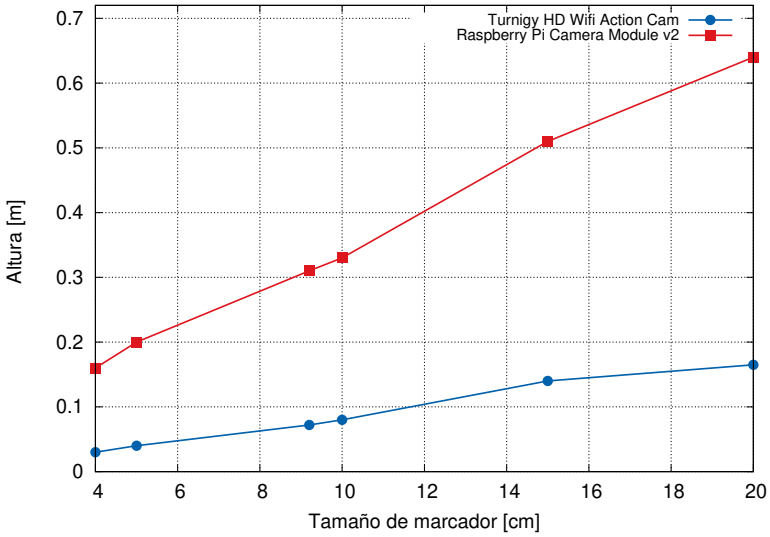


Figura 5.3: Mínima altura de aterrizaje

Por otro lado, las pruebas desarrolladas en el escenario Iluminación empleando luz natural y luz artificial de forma directa para ambos casos, demostraron que la iluminación con luz natural influye en la detección en el módulo de *Raspberry Pi*, produciendo detección intermitente en *frames* consecutivos, como se muestra en la Figura 5.4. Dicho comportamiento se produce para todos los tamaños de marcador. Este hecho no se replicó en el escenario iluminado con luz artificial directa para el módulo, y en ningún escenario para la cámara *Turnigy*.

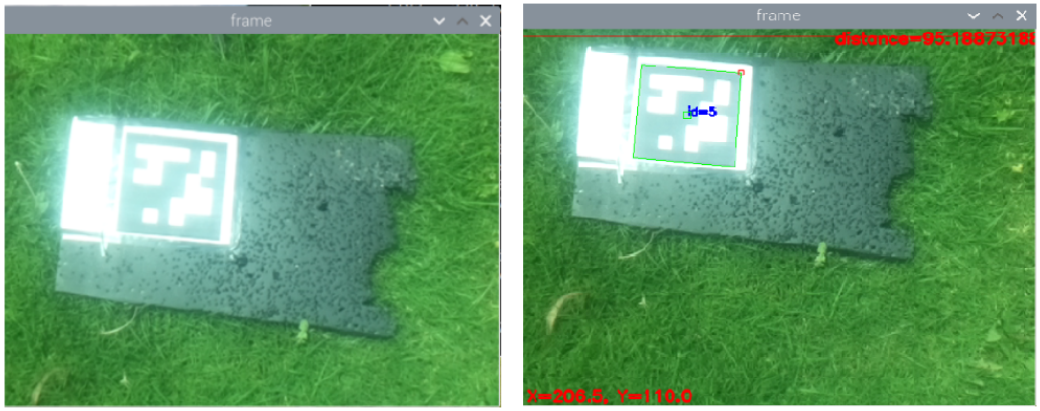


Figura 5.4: Intermitencia en la detección del marcador en frames consecutivos provocada por la influencia de luz natural directa sobre el marcador

De acuerdo a los resultados obtenidos en la detección del marcador, se seleccionó la cámara *Turnigy* para el desarrollo de este trabajo. La Figura 5.5 muestra la detección de un marcador de 20 cm

empleando este dispositivo.

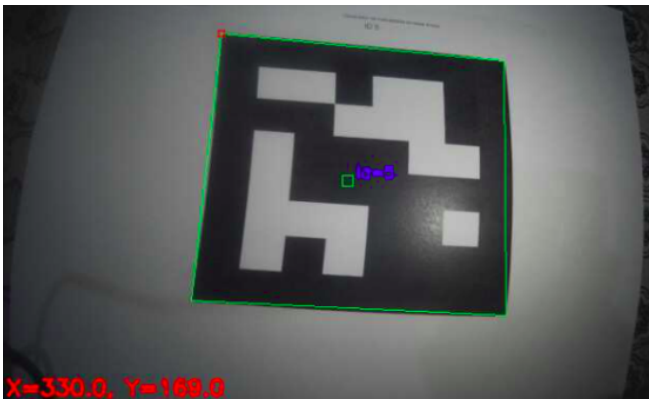


Figura 5.5: Detección de marcador *ArUco* empleando la cámara Turnigy en un ambiente con iluminación artificial directa

5.2. Algoritmos de vuelo

Todas las pruebas de vuelo se desarrollaron bajo la presencia de un piloto de respaldo, encargado de controlar el vehículo usando un radio control. El piloto puede cambiar el modo de vuelo del UAV en cualquier momento, inhabilitando el control proporcionado por la placa SBC. Este recurso se emplea en el caso de presentarse algún error o altercado en el desarrollo de la prueba.

Se desarrollaron pruebas con el Algoritmo 1 definiendo cinco y siete puntos de ruta (*waypoints*). El vehículo inicia el proceso de aterrizaje si la distancia entre el punto de ubicación actual del dron y el punto de referencia es menor al margen de error permitido. El margen de error se estableció en 45 cm. Para valores menores a 45 cm, el vehículo se mantiene en vuelo y no puede completar otros procesos.

Los experimentos contemplados para evaluar el algoritmo de orientación se centraron en despegar el vehículo sobre el marcador, corregir el rumbo y aterrizar. La Tabla 5.1 presenta diez valores de *yaw* objetivo (orientación del marcador) y el ángulo *yaw* de corrección calculado empleando el Algoritmo 3 para cada caso. El error medio entre el ángulo de orientación objetivo y el *yaw* calculado es 4,3°. La corrección de *yaw* en el vehículo empleando el Algoritmo 3 se logra correctamente estableciendo un margen de tolerancia de 8°.

Tabla 5.1: Error en la corrección de rumbo

Orientación de marcador [°]	Yaw calculado [°]	Error [°]
50	57	7
24	29	5
345	349	4
130	134	4
176	178	2
212	217	5
232	239	7
278	282	4
20	22	2
355	358	3

El valor adimensional de HDOP requerido para una lectura confiable de posicionamiento GPS debe ser menor a 1 en todo momento. Valores altos de Dilution of precision (DOP) desembocan en una mayor dispersión del error de posición [73]. Sin embargo, el número de satélites que mantienen un enlace con el módulo GPS del vehículo es variable durante el tiempo de vuelo. La Figura 5.6 presenta valores de HDOP típicos durante el vuelo del vehículo. Se puede observar que el valor adimensional para dilusión horizontal es menor mientras mayor sea el número de enlaces establecidos. La variación del parámetro HDOP en un momento dado influye directamente en la precisión de lectura, incrementando el error de posicionamiento, traducido en metros.

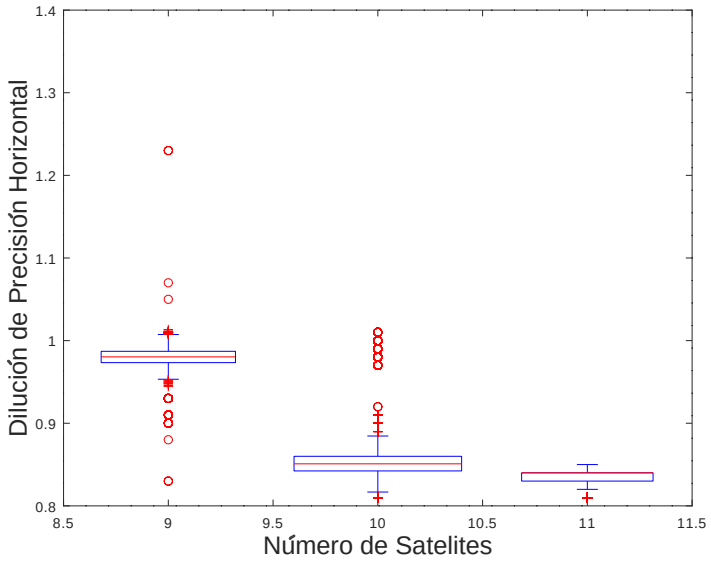


Figura 5.6: Variación de HDOP para un número de satélites

Para dimensionar la influencia del error de posición en el aterrizaje del dron, se desplegó un plan de vuelo con un solo punto de ruta (*waypoint*). Este punto hace referencia a un marcador colocado en tierra. El vehículo despega de un lugar cercano, se dirige a la ubicación de referencia, y aterriza. Se

realizaron múltiples pruebas, donde se midió la distancia entre la ubicación real del punto de referencia y el centro de la cámara del lugar de aterrizaje del vehículo, los resultados de doce pruebas se presentan en la Figura 5.7. Como muestra la figura, uno de estos aterrizajes está a menos de 1 m del punto de referencia, y la mayoría de ellos se encuentran a menos de 2 m. La distancia media desde el punto de referencia fue 143,2 cm. El error estándar fue 0,1 m, mientras que, el error cuadrático medio fue 0,34 m.

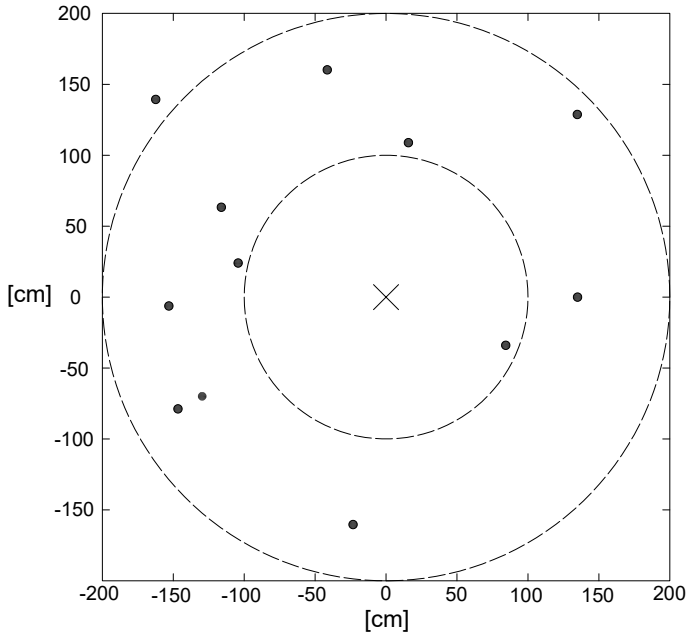


Figura 5.7: Aterrizaje del vehículo con el modo *LAND*

Se desarrolló el mismo proceso empleando el Algoritmo 5. La Figura 5.8 presenta el resultado de doce aterrizajes. Como muestra la figura, nueve aterrizajes alcanzaron una distancia entre el centro de la cámara y el marcador menor a 50 cm. La Figura 5.9 presenta el aterrizaje con menor distancia obtenido durante el desarrollo de las pruebas. La distancia entre el centro de la cámara y el marcador para esta prueba fue de 3 cm. La distancia media desde el marcador fue 22,86 cm. El error estándar fue de 6,3 cm, mientras que, el error cuadrático medio fue de 21 cm. La altura de aterrizaje que mejores resultados generó fue de 50 cm. Una altura menor a esta, ocasionaba vuelo errático, provocado para este caso en particular por un error en el barómetro del controlador de vuelo.

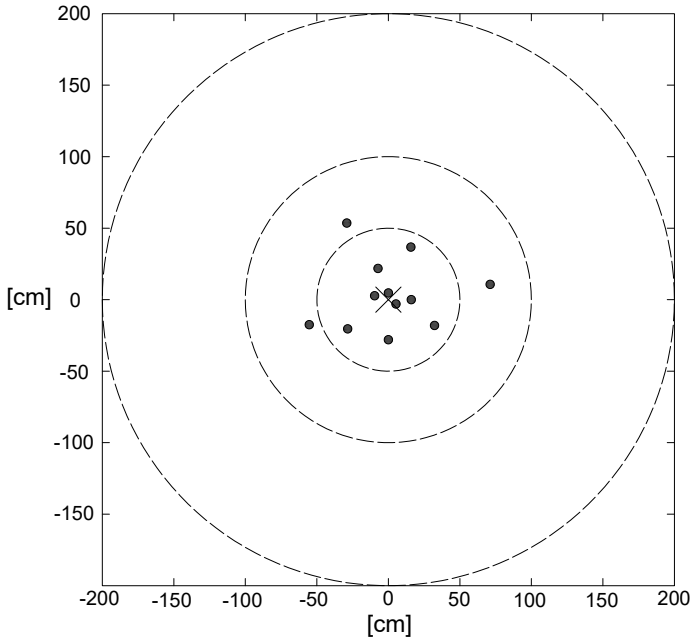


Figura 5.8: Aterrizaje usando sistema de aterrizaje de precisión

Figura 5.9: Aterrizaje de precisión

5.3. Recolección de datos

Las pruebas de rendimiento de ION-DTN se desarrollaron usando tres nodos DTN como expone la Figura 4.18. La separación entre nodos no fue mayor a 6 metros en dos escenarios:

- Escenario 1: transmisión de información desde el nodo 3 hacía el nodo 1 (*server*) a través del nodo 2 (UAV).
- Escenario 2: Transmisión de información desde el nodo 3 hacia el nodo 2.

Durante los experimentos se utilizó *tcpdump* para capturar los datos de tráfico en la red. Se emplearon estos datos para calcular el retraso, *throughput* y probabilidad de entrega para analizar el rendimiento de ION-DTN.

El retraso de paquetes se analizó enviando archivos de diferente tamaño en el Escenario 1, con una separación entre nodos de 3 metros. Las pruebas se realizaron dentro del rango de comunicación de los nodos y sin desconexión de ninguno de ellos. El intervalo de tiempo de entrega, el tamaño de archivo y el número de paquetes enviados se presenta en la Tabla 5.2. El intervalo de tiempo de entrega se refiere al tiempo que toma la transmisión de datos de extremo a extremo.

Tabla 5.2: Intervalo de tiempo de llegada por tamaño de archivo

Tamaño de archivo [MB]	Intervalo de tiempo [s]	Número de paquetes
0,1	2,84	260
0,3	6,37	546
0,5	8,33	595
1,5	24,64	1529
3	40,57	2861
4	59,36	2927

Los resultados demuestran un incremento en el intervalo de tiempo de entrega a medida que aumenta el tamaño de archivo. La Figura 5.10 presenta el retraso medio en la entrega de paquetes por tamaño de archivo.

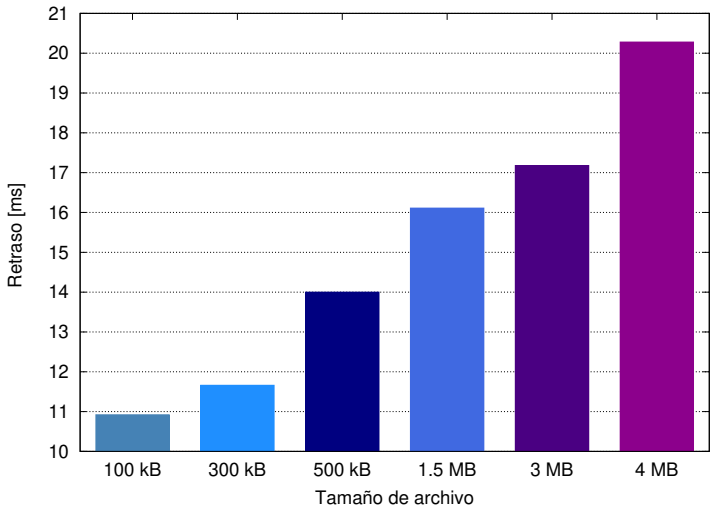
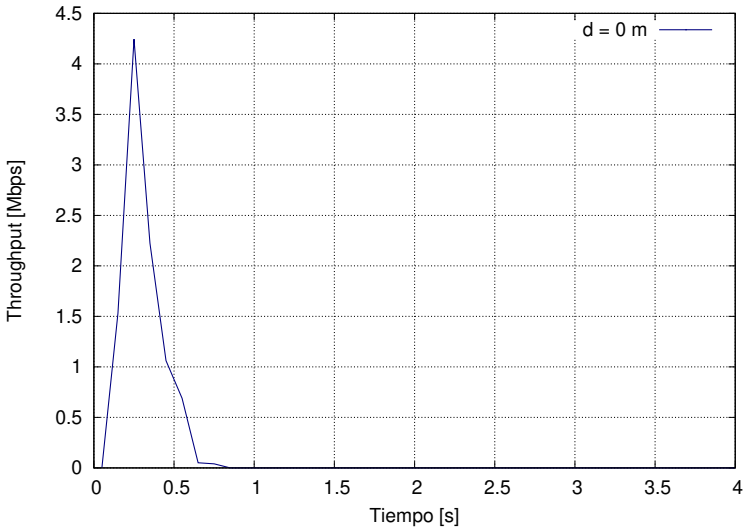


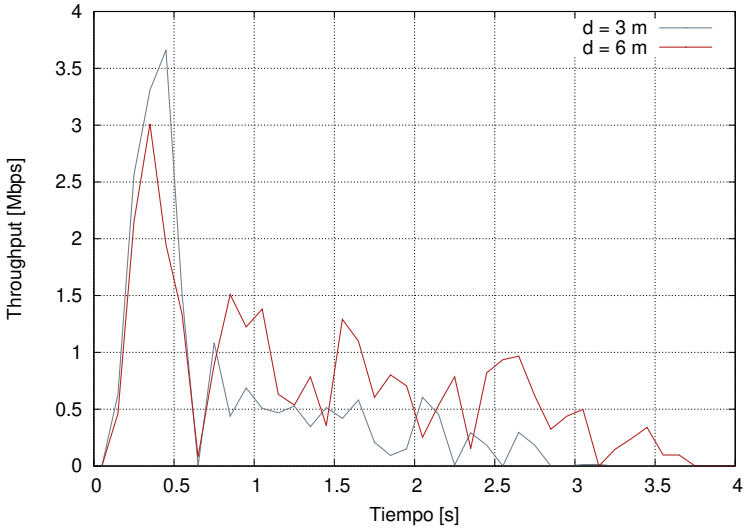
Figura 5.10: Retraso medio para envío de diferente tamaño de archivos

El análisis del *throughput* se desarrolló en el Escenario 2, sin tiempo de desconexión de nodos. Se consideró tres distancias de separación entre nodos. La Figura 5.11 presenta el *throughput* alcanzado en la transmisión para diferentes distancias de separación entre nodos. Cuando la distancia de separación es cero, es decir, el vehículo ha completado el proceso de aterrizaje, la transmisión de datos se efectúa en menos de un segundo, como muestra la Figura 5.11a. Por otro lado, cuando el vehículo está en vuelo con una distancia de separación de 3 y 6 metros, la transmisión de información se realiza hasta en 4

segundos, debido a las retransmisiones que se producen, como lo muestra la Figura 5.11b. Además, se observa que el *throughput* máximo alcanzado se reduce a medida que la separación entre nodos aumenta.



(a) UAV en tierra, $d = 0m$



(b) UAV en vuelo, $d = 3$ y $d = 6m$

Figura 5.11: Throughput en el envío de archivos para diferentes distancia

La probabilidad de entrega se analizó utilizando el Escenario 1 y modificando el tiempo de desconexión del nodo 1 (servidor). Este tiempo de desconexión se refiere al tiempo que el servidor no está conectado al nodo móvil. La desconexión se produce porque el nodo móvil se encuentra recolectando datos en un nodo fuera del rango de comunicación inalámbrica del servidor. La Tabla 5.3 presenta tiempos de desconexión de 1 minuto, 3 minutos y 10 minutos, intervalos de tiempo de entrega y el número de paquetes enviados para el desarrollo de tres transmisiones por tiempo de desconexión.

Tabla 5.3: Intervalo de tiempo de entrega con desconexión

Tiempo de desconexión [min]	Prueba	Intervalo de tiempo [s]	Número de paquetes
1	1	0,34	260
	2	1,26	595
	3	3,63	1529
3	1	0,85	260
	2	1,57	546
	3	8,95	2861
10	1	1,16	546
	2	1,65	595
	3	4,47	1529

En cada prueba se envió un archivo de diferente tamaño. Los resultados muestran que a medida que el tamaño de archivo crece, el tiempo de entrega aumenta. Sin embargo, el valor de tiempo de entrega es menor que el tiempo obtenido en los experimentos de retraso medio. Esta reducción en el intervalo de tiempo de entrega se debe a que la transmisión se realiza de forma directa desde el nodo móvil, que almacena los paquetes enviados desde el nodo estático. Además, los archivos enviados llegan al servidor sin ningún error estableciendo la probabilidad de entrega en 1, comprobando que el proceso de almacenamiento y reenvío de DTN funciona correctamente en el escenario planteado.

Capítulo 6

Conclusiones y Recomendaciones

En este capítulo se presentan las contribuciones derivadas del desarrollo y experimentación del presente trabajo (Sección 6.1). A continuación, se presentan algunas recomendaciones a considerar al experimentar con un UAV (Sección 6.2). Para finalizar el capítulo se exponen diversos problemas suscitados en el desarrollo del trabajo y sus posibles soluciones (Sección 6.3), además de trabajos futuros (Sección 6.4) que puedan derivarse de la temática abordada en este proyecto.

6.1. Conclusiones

Este trabajo de titulación es una primera aproximación para el desarrollo de aplicaciones que exploten la capacidad de adaptación de un vehículo aéreo no tripulado en aplicaciones que desplieguen redes de sensores. En esta primera aproximación se logró emplear una cámara para obtener características de un marcador *ArUco*, y usarlas para diseñar algoritmos que controlen el movimiento del dron en la etapa de aterrizaje y la recolección de datos, y evaluar el impacto del entorno en los algoritmos que conforman el sistema de aterrizaje. Además, se consiguió introducir el aterrizaje como alternativa para reducir el consumo energético en la recolección de datos usando drones, evidenciar el funcionamiento de una red DTN, contemplarla como una opción valida para la recolección de datos en redes de sensores, y determinar las principales limitaciones presentes en el desarrollo de aplicaciones que involucren vehículos aéreos no tripulados.

En la revisión de la literatura se encontró que existen varios algoritmos de aterrizaje autónomo. Estos algoritmos se enfocan en controlar de manera precisa los movimientos del vehículo hasta completar el aterrizaje en una plataforma determinada. Todos los algoritmos emplean visión artificial para obtener y validar la información del entorno. Por otro lado, los algoritmos de recolección eficiente de datos, se centran en optimizar los recursos de la red, tanto en la estructura, configuración y consumo de energía de los nodos, como en el cálculo de ruta del vehículo al efectuar la recolección de datos.

En función de la literatura analizada se definió como métrica para evaluar el aterrizaje a la distancia medida entre el centro de la cámara del vehículo y la plataforma de aterrizaje. Además, se definieron tres parámetros: retraso de paquetes, *throughput* y probabilidad de entrega como métricas de desempeño

de red. Con estos parámetros se evaluó el impacto que tiene la distancia de separación entre nodos y el vuelo del vehículo en el almacenamiento, transmisión y entrega de paquetes en la red DTN.

El desarrollo de aplicaciones para sistemas que emplean vehículos aéreos no tripulados presenta retos asociados a la variabilidad de *hardware*. La compatibilidad de sensores y actuadores depende del *firmware* cargado en el controlador. El algoritmo de aterrizaje diseñado en este trabajo emplea una cámara para capturar imágenes del marcador, obtener información del entorno y emplearla para extraer características que serán usadas para controlar el movimiento del vehículo. El sistema de control de aterrizaje para recolección de datos se centró en la ubicación de un marcador *ArUco*.

El proceso de aterrizaje y recolección de datos se estructuró estableciendo siete algoritmos: cuatro algoritmos de vuelo, un algoritmo de calibración para la cámara, un algoritmo de detección del marcador *ArUco*, y un algoritmo de recolección de datos en una red tolerante a retrasos.

La detección del marcador se considera exitosa al evaluar la influencia de factores como iluminación, y altura de detección. Los resultados mostraron que la influencia de estos factores depende exclusivamente de las características del dispositivo de captura empleado.

Si bien los algoritmos de visión artificial otorgan ventajas para el desarrollo de procesos en el dron, el consumo de potencia asociado a este proceso representa una carga adicional para el consumo de energía. Este proyecto se centró en el desarrollo de algoritmos de visión artificial que garanticen una mayor precisión en el proceso aterrizaje para efectuar recolección de datos. Se introduce al aterrizaje como una potencial solución para reducir el consumo energético del vehículo durante la transmisión de información.

La calibración minuciosa de sensores y optimización de parámetros del piloto automático permiten reducir el impacto que generan factores como la velocidad del viento y HDOP en el rendimiento de los diferentes algoritmos de vuelo. El análisis de comportamiento presentado durante la ejecución de pruebas aisladas, permitió ajustar parámetros de los algoritmos que mejoraron la detección del marcador con menor número de capturas, menor número de movimientos para posicionar el UAV sobre el marcador, determinar la dirección de giro *yaw* para la corrección de orientación y establecer la altura mínima de aterrizaje sin que afecte la operación de los otros algoritmos. Los algoritmos de vuelo presentaron resultados favorables, con un mínimo porcentaje de error. El algoritmo de corrección de orientación presenta un error promedio de 4.3°, mientras que el algoritmo de aterrizaje presenta una distancia media entre el centro de la cámara y el marcador de 22.86 cm.

Por otro lado, el análisis de rendimiento de ION-DTN demuestra que el proceso de recolección de datos empleando almacenamiento y retransmisión en una red DTN se alcanza sin importar el tiempo de desconexión del servidor. Para reducir el consumo de batería del UAV, el aterrizaje se presenta como la mejor opción, al proporcionar un menor tiempo de transmisión a mayores tasas de datos.

A pesar de que el desarrollo de *hardware, software* y *firmware* para drones se encuentra activo y respaldado por cientos de profesionales y empresas dedicadas, un dron aún cuenta con limitaciones. La principal limitación de un UAV es el consumo de potencia de la batería. Si bien una batería puede ser de diferente tipo y capacidad, a medida que el tamaño, número de periféricos y capacidad de la batería aumentan, el consumo de potencia necesaria para elevar, y controlar el vehículo aumenta. Generalmente, el peso total del vehículo y la capacidad de la batería son aspectos críticos para el diseño y configuración del vehículo.

6.2. Recomendaciones

- Mantener un piloto de respaldo para todas las pruebas realizadas. De esta manera, se evitan daños y altercados con terceros.
- La calibración de ESCs a través del controlador puede no tomar efecto. Emplear más de un método de calibración.
- Las líneas de distribución de energía eléctrica, al igual que, los elementos de alta tensión ocasionan lecturas erráticas en la brújula. Experimentar o volar el dron a una distancia prudente de estos elementos.
- Monitorear el vehículo a través de una estación terrestre durante todo el vuelo permite detectar y corregir errores.
- De preferencia, configurar sensores de respaldo como barómetro o GPS.
- Configurar un canal de seguridad en el radio control, que desarme el sistema de propulsión del vehículo ante eventuales fallos o errores.

6.3. Experiencias de desarrollo

Esta sección detalla algunos problemas suscitados durante el desarrollo de este trabajo. Además, se plantean posibles soluciones a estas problemáticas.

- Los *frames* plegables para drones experimentan pandeo. Colocar la batería en la parte superior del marco mitiga las vibraciones producidas por esta alteración.
- Una mala sintonización de PID deriva en movimientos bruscos al pilotar el vehículo. El modo *AUTOTUNE* puede sintonizar los valores de PID automáticamente. Sin embargo, este proceso se ve afectado si el vehículo no presenta una correcta distribución de peso.
- La distribución de peso del vehículo causa vibraciones. El centro de gravedad del vehículo debe estar lo más cercano al centro del vehículo.
- El balanceo del dron durante un vuelo estacionario es ocasionado por una mala alineación de motores y/o hélices.
- Al establecer un valor de *heading* la velocidad de giro causa pérdida de altura. Definir una dirección de giro reduce este efecto.
- En algunas versiones de *firmware* se requiere establecer un movimiento falso previo al proceso de modificación de rumbo (*heading*).
- Un ESC puede quemarse si se está usando una batería no compatible.

6.4. Trabajos futuros

Se puede derivar varios trabajos futuros siguiendo la temática planeada en este trabajo:

- Aterrizaje de precisión para efectuar procesos complementarios como reemplazo o carga de batería.
- Análisis de eficiencia energética en nodos DTN al emplear comunicación programada.

Anexo A

Configuración del vehículo aéreo no tripulado

En este anexo se presenta el montaje y configuración del UAV. El anexo inicia exponiendo la conexión de cada subsistema que conforma un *quadcopter* (Sección A.1). Luego, se especifica la forma de cargar o actualizar el *firmware* del vehículo empleando MP (Sección A.2), y se presenta la calibración de los distintos sensores del dron (Sección A.3). El anexo finaliza presentando una serie de ajustes de parámetros del vehículo para mejorar su comportamiento durante el vuelo (Sección A.4).

A.1. Conexión de subsistemas del vehículo

Para el desarrollo de este proyecto, se empleó un *quadcopter* con *frame* plegable en configuración *X*. El vehículo consta de seis subsistemas.

A.1.1. Sistema de propulsión

El sistema de propulsión de un dron está compuesto por los siguientes elementos:

- Motores
- ESCs
- Controlador de Vuelo
- Módulo de distribución de energía
- Batería

En la Figura A.1 se presenta la conexión de todos los elementos antes mencionados. Este es un esquema general, debido a que, existen placas que incorporan los cuatro ESCs y el módulo de potencia en un solo elemento. Generalmente, el módulo de distribución de energía se conecta al puerto de alimentación del controlador (*POWER*), por medio de un conector *DF-13* de 6 pines, que ofrece dos líneas de *VCC* de al menos 2,5 A, dos líneas *GND*, y dos señales provenientes de los sensores de voltaje y corriente del módulo de distribución de energía.

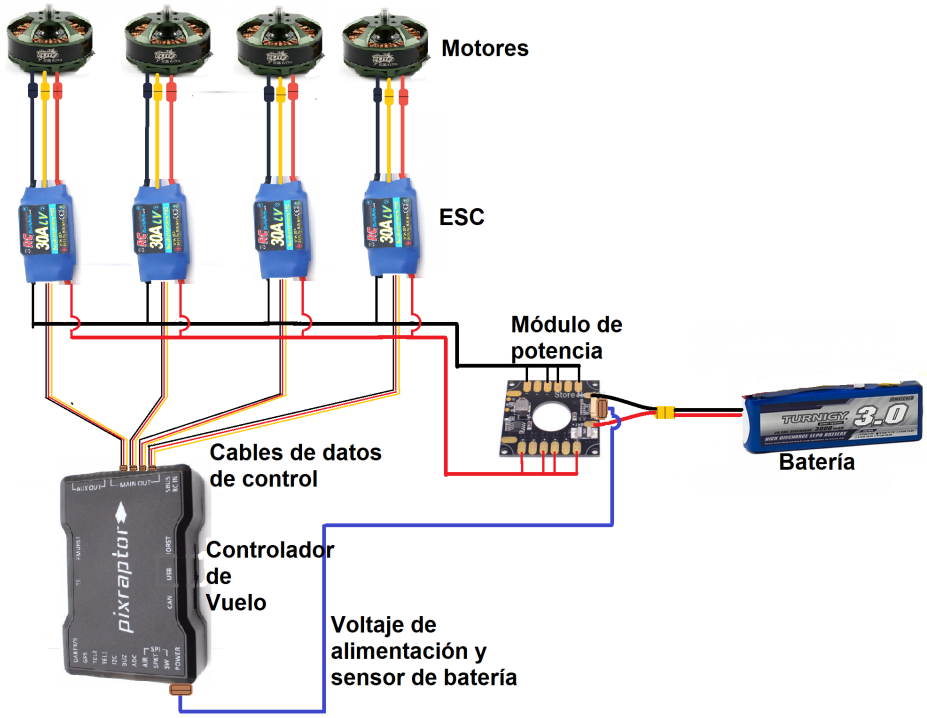


Figura A.1: Cableado y conexión de elementos del sistema de propulsión

A.1.2. Sistema de posicionamiento GPS

El controlador *PixRaptor* posee un puerto exclusivo para el GPS, y un puerto de comunicación serial (*I2C*), que puede usarse para conectar una brújula externa que ayude al sistema de posicionamiento, como muestra la Figura A.2.

Figura A.2: Conexión de GPS y brújula al controlador *PixRaptor*

A.1.3. Sistema de seguridad

El *firmware* de *Copter* en todas sus versiones requiere que el vehículo cuente con dos sistemas de seguridad, que alerten y protejan al piloto automático de eventuales fallas que puedan ocasionar algún tipo lesión a terceros o daños en la unidad. En este caso, el controlador debe contar con un *switch* de bloqueo de motores, que evite que los motores se inicien mientras no se presione el botón [74]. El

segundo elemento del sistema de seguridad es un zumbador, que emite diferentes tonos para alertar de algún problema o error en el vehículo. La conexión de estos elementos se presenta en la Figura A.3.

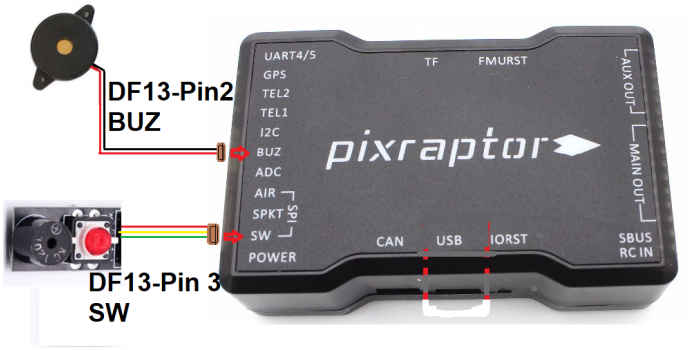


Figura A.3: Conexión de *switch* y zumbador

A.1.4. Sistema de telemetría

El sistema de telemetría permite transmitir información del controlador y sus sensores a un *host* remoto para mantener una retroalimentación del estado del piloto automático durante el vuelo [41]. El controlador de vuelo posee un puerto específico que permite conectar el radio de telemetría, como se muestra en la Figura A.4. Actualmente, existen dispositivos que pueden conectarse directamente a *smartphones*, y GCSs que se ejecutan en *Android*.

Figura A.4: Conexión de Radio de telemetría

A.1.5. Sistema de radio control

El controlador de vuelo es compatible con una gran diversidad de dispositivos de radio control. El módulo *FRSky-X8R* es un receptor de telemetría de 16 canales, compatible con múltiples protocolos de comunicación serial como: *FRSky X8R. X6R, X4RSB, XSR, XM, XM+, RX8R, L9R,* entre otros [75]. Este módulo permite mostrar información del controlador en el transmisor de radio o control remoto. *FRSky-X8R* establece comunicación serial con el controlador a través de su puerto *RC*, como se muestra en la Figura A.5.

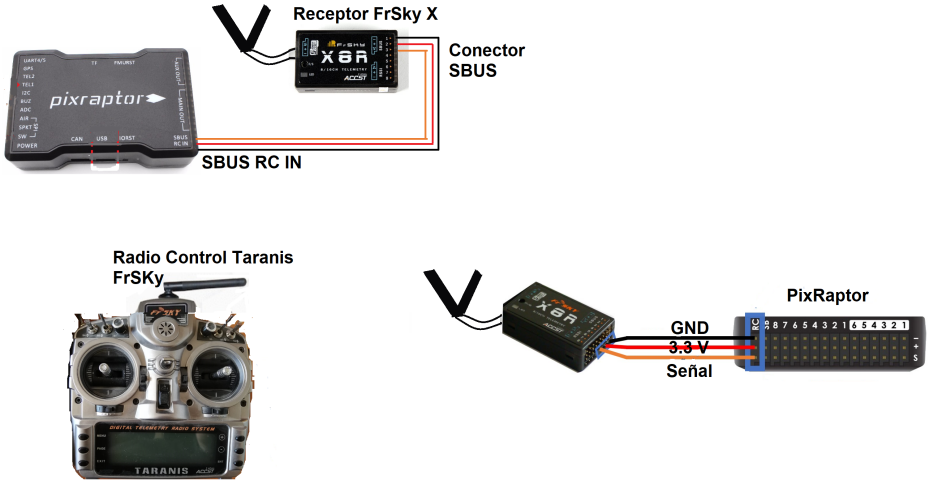


Figura A.5: Conexión del sistema de radio control

A.1.6. Sistema de estabilización de la cámara

Entre las múltiples funciones que puede cumplir un dron se encuentra la captura imágenes. No obstante, las vibraciones producidas por las hélices, y el balanceo provocado por el movimiento que experimenta el *frame* del UAV dificultan el proceso de captura de imágenes. Por tal motivo, es común el uso de un estabilizador (*gimbal*) de varios ejes, que mitigue los movimientos involuntarios de la cámara en el vehículo. Este dispositivo cuenta con una serie de gomas, que disminuyen la vibración, y uno o varios motores que corrigen la inclinación de la cámara en todos sus ejes (*pan, roll, tilt*). La Figura A.6 presenta los componentes del *gimbal* para UAV *TAROT 2D* de dos ejes. El diagrama de conexión de elementos del *gimbal* está expuesto en el manual de usuario del dispositivo [76].

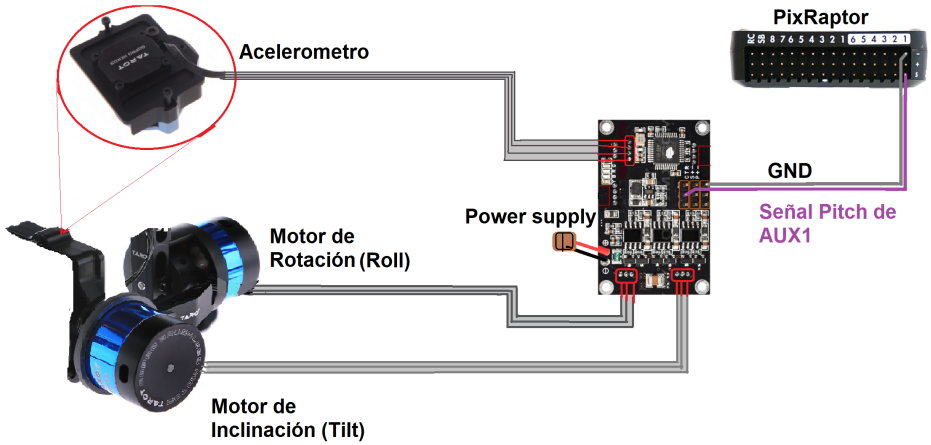


Figura A.6: Esquema de conexión de *Gimbal Tarot 2D*

A.2. Actualización de firmware

El *firmware* determina el tipo de vehículo que se desplegará. Para cargar o actualizar el *firmware* en el controlador se debe emplear el puerto micro Universal Serial Bus (USB) del dispositivo y un

puerto USB de la computadora que ejecuta MP. La Figura A.7 presenta la ubicación del puerto en el controlador *PixRaptor*. El controlador es un dispositivo con tecnología Plug and Play (PnP).

Figura A.7: Puerto micro-USB en el controlador *PixRaptor*

Para establecer conexión con el controlador, efectúe los siguientes pasos:

1. Dentro de la barra de conexiones de MP, seleccione el puerto de comunicación.
2. Defina en 9600 la tasa de baudios, como lo muestra la Figura A.8

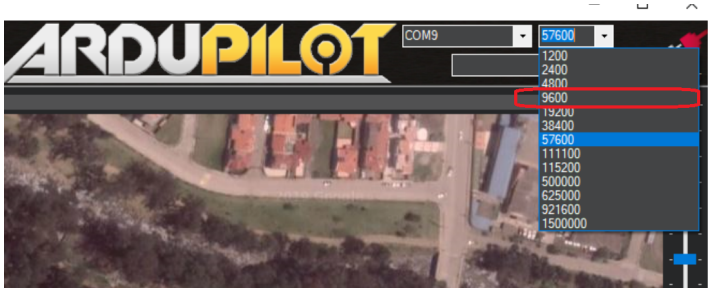


Figura A.8: Conexión con *Mission Planner*

Para cargar el *firmware* al controlador, realice lo siguiente:

1. Dentro de la ventana de MP, seleccione la pestaña (*INITIAL SETUP*).
2. Diríjase a la opción *Install Firmware*.
3. Seleccione el marco para *quadcopter* y la versión de *firmware* que instalará. De forma predeterminada, se presenta la última versión de *firmware* que *Ardupilot* tiene disponible, como se muestra en la Figura A.9.

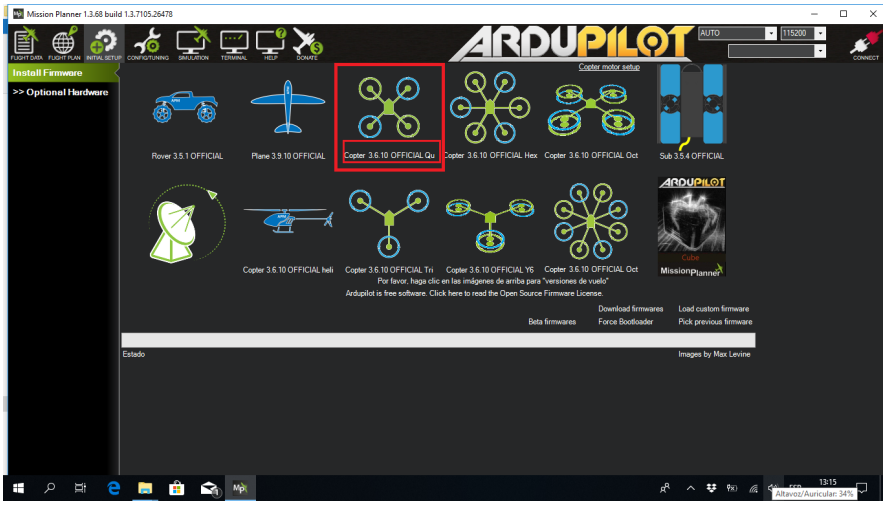


Figura A.9: Tipo de *frame* y versión de *firmware*

Para descargar e instalar una versión anterior, ubique la casilla *Pick Previous Firmware* en la parte inferior derecha, como lo muestra la Figura A.10, y seleccione la versión de *ArduCopter* que desee cargar al controlador.

Figura A.10: Instalación de versiones anteriores de *firmware*

Antes de iniciar el proceso de actualización es necesario retirar las hélices y desconectar la batería del vehículo. MP genera el mensaje de advertencia que muestra la Figura A.11 previo al inicio de descarga de *firmware*.

Figura A.11: Indicaciones para descarga e instalación de *firmware*

El proceso de instalación requiere desconectar el cable del controlador y conectarlo nuevamente. Al finalizar la instalación el controlador se reiniciará automáticamente.

A.3. Calibración de hardware externo

Instalado el *firmware*, se procede a calibrar los diferentes sensores y periféricos que intervienen en el vuelo del UAV. Conecte el controlador a MP empleando un cable USB o el módulo de telemetría. Cabe mencionar que, para utilizar el módulo de telemetría se define la tasa de baudios en 115200.

A.3.1. Acelerómetro

El acelerómetro es el dispositivo que realiza el control del nivel durante el vuelo. Para realizar la calibración siga los siguientes pasos:

1. Diríjase a la pestaña *INITIAL SETUP*.
2. Dentro de la sección *Mandatory Hardware* seleccione *Accel Calibration*.
3. Coloque el vehículo en una superficie plana y nivelada, y presione el botón *click when done* como lo sugiere el aviso en la Figura A.12.
4. Presione el botón *Calibrate Level* y espere. Cuando la calibración finalice se generará un aviso de calibración exitosa.

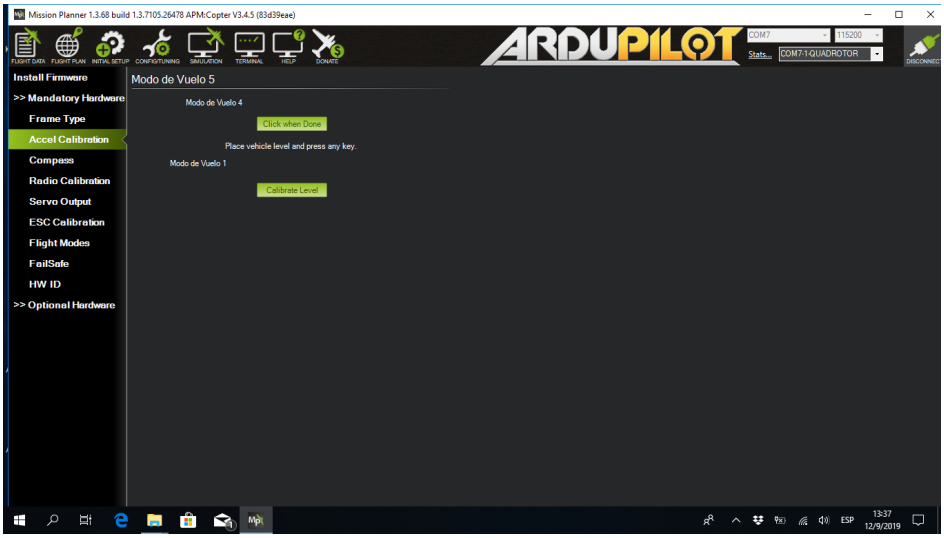


Figura A.12: Calibración de acelerómetro

A.3.2. Brújula

La brújula es la encargada de definir el rumbo del UAV durante el vuelo. Antes de iniciar el proceso de calibración, aleje todos los objetos que produzcan campos magnéticos, debido a que provocan una calibración errónea.

Para realizar la calibración de la brújula, efectúe lo siguiente:

1. Dentro de la opción *Compass* seleccione cual será la brújula primaria (*Primary Compass*).
2. Asigne *Compass1* a la brújula externa, *Compass2* a la brújula interna del controlador y *Compass3* a una segunda brújula interna en el caso de que el controlador la posea. Se puede deshabilitar el uso de cualquiera de estas a través de la opción *Use this compass*. La asignación de brújula primaria, y secundaria se muestra en la Figura A.13.
3. Inicie la calibración.

El controlador posee una brújula interna, pero se recomienda no usarla por su bajo nivel de confianza. Generalmente, se emplea para corroborar el rumbo que se obtiene de la brújula externa.

Si se presenta el error *Inconsistent Compasses* para cualquier dispositivo, se debe deshabilitar la brújula en la opción *Use this compass*.

Figura A.13: Calibración de brújula

Para una calibración exitosa el usuario debe situar el dron en las posiciones: nivel, espalda, nariz abajo, nariz arriba, lateral izquierdo, y lateral derecho. Se debe girar el vehículo en sentido antihorario en cada posición, como lo ilustra la Figura A.14.

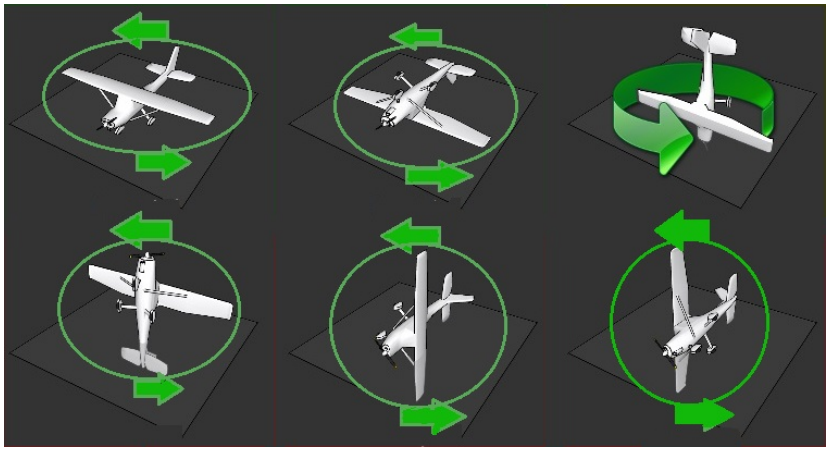


Figura A.14: Pasos para calibrar la brújula

Los valores máximos y mínimos de compensación de la brújula aparecen cuando la calibración está completa (*OFFSETS*).

A.3.3. Radio control

El transmisor de radio es el elemento que permite al piloto controlar los movimientos del vehículo en diferentes modos de vuelo. Al existir diferentes tipos de transmisores de radiofrecuencia se realiza la calibración con el objetivo de detectar y establecer los valores máximos y mínimos de los canales del transmisor. Se debe tener en cuenta las siguientes consideraciones:

- No conectar la batería al dron.
- Conectar el controlador *PixRaptor* a la computadora usando un cable USB.
- No colocar las hélices en ningún momento.
- Verificar la conexión del receptor de radio y el controlador de vuelo.

Para dar inicio al proceso de calibración, se realiza lo siguiente:

1. Encienda el radio control.
2. Diríjase a la opción *Radio Calibration* y seleccione *calibrate*.
3. Durante el proceso de calibración las palas, interruptores y perillas del transmisor deben moverse hacia todas sus posiciones. Las barras de color verde que se muestran en la Figura A.15 se moverán indicando el valor máximo y mínimo leído por el receptor para cada canal.
4. Emplee estas barras para determinar la asignación de canales en el radio control. Para algunos canales la magnitud de su valor detectado se ve modificado empleando combinaciones de interruptores.

Figura A.15: Detección de canales del radio control

A.3.4. Controlador de velocidad electrónico

Un ESC es el elemento encargado de controlar la velocidad de giro de cada motor del UAV. La calibración del controlador de velocidad depende del tipo y marca del dispositivo. En esta sección mencionaremos dos métodos de calibración.

A.3.4.1. Calibración manual

Este método de calibración hace uso del radio control y el receptor *FrSky*. Para iniciar la calibración, desconecte el cable de control del ESC, conéctelo al canal del acelerador (*Throttle*) y siga los siguientes pasos:

1. Desconecte la batería.

2. Una vez conectado el cable de control del ESC al receptor, encienda el transmisor de radio y mueva la pala del acelerador a su valor máximo.
3. Conecte la batería y escuchará un tono largo seguido por dos tonos cortos.
4. Mueva la pala del acelerador hasta su valor mínimo.
5. Escuchará una serie de tonos cortos acorde al número de celdas de la batería y un tono largo.
6. Desconecte la batería y repita el proceso para todos los ESCs.

A.3.4.2. Calibración semiautomática

1. Conecte el piloto automático a MP.
2. Dentro de la sección *ESC Calibration* de *Mandatory Hardware*, identifique el tipo de ESC: *Normal*, *DShot* , *BLHeli* y *DJI Opto* y selecciónelo.
3. Inicie la calibración.
4. Desconecte la batería y el cable USB.
5. Conecte nuevamente la batería.
6. Espere la reproducción del tono de armado.
7. Presione el botón de seguridad hasta que tome un color sólido y se presente un tono constante seguido por dos tonos cortos.
8. Después de algunos segundos escuchará una serie de pitidos acorde al número de celdas de la batería.
9. Desconectar y conectar la batería para finalizar el proceso.

A.3.5. Modos de vuelo

El *firmware* instalado en el controlador *PixRaptor* pone a disposición 23 modos de vuelo. El número de modos de vuelo que se puede configurar depende del transmisor de radio. La Tabla A.1 presenta los modos de vuelo manuales y automáticos considerados en la experimentación de este proyecto. La tabla detalla brevemente las características del modo de vuelo: control de altitud, control de posición, y requerimiento de lectura de posicionamiento GPS.

Tabla A.1: Modos de vuelo

Modo	Control de altitud	Control de posición	GPS	Descripción
Alt Hold	Piloto controla la velocidad de ascenso	Control manual	No	Mantiene la altitud y controla el nivel de *roll* y *pitch*
Auto	Automático	Automático	Si	Ejecuta una misión creada en la estación de control en tierra
Auto Tune	Piloto controla la velocidad de ascenso	Automático / corrección manual	Si	Afinación automática de parámetros para mejorar los bucles de control
Guided	Automático	Automático	Si	Movimientos controlados mediante comandos de protocolo
Land	Automático	Automático	Si	Inicia el descenso del vehículo. Finaliza al llegar a tierra
Loiter	Piloto controla la velocidad de ascenso	Piloto controla los movimientos	Si	Mantiene la altitud, posición y usa el GPS para controlar el movimiento
PosHold	Piloto controla la velocidad de ascenso	Piloto controla los movimientos con limites preajustados	Si	Control manual de balanceo y cabeceo cuando los mandos (palas) no están centrados
Return To Launch	Automático	Automático	Si	Regresa al punto de despegue. Incluye aterrizaje
Stabilize	Control manual total	Piloto controla los movimientos con limites preajustados	No	Auto nivela los ejes de balanceo. Control total de los movimientos

Para configurar los modos de vuelo en el radio control empleando MP siga los siguientes pasos:

1. Dentro de la sección *Mandatory Hardware*, diríjase a la opción *Flight Modes*.
2. Seleccione el modo de vuelo que desea configurar. MP permite configurar al menos seis modos de vuelo para un radio control específico.
3. Mueva los interruptores del radio control a fin de determinar el canal asociado al modo de vuelo.
4. Reinicie el controlador para guardar los cambios.

A.3.6. Monitor de batería

El nivel de batería es el parámetro de telemetría que otorga información en tiempo real del voltaje y corriente de la batería. Esta información es provista por el monitor de energía o sensor de alimentación,

que mide el voltaje y corriente de la batería. *ArduPilot* tiene compatibilidad con diversos módulos de potencia y energía.

- Módulo de energía común: módulo de alimentación analógico que proporciona una fuente de alimentación estable al controlador de vuelo. Además, permite el monitoreo de consumo de voltaje y corriente.
- Módulo de energía *AirbotPower*: módulo empleado para sistemas de alta potencia, en donde intervienen baterías de 4 a 8 celdas y corriente de 90 a 150 amperios. Esta placa evita el cableado de distribución para cada ESC.
- *Mauch Power Monitor:* es un sensor de corriente de campo magnético. Este sensor consume una cantidad menor de potencia con respecto a los dos anteriores y no se ve afectado por la temperatura del sistema.

A.3.7. Estabilizador de cámara

Un *gimbal* es un soporte giratorio de varios ejes empleado para mantener o estabilizar la posición de una cámara. *Ardupilot* permite emplear *gimbals* de hasta 3 ejes. Los ejes funcionales que pueden configurarse en MP se muestran en la Figura A.16. El *gimbal* se conecta al controlador de vuelo y a una fuente de alimentación con valores de voltaje en el rango de 7,4 V a 14,8 V, el valor recomendado es 12 V. En este caso, se configuró únicamente el movimiento de *tilt* para la cámara incorporada en el dron. Para efectuar la configuración del *gimbal* siga los siguientes pasos:

1. Ubíquese en la pestaña *INITIAL SETUP*, seleccione la sección *Optional Hardware* y elija la opción *Camera Gimbal*.
2. Establezca el tipo de dispositivo en *SERVO*.
3. Fije la inclinación de *tilt* en *SERVO9*.
4. Seleccione el canal de comunicación para controlar la posición de *tilt* con el control de radio, generalmente es el *CH6*.
5. Establezca los límites de ángulo y *SERVO* (*Servo Limits* y *Angle Limits*).

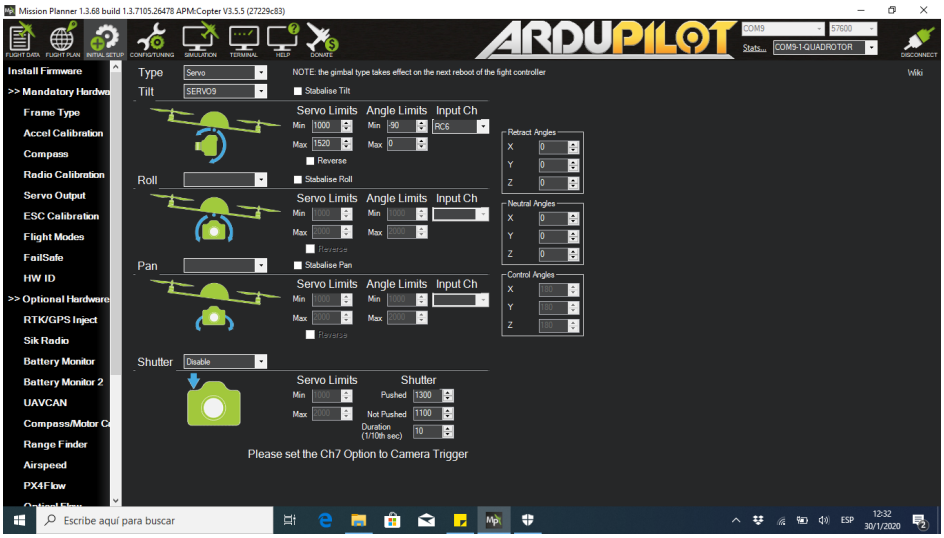


Figura A.16: Activación y calibración de *gimbal* para cámara

A.4. Puesta a punto o *Fine Tuning*

La puesta a punto del UAV está conformada por una serie de acciones que se ejecutan para ajustar parámetros de funcionamiento del vehículo. Dichas acciones permiten tener un mayor control y una mejor experiencia de vuelo, reduciendo el riesgo de fallas o errores.

A.4.1. Máximo ángulo de inclinación

La velocidad de vuelo del UAV está relacionada con los ángulos *pitch* y *yaw*. Estos dos ángulos permiten establecer un límite de máxima inclinación, que determina la velocidad de vuelo. Para modificar estos parámetros se requiere acceder a la lista completa de parámetros del vehículo en MP. Los parámetros a considerar son:

- **`BAL PITCHMAX` BalanceBot max:** ángulo de inclinación expresado grados cuando la pala de inclinación *pitch* se encuentra en su máximo valor.
- **`CRASH ANGLE`:** límite de ángulo de inclinación de *pitch* y *roll*.

A.4.2. Sintonización automática

La función *AutoTune* tiene como objetivo sintonizar de manera automática los valores de ganancia P, D y la aceleración máxima de rotación, para proporcionar una respuesta confiable sin sobre-impulso significativo, que afecte la altura y el control del vehículo. Algunas perturbaciones que pueden originar valores erróneos en el proceso de sintonización son:

- Giroscopio en mal estado.
- Marco experimenta pandeo.
- Carga no óptima (motores sobrecargados).

Se puede optar por sintonizar un eje de movimiento a la vez, modificando los siguientes parámetros:

- `AUTOTUNE_AXES`: se define uno a uno los ejes de afinación, 1 para *Roll*, 2 para *Pitch*, y 3 para *Yaw*.
- `AUTOTUNE_AGGR`: configuración de la agresividad de movimiento (0.1-agresivo, 0.075-medio,y 0.050-débil).

A.4.3. Configuraciones de respaldo

Controladores como *PixHawk* o *PixRaptor* permiten conectar sensores externos de respaldo, que ayudan al UAV en el caso de presentar fallas.

A.4.3.1. Sistema de posicionamiento

Algunos controladores permiten la conexión dual de módulos GPS. Esta configuración reduce la posibilidad de errores relacionados con la lectura de posicionamiento del vehículo. La conexión dual se encuentra disponible para versiones de *firmware Copter 3.5* o superiores. Cabe recalcar que los dispositivos conectados deben ser del mismo fabricante [77].

La configuración se realiza por medio de la estación terrestre, modificando los parámetros:

- `GPS_TYPE2 = 1`: permite al sistema detectar automáticamente el tipo de GPS que se encuentra conectado.

- `SERIAL4_PROTOCOL = 5`: en el caso de que el controlador cuente con un puerto serial adicional, como muestra la Figura A.17. Caso contrario, se puede emplear el puerto de telemetría 1 o 2 (`SERIAL1_PROTOCOL = 5` o `SERIAL2_PROTOCOL = 5`).
- `GPS_AUTO_SWITCH`: 1 (el valor de posición es una combinación entre los datos adquiridos por los dos GPS), 2 (usa la mejor lectura de uno de los dos GPS, el que tenga menor valor de dilusión).

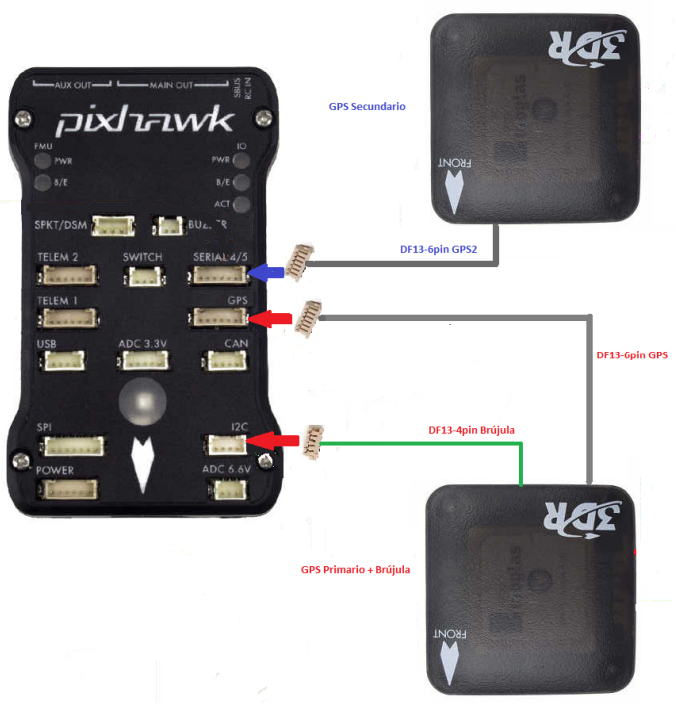


Figura A.17: Diagrama de conexión de GPS Dual

A.4.3.2. Barómetro

El controlador de vuelo *PixRaptor* incorpora un medidor de presión (barómetro) que le permite determinar la altura de vuelo. Sin embargo, debido a perturbaciones generadas por vibración, en algunas ocasiones se requiere mantener una lectura barométrica adicional. Esta configuración solo se encuentra disponible para versiones de *firmware Copter 4.0*. Se debe modificar los parámetros: [78]:

- `GND_PROBE_EXT`: 4 para barómetro *Drotek MS5611* o 2 para barómetro *Adafruit BMP280*.
- `GND_PRIMARY =1` establece el barómetro externo como primario.

Anexo B

Configuración del dispositivo Raspberry Pi

Este anexo detalla el proceso de instalación y configuración de *software* del dispositivo *Raspberry Pi*. En primer lugar, se describe el proceso de descarga, escritura y configuración del sistema operativo *Raspbian Buster* (Sección B.1). A continuación, se presenta el proceso de instalación de la biblioteca *OpenCV* (Sección B.2) y el paquete *Dronekit* de *Python* (Sección B.3). La Sección B.4 presenta la instalación y configuración de la estación terrestre *MavProxy*, necesaria para establecer comunicación con el controlador de vuelo. Finalmente, se presenta la instalación y configuración de ION-DTN, necesario para la transmisión de datos en los diferentes nodos (Sección B.5).

B.1. Sistema operativo

Raspbian es un sistema operativo libre basado en *Debian GNU/Linux* optimizado para operar de manera eficiente con el hardware de una *Raspberry Pi*. El proyecto se encuentra en constante desarrollo gracias a equipos de fanáticos de la plataforma y expertos que forman parte del proyecto *Debian* [79].

La Fundación *Raspberry Pi* permite descargar una imagen del sistema operativo para sus dispositivos desde `https://www.raspberrypi.org/downloads/raspbian/`. En este sitio web aparece la última versión de *Raspbian*. En este caso se seleccionó *Raspian Buster* en su versión GUI, una imagen basada en *Debian Buster* por su mayor seguridad y adaptabilidad a dispositivos de modelos más recientes.

Una vez descargada la imagen del sistema operativo se puede utilizar comandos desde la terminal para copiar y descomprimir la imagen. Para copiar la imagen descomprimida a una tarjeta microSD es necesario emplear el comando `dd` expuesto en el Código B.1. El parámetro `if` señala la ruta de la imagen o archivo de entrada, mientras que el parámetro `of` establece el dispositivo de almacenamiento destino o archivo de salida en el que se copiará la imagen.

```
1 sudo dd bs=4M if=2019-07-10-raspbian-buster.img of=/dev/mmcblk0 status=
      progress conv=fsync
```

Código B.1: Escritura de la imagen de *Raspbian* en un dispositivo

Para obtener el nombre del dispositivo destino, conéctelo al ordenador y ejecute el comando `sudo fdisk -l` o `sudo lsblk`. Podrá observar los distintos dispositivos de almacenamiento del sistema y sus particiones.

Si no ha podido descomprimir la imagen de *Raspbian* puede utilizar el Código B.2 para descomprimir y escribir la imagen en una tarjeta microSD.

```
unzip -p 2019-07-10-raspbian-buster.zip | sudo dd of=/dev/mmcblk0 bs=4M
    status=progress conv=fsync
```

Código B.2: Escritura de la imagen de *Raspbian* desde archivo comprimido

Si fallase alguno de los comandos antes mencionados, *Ubuntu* cuenta con una aplicación para gestionar los discos del sistema denominada *Discos*. Para montar la imagen de *Raspbian* en una tarjeta microSD basta con descomprimir el fichero descargado, desplegar el menú de opciones del archivo y seleccionar la opción *Abrir con... Escritor de imágenes de disco*.

El comando `dd` permite además crear una imagen con el contenido (*backup*) de una tarjeta microSD. Para ello, es necesario utilizar el comando expuesto en el código B.3. El archivo generado está comprimido.

```
sudo dd bs=4M if=/dev/mmcblk0 | gzip > imageFEB2020.img.gz
```

Código B.3: Crear imagen de respaldo de *Raspbian*

La primera interacción con el sistema operativo precisa de al menos tres periféricos para el ingreso de comandos y configuraciones iniciales. No obstante, la configuración de red y herramientas de conexión remota entre dispositivos permiten reducir el número de periféricos necesarios para trabajar con *Raspbian*.

Para lograr esto, es preciso usar Secure SHell (SSH), un protocolo que emplea una arquitectura cliente-servidor para establecer comunicaciones seguras/cifradas [80] a través de sesiones entre dos sistemas. En todas las distribuciones de *Linux* el protocolo SSH se encuentra de forma nativa en el sistema. Para establecer una sesión con un *host* remoto es necesario definir un usuario y un *host*, como se expone en el Código B.4.

```
ssh {usuario}@{host}
```

Código B.4: Sintaxis del comando ssh

En este caso concreto, la mejor forma de emplear este comando es configurar la *Raspberry Pi* como un punto de acceso, y mantener a la computadora cliente en la misma red. Existen diferentes métodos para establecer una sesión remota con la *Raspberry Pi*. Sin embargo, se seleccionó este método debido a que la interfaz inalámbrica permite el monitoreo de procesos durante el vuelo y aterrizaje del vehículo.

Para crear un punto de acceso en la placa necesitamos de un servidor Domain Nane Server (DNS) (*DNSMasq*) y *HostAPD*, una herramienta capaz de utilizar una interfaz de red inalámbrica como punto de acceso. La instalación de estos dos componentes se logra a través del Código B.5.

```
sudo apt install dnsmasq hostapd
```

Código B.5: Instalación de *DNSMasq* y *HostAPD*

Es necesario detener el servicio de estos paquetes de *software* para modificar los archivos de configuración necesarios. El Código B.6 expone los comandos necesarios para esta acción.

```
sudo systemctl stop dnsmasq
sudo systemctl stop hostapd
```

Código B.6: Interrupción del servicio de *DNSMasq* y *HostAPD*

La *Raspberry Pi* debe tener una dirección IP estática asignada a la interfaz de red inalámbrica, que por defecto se denomina `wlan0` dentro del sistema operativo. Para establecer una dirección IP fija, debemos editar el archivo de configuración del servidor Dynamic Host Configuration Protocol (DHCP) del dispositivo (*dhcpd*), usando el Código B.7.

```
sudo nano /etc/dhcpcd.conf
```

Código B.7: Edición del archivo de configuración de *dhcpd*

Dentro del archivo de configuración, asigne una dirección IP estática a la interfaz de red junto a una máscara de red, como se muestra en el Código B.8.

/etc/dhcpd.conf

```
interface wlan0
    static ip_address=192.168.0.1/24
    nohook wpa_supplicant
```

Código B.8: Contenido del archivo *dhcpd.conf*

Reinicie el servicio de *dhcpd* usando el Código B.9 para que tome efecto la configuración realizada.

```
sudo service dhcpcd restart
```

Código B.9: Reinicio del servicio de *dhcpd*

El servicio DHCP para el punto de acceso será proporcionado por *DNSMasq*. En primer lugar, es necesario declarar la interfaz de red que estará asociada al servicio, el rango de direcciones IP disponibles, la máscara de red y el tiempo máximo de conexión por dispositivo. Para acceder al archivo de configuración utilice el Código B.10.

```
sudo nano /etc/dnsmasq.conf
```

Código B.10: Edición del archivo de configuración de *DNSMasq*

El contenido del archivo de configuración se presenta en el Código B.11.

/etc/dnsmasq.conf

```
interface=wlan0
dhcp-range=192.168.0.2,192.168.0.20,255.255.255.0,24h
```

Código B.11: Contenido del archivo *dnsmasq.conf*

La interfaz de red cableada (`eth0`) se omite dentro de esta configuración, puesto que, será utilizada como salida a internet para la instalación y configuración de otros paquetes necesarios. Para finalizar con la configuración del servicio, inicie *DNSMasq* con el Código B.12.

```
sudo systemctl start dnsmasq
```

Código B.12: Inicio del servicio de *DNSMasq*

Para finalizar la configuración del punto de acceso, se debe definir los parámetros de la red inalámbrica. Abra el archivo de configuración de *HostAPD* usando el Código B.13.

```
sudo nano /etc/hostapd/hostapd.conf
```

Código B.13: Edición del archivo de configuración de *HostAPD*

Dentro de este archivo, defina la interfaz de red inalámbrica como la interfaz ligada al servicio, asigne un identificador de red (*SSID*), una contraseña, tipo de seguridad y cifrado, entre otros parámetros. El contenido del archivo de configuración se presenta en el Código B.14.

/etc/hostapd/hostapd.conf

```
interface=wlan0
driver=nl80211
ssid=raspiuav
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=raspiuav1
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Código B.14: Contenido del archivo *hostapd.conf*

Para que la configuración se efectúe con éxito es necesario definir la ruta del archivo de configuración. Utilice el Código B.15 para abrir el archivo de inicio de *HostAPD*.

```
sudo nano /etc/default/hostapd
```

Código B.15: Edición del archivo de inicio de *HostAPD*

Localice la línea que contenga el parámetro `DAEMON_CONF` y coloque la ruta del archivo de configuración: `DAEMON_CONF=``/etc/hostapd/hostapd.conf''`. Para finalizar, habilite e inicie el servicio de *HostAPD* con los comandos expuestos en el Código B.16, y reinicie el dispositivo para que el punto de acceso sea visible.

```
sudo systemctl unmask hostapd
sudo systemctl enable hostapd
sudo systemctl start hostapd
```

Código B.16: Inicio del servicio de *HostAPD*

Para acceder de forma remota a la *Raspberry Pi*, utilice un terminal, defina el usuario como **pi** y el *host* como la dirección IP del punto de acceso creado. La contraseña por defecto es *raspberry*. Emplee la sintaxis expuesta en el Código B.4. Puede modificar el usuario y contraseña por defecto usando `sudo raspi-config`. Otras alternativas de software para conexión remota son: *VNC Viewer* y *PuTTY*

B.2. OpenCV

La instalación de OpenCV en una distribución de *Linux* comúnmente se realiza compilando desde la fuente. Sin embargo, desde finales del año 2018 se encuentra disponible una instalación más rápida para la instalación de esta biblioteca. Cabe recalcar que, esta instalación no establece una versión completa. Puede que para algunos proyectos requiera la instalación completa desde la fuente. Las siguientes instrucciones se realizaron en *Raspbian Buster* y se recomiendan para placas *Raspberry Pi 3B/3B+/4*.

En primer lugar, es necesario expandir el sistema de archivos para emplear todo el almacenamiento de la tarjeta microSD. Para ello, utilice el comando del Código B.17, diríjase a la opción *Advanced Options*, y luego a, *Expand Filesystem*.

```
sudo raspi-config
```

Código B.17: Expansión del sistema de archivos

A continuación, reinicie el dispositivo con el comando del Código B.18. Puede verificar el resultado del paso anterior usando `df -h` en una terminal.

```
sudo reboot
```

Código B.18: Reiniciar *Raspberry Pi*

El siguiente paso es instalar dependencias. En primer lugar es necesario actualizar los paquetes existentes con el Código B.19.

```
sudo apt-get update && sudo apt-get upgrade
```

Código B.19: Actualizar paquetes existentes

A continuación, instale paquetes esenciales de desarrollo, una interfaz para llamar bibliotecas y el compilador `CMAKE` con el Código B.20.

```
sudo apt-get install build-essential cmake pkg-config
```

Código B.20: Instalar paquetes esenciales de desarrollo

Luego, instale paquetes para gestionar entrada y salida de imágenes desde el disco. Estos paquetes incluyen diversos formatos de archivos de imagen. De forma similar, necesitamos paquetes para gestionar archivos de video. Utilice los comandos del Código B.21.

```
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-
    dev
sudo apt-get install libxvidcore-dev libx264-dev
```

Código B.21: Instalar paquetes de gestión de imagen y video

Highgui es un submódulo de OpenCV que sirve para crear GUI básicas. Para instalar esta herramienta necesitamos del kit de desarrollo *GTK*. Utilice el Código B.22 para instalar estas dependencias.

```
sudo apt-get install libfontconfig1-dev libcairo2-dev
sudo apt-get install libgdk-pixbuf2.0-dev libpango1.0-dev
```

```
sudo apt-get install libgtk2.0-dev libgtk-3-dev
```

Código B.22: Instalar *Highgui* usando el kit *GTK*

Para optimizar algunas operaciones algebraicas es necesario instalar *ATLAS* y *Gfortran*. Emplee el Código B.23 para ello.

```
sudo apt-get install libatlas-base-dev gfortran
```

Código B.23: Instalar *ATLAS* y *Gfortran*

Para el manejo, procesamiento, almacenamiento de datos, e integración de sistemas de ventanas, eventos, gráficos e imágenes instalaremos *HDF5* y *GUI Qt*. Emplee el Código B.24 para este propósito.

```
sudo apt-get install libhdf5-dev libhdf5-serial-dev libhdf5-103
sudo apt-get install libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5
```

Código B.24: Instalar *HDF5* y *GUI Qt*

Luego, instale archivos de encabezado para construir módulos, o incrustar *Python* en aplicaciones. El Código B.25 permite instalar encabezados para compilar OpenCV con enlaces de *Python*.

```
sudo apt-get install python3-dev
```

Código B.25: Instalar encabezados para *Python*

A continuación, cree un entorno virtual de *Python*. Utilice el Código B.26 para descargar e instalar *pip* en el sistema.

```
wget https://bootstrap.pypa.io/get-pip.py
sudo python3 get-pip.py
sudo rm -rf ~/.cache/pip
```

Código B.26: Instalar PIP

El siguiente paso es instalar *virtualenv* y *virtualenvwrapper*. Emplee el Código B.27.

```
sudo pip install virtualenv virtualenvwrapper
```

Código B.27: Instalar *virtualenv* y *virtualenvwrapper*

Es necesario modificar el archivo *bashrc*. Ábralo usando el Código B.28.

```
nano ~/.bashrc
```

Código B.28: Modificar el archivo *bashrc*

En la parte final del archivo agregue el contenido del Código B.29

```
export WORKON_HOME = $HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON = /usr/bin/python
source /usr/local/bin/virtualenvwrapper.sh
```

Código B.29: Exportar rutas de entorno virtual

Vuelva a cargar el archivo `bashrc` para aplicar los cambios realizados. Utilice el Código B.30.

```
source ~/.bashrc
```

Código B.30: Cargar del archivo *bashrc*

A continuación, cree un entorno virtual denominado *CV*. El Código B.31 se emplea para crear un entorno virtual en *Python*.

```
mkvirtualenv cv -p python3
```

Código B.31: Crear un entorno virtual en *Python*

Finalmente, puede instalar OpenCV empleando `pip` dentro del entorno virtual creado. Para ello emplee el Código B.32 dentro de **CV**. Si lo desea puede asignar una versión específica para la instalación como se muestra en el Código B.33.

```
pip install opencv-contrib-python
```

Código B.32: Instalar *OpenCV* usando PIP

```
pip install opencv-contrib-python == 3.4.6.27
```

Código B.33: Instalar una versión específica de *OpenCV* usando PIP

Este proceso puede tardar unos minutos. Al finalizar, puede verificar la versión de OpenCV instalada usando el Código B.34.

```
workon cv
python3
>>> import cv2
>>> cv2.__version__
```

Código B.34: Verificación de versión de *OpenCV*

Si este proceso no funciona, debe compilar OpenCV desde la fuente.

B.3. Dronekit

Dronekit se instala empleando `pip` en las distribuciones de *Linux*. Utilice el Código B.35 para instalar *Dronekit*.

```
sudo pip install dronekit
```

Código B.35: Instalar *Dronekit* usando pip

Dronekit también proporciona un simulador de piloto automático (SITL) reducido, que proporciona comandos básicos para ejecutar binarios preconstruidos para diferentes tipos de vehículos. Actualmente solo proporciona binarios para plataformas x86. Por tanto, *Dronekit-SITL* no puede ser ejecutado en plataformas *ARM* como *Raspberry Pi*[81]. Puede instalar *Dronekit-SITL* en una computadora para emplearla como simulador de vuelo y experimentar con las aplicaciones desarrolladas. El Código B.36 presenta el comando necesario para la instalación del simulador en *Linux*.

```
sudo pip install dronekit-sitl
```

Código B.36: Instalar *Dronekit-sitl* usando pip

En el Anexo C se presenta la instalación y configuración de SITL para *Linux*. Este simulador cuenta con más funcionalidades y herramientas. Sin embargo, si desea emplear *Dronekit-sitl* puede obtener una revisión detallada de su configuración en su repositorio oficial.

Dronekit proporciona una amplia variedad de ejemplos para analizar el comportamiento de su API. El Código B.37 permite obtener una copia del repositorio de ejemplos.

```
git clone http://github.com/dronekit/dronekit-python.git
```

Código B.37: Clonar repositorio de ejemplos de *Dronekit*

B.4. Comunicación con el controlador de vuelo

Raspberry Pi se comunica con el controlador de vuelo utilizando el protocolo MAVLink a través de una conexión serie. Por tanto, es necesario utilizar alguna de las interfaces serie del dispositivo. La primera interacción entre el controlador de vuelo, *PixRaptor*, en este caso, se realiza utilizando un cable microUSB desde la computadora a bordo. Sin embargo, es posible emplear un puerto de telemetría del controlador y configurar los pines General Purpose Input/Output (GPIO) para establecer comunicación a través del puerto serie 0 del *Raspberry Pi*.

El puerto *TELEM2* del controlador de vuelo debe conectarse a los pines *5V*, *Ground*, *TX* y *RX* como se muestra en la Figura B.1. Además de este proceso, es necesario modificar los siguientes parámetros en el controlador de vuelo:

- `SERIAL2_PROTOCOL = 2`, para habilitar el protocolo MAVLink en el puerto serie 2
- `SERIAL2_BAUD = 57`, para establecer la tasa de baudios en 57600.

Los parámetros se pueden modificar directamente desde MP.

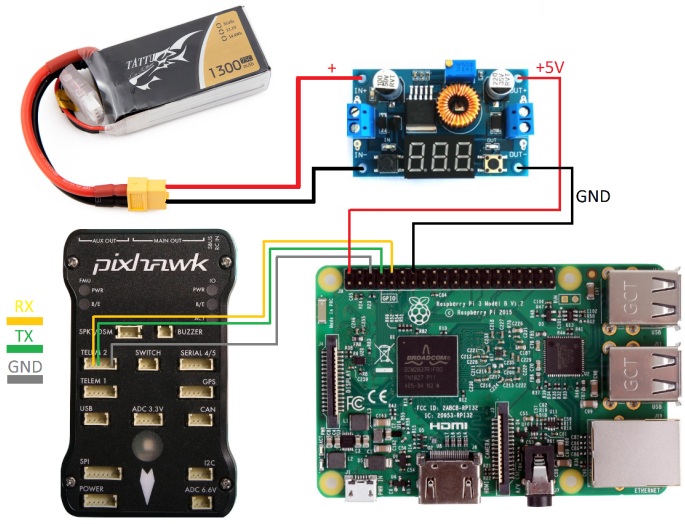


Figura B.1: Conexión serie entre *Raspberry Pi 3B* y *PixHawk*

B.4.1. MavProxy

MAVProxy requiere de algunas dependencias adicionales para su ejecución. Utilice el Código B.38 para instalar *MAVProxy* y los paquetes adicionales.

```
sudo apt-get update
sudo apt-get install python3-opencv python3-wxgtk3.0 libxml2-dev python3-
    matplotlib python3-lxml
sudo pip install future
sudo pip install pymavlink
sudo pip install mavproxy
```

Código B.38: Instalar *MAVProxy*

A continuación, es necesario configurar el puerto serie en el dispositivo *Raspberry Pi* para establecer comunicación con el controlador *PixRaptor* utilizando *MAVProxy*.

B.4.2. Configuración del puerto serie: GPIO

La placa *Raspberry Pi 3B* cuenta con 40 pines GPIO, que pueden configurarse como pines de entrada y salida. Adicionalmente, los pines *GPIO14* y *GPIO15* pueden configurarse para que funcionen como pines *TX* y *RX* de un puerto serie [82].

La placa *Raspberry Pi* asigna el puerto serie Universal Asynchronous Receiver/Transmitter (UART) principal, *ttyAMA0*, al dispositivo *Bluetooth*. El segundo puerto serie denominado *miniUART*, recae sobre el puerto *ttyS0*, siendo este un puerto con menor rendimiento. El puerto `ttyS0` puede ser usado por los pines GPIO. Sin embargo, en versiones de *Raspbian* como *Buster* se encuentra deshabilitado de forma predeterminada. Para habilitarlo, utilice el Código B.39 para acceder al archivo de configuración de arranque del sistema.

```
sudo nano /boot/config.txt
```

Código B.39: Habilitar puerto serie en cabecera *GPIO*

En la parte inferior del archivo `/boot/config.txt` agregue lo expuesto en el Código B.40.

```
enable_uart=1
core_freq = 250
```

Código B.40: Contenido del archivo *config*

La sentencia `core_freq` desbloquea la frecuencia del núcleo de la placa para que pueda ser usada por el puerto. Reinicie el dispositivo para que las configuraciones tomen efecto.

El puerto `ttyS0` se encuentra vinculado al servicio *getty* (consola). Es necesario desactivar este servicio utilizando el Código B.41.

```
sudo systemctl stop serial-getty@ttyS0.service
sudo systemctl disable serial-getty@ttyS0.service
```

Código B.41: Desactivar servicio *getty* de puerto *ttyS0*

Para finalizar con este proceso, debe eliminar la consola del archivo *cmdline*. Utilice el Código B.42 para acceder al archivo de configuración. Elimine la declaración `console=serial0,115200` y guarde los cambios.

```
sudo nano /boot/cmdline.txt
```

Código B.42: Eliminar consola del archivo *cmdline*

Para verificar si la designación ha sido realizada, utilice el comando `ls -l /dev` para revisar los alias asignados a los puertos serie. La Figura B.2 presenta el resultado esperado.

```
lrwxrwxrwx  1 root root           5 ene 28 11:57 serial0 -> ttyS0
lrwxrwxrwx  1 root root           7 ene 28 11:57 serial1 -> ttyAMA0
```

Figura B.2: Alias de los puertos serie en el dispositivo *Raspberry Pi 3B*

Para probar la configuración realizada, conecte los pines del encabezado GPIO al puerto `TELEM2` como en la Figura B.1. Abra una terminal en la *Raspberry Pi* y utilice los comandos del Código B.43 para establecer comunicación con el controlador de vuelo.

```
sudo -s
python3 mavproxy.py --master=/dev/ttyS0 --baudrate 56200 --aircraft
    MyCopter
```

Código B.43: Establecer comunicación con el controlador de vuelo usando *MAVProxy*

```
pi@raspberrypi:~ $ sudo -s
root@raspberrypi:/home/pi# mavproxy.py --master=/dev/ttyS0 --baudrate 57600 --aircraft RaspiDrone
Connect /dev/ttyS0 source_system=255
no script RaspiDrone/mavinit.scr
Log Directory: RaspiDrone/logs/2020-02-13/flight1
Telemetry log: RaspiDrone/logs/2020-02-13/flight1/flight.tlog
Waiting for heartbeat from /dev/ttyS0
 MAV> online system 1
STABILIZE> Mode STABILIZE
fence breach
APM: APM:Copter V3.5.5 (27229c83)
APM: PX4: 0384802e NuttX: 1bcae90b
APM: Frame: QUAD
APM: PX4v2 0047002C 31345117 32363633
APM: PreArm: Compasses inconsistent
APM: PreArm: Throttle below Failsafe
Received 773 parameters
Saved 773 parameters to RaspiDrone/logs/2020-02-13/flight1/mav.parm
APM: PreArm: Compasses inconsistent
```

Figura B.3: Parámetros del piloto automático obtenidos a través de *MavProxy*

Al ejecutar este comando, debe desplegarse información del *firmware* del piloto automático. En este caso, el controlador de vuelo *PixRaptor* muestra la versión de *firmware* de *ArduCopter*, y se descargan los parámetros del controlador. De forma predeterminada se establece el modo de vuelo en estabilizado (*STABILIZE*). La Figura B.3 presenta el resultado obtenido al ejecutar los comandos del Código B.43.

Puede verificar el funcionamiento de *MAVProxy* usando los comandos del Código B.44. Retire las hélices del vehículo antes de realizar cualquier acción.

```
param show ARMING_CHECK
param set ARMING_CHECK 0
arm throttle
```

Código B.44: Activar el acelerador del vehículo usando *MAVProxy*

`ARMING_CHECK` es el parámetro que activa o desactiva la revisión de los requisitos previos al armado del sistema de propulsión del vehículo. Estos requisitos representan un mecanismo de seguridad tanto para el piloto como para el vehículo. La revisión contempla calibración, estado y detección de errores de componentes como GPS, barómetro, batería, brújula, acelerómetros, radio control, *Safety Switch*, sistema, entre otros [83].

Al establecer este parámetro en 0 se elude esta revisión. Por tanto, emplee este comando solo para probar la comunicación de la computadora a bordo con el controlador de vuelo. Antes de efectuar cualquier proceso, retire las hélices para evitar cualquier altercado. `ARM THROTTLE` armará el sistema de propulsión, los motores empezarán a moverse y el palo del acelerador podrá ser usado. En algunos vehículos es necesario presionar el interruptor de seguridad (*Safety Switch*) hasta que tome un color sólido para activar la comunicación con el radio control.

Finalmente, para conectarse con el piloto automático usando *Dronekit* debe ejecutar el *script* de *Python* y definir el dispositivo serie y la tasa de baudios en el argumento `connect` como se presenta en el Código B.45.

```
python script.py --connect '/dev/ttyS0,57600'
```

Código B.45: Establecer conexión entre *Raspberry Pi* y *PixRaptor* a través de *Python*

B.5. ION-DTN

Esta sección presenta el proceso de descarga e instalación de ION-DTN y *pyion*. Además, presenta la estructura del archivo de configuración de inicio para la implementación.

B.5.1. Instalación

La instalación de ION-DTN en *Ubuntu* y *Raspbian* requiere que los paquetes del sistema se encuentren actualizados. Emplee el Código B.46 para este propósito

```
sudo apt-get update
```

Código B.46: Actualizar paquetes existentes

Luego, utilizando el Código B.47 instale el paquete *build essentials*.

```
sudo apt-get install build-essentials -y
```

Código B.47: Instalar *build essentials*

A continuación, descargue el paquete de ION-DTN desde su repositorio y descomprímalo, utilizando el Código B.48.

```
wget https://sourceforge.net/projects/ion-dtn/files/ion-3.7.0.tar.gz/
    download
tar xzvf download
```

Código B.48: Descargar *ION*

Ubíquese dentro de la carpeta de *ion* y clone el paquete de extensión de *Python*, *pyion*, utilice el Código B.49 para lograrlo. La versión de ION-DTN y *pyion* debe ser la misma para evitar errores en su ejecución.

```
git clone --single-branch --branch=v3.7.0 https://github.com/msancheznet/
    pyion.git
```

Código B.49: Descargar *pyion*

El siguiente paso es compilar el código fuente de ION-DTN. Ejecute el Código B.50 para iniciar este proceso, puede tardar algunos minutos

```
./configure
make
```

Código B.50: Compilar *ION*

La compilación producirá un error referido a *memcpy*. Diríjase hacia la línea que genera el error y cambie *memcpy* por *strcpy* [1] Luego, compile otra vez. A continuación, mueva los archivos de ION-DTN hacia los directorios del sistema empleando el Código B.51

```
sudo make install
```

Código B.51: Compilar *ION*

Para finalizar la instalación de ION-DTN, cree los enlaces y la caché a las bibliotecas compartidas de *pyion*, ejecute el Código B.52.

```
sudo ldconfig
```

Código B.52: Crear y actualizar caché

La instalación de *pyion* requiere de los paquetes *autotools* y *automake*. Emplee el Código B.53 para lograrlo.

```
sudo apt-get install autotools-dev automake python3-dev
```

Código B.53: Instalar paquetes de desarrollo

A continuación, ingrese al archivo `bashrc` y agregue al final del mismo la ruta de la librería `LD_LIBRARY_PATH=/usr/local/lib` utilizando el Código B.54.

```
nano ~/.bashrc
```

Código B.54: Abrir el archivo *bashrc*

Luego, exporte la variable `ION_HOME` con el Código B.55

```
export ION_HOME=~/ion-3.7.0/
```

Código B.55: Exportar la variable *HOME*

Finalmente, diríjase a la carpeta de *pyion* y ejecute el *script* de instalación, emplee el Código B.56.

```
sudo -E python3 setup.py install
```

Código B.56: Instalar *pyion*

[1] El uso de *strcpy* evita errores al almacenar el caracter de terminación de cadena. Si bien *memcpy* puede ser más eficiente, requiere definir la longitud de la cadena. Este valor no siempre se conoce[84].

B.5.2. Configuración de nodo

La configuración de ION-DTN se declara previo al inicio del nodo. ION-DTN emplea ocho utilidades que administran la configuración: *bsspadmin*, *dtn2admin*, *ionadmin*, *ionsecadmin*, *bpadmin*, *ipnadmin*, *cfdpadmin* y *ltpadmin*.

Para este proyecto, la configuración de inicio de ION-DTN se detalla en dos archivos: `ioninit.ionconfig` y `host.rc`. El archivo `ioninit` especifica los parámetros de tiempo de ejecución de la implementación, mientras que, `host` establece los comandos necesarios para configurar las utilidades de administración de inicio. Dentro del archivo `ioninit` se debe establecer los siguientes parámetros:

- **`wmkey`:** llave con la que se identificará la memoria compartida.
- **`wmSize`:** tamaño de bloque de la memoria de trabajo.
- **`wmAddress`:** dirección del bloque de memoria volátil.
- **`sdrName`:** cadena de caracteres que identifican la base de datos Simple Data Recorder (SDR).
- **`sdrWmSize`:** tamaño de bloque de memoria reservado para la memoria de trabajo para SDR.
- **`configFlags`:** suma de banderas que caracterizan el comportamiento de SDR: `SDR_in_DRAM (1)`, `SDR_IN_FILE (2)`, `SDR_REVERSIBLE (4)` y `SDR_BOUNDED (8)`.
- **`heapWords`:** número de palabras de almacenamiento que usará la base de datos SDR.
- **`heapKey`:** clave de memoria compartida para ubicar el bloque de memoria.
- **`pathName`:** ruta del directorio donde se ubicará el archivo *heap* para la SDR, y el archivo de *log* para la base de datos.

Puede obtener información detallada de la implementación y su configuración en [23]. Por otro lado, el archivo `host` inicia especificando los comandos de *ionadmin*, se detalla el comando `1`, que requiere de un valor numérico para el identificador de nodo, y la ruta del archivo de parámetros de tiempo de ejecución, como muestra el Código B.57.

```
1 1 nodo_id ruta_archivo_ioninit
```

Código B.57: Inicio de *ionadmin*

A continuación, se declara el inicio de los comandos usando `s`. Luego, se debe definir los contactos y rangos de contacto para el nodo. Los contactos se establecen de acuerdo a la ruta de transferencia y no son bidireccionales. Los rangos establecen el tiempo que los paquetes permanecen en el nodo antes de ser eliminados. El rango cubre ambas direcciones, se debe definir un rango por combinación de nodos. El Código B.58 presenta la sintaxis para estos comandos.

```
1 a contact tiempo_inicio tiempo_final nodo1_id nodo2_id
2 a range tiempo_inicio tiempo_final nodo1_id nodo2_id 1
```

Código B.58: Sintaxis para declaración de contactos y rangos

Para finalizar la sección de *ionadmin* se define un valor en $bytes/s$ esperado de consumo y producción de datos para el nodo, como se muestra en el Código B.59.

```
1 m production valor_produccion
2 m consumption valor_consumo
```

Código B.59: Consumo y producción en un nodos

La siguiente sección en ser declarada es *bpadmin*. Esta sección inicia con el comando `1`. Luego, se define un esquema *ipn*, se especifica los *endpoints* asociados al nodo y la configuración para la cola de recepción: `x` descarta *bundles* si la aplicación no es miembro del *endpoint* o *q* encola los *bundles*. El Código B.60 presenta la sintaxis a seguir.

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:nodo_id.numero_servicio x/q
```

Código B.60: Declaración de *endpoints*

En seguida, se define el protocolo de transporte de la capa de convergencia que será empleado en el nodo y ductos de entrada y salida. Los ductos especifican un protocolo de transporte, una dirección IP y un servicio de la capa de convergencia asociado, como muestra el Código B.61. La sección finaliza con el comando `s`.

```
a protocol protocolo_transporte tamano_cargautil tamano_encabezado
a induct protocolo_transporte:puerto servicio
a outduct protocolo_transporte:puerto servicio
s
```

Código B.61: Declaración de ductos de entrada y salida

Para finalizar, se declara la sección *ipnadmin*. Esta sección define los planes de egreso para los contactos del nodo. Cada nodo debe definir un plan de egreso para si mismo. Un plan de egreso requiere del identificador del nodo receptor, el protocolo de transporte, una dirección IP y un puerto asociado. Si se requiere de enrutamiento estático se puede usar el comando `a group` para asignar una ruta predeterminada para un rango de identificadores. La sintaxis de esta sección se presenta en el Código B.62.

```
a plan id_nodo_rx protocolo_transporte/direccion_IP:puerto
a group id_nodo_inicio id_nodo_final ID_gateway_endpoint
```

Código B.62: Declaración de planes de egreso

El Código B.63 presenta un ejemplo de archivo de configuración `host.rc`.

host.rc

```
## begin ionadmin
1 2 /home/pi/ion-3.7.0/dtn/iondtn.ionconfig

s
a contact +1 +3600 2 1 100000
a contact +1 +3600 1 2 100000
a contact +1 +3600 3 2 100000
a contact +1 +3600 2 3 100000
a range +1 +3600 2 1 1
a range +1 +3600 1 3 1
m production 1000000
m consumption 1000000
## end ionadmin
## begin bpadmin
1
a endpoint ipn:2.0 q
a endpoint ipn:2.1 q
a protocol tcp 1400 100
a induct tcp 192.168.0.1:4556 tcpcli
a outduct tcp 192.168.0.19:4556 tcpclo
a outduct tcp 192.168.0.11:4556 tcpclo
s
## end bpadmin
## begin ipnadmin
a plan 2 tcp/192.168.0.1:4556
a plan 1 tcp/192.168.0.19:4556
a plan 3 tcp/192.168.0.11:4556
a group 1 3 ipn:3.1
## end ipnadmin
```

Código B.63: Contenido del archivo `host.rc`

Para iniciar ION-DTN emplee el Código B.64.

```
1 ionstart -I host.rc
```

Código B.64: Iniciar *ION-DTN*

Solo si el motor de ION-DTN está iniciado en el nodo se puede usar el paquete *pyion*. Para finalizar el servicio DTN emplee el Código B.65.

```
1 killm
```

Código B.65: Finalizar *ION-DTN*

Anexo C

Simulador SITL

En este anexo se presenta el proceso de configuración del simulador SITL. En primer lugar, se expone el proceso de instalación del simulador para *Ubuntu*, y su configuración para un vehículo tipo *Copter* en una placa *Pixhawk1* (Sección C.1). Finalmente, se presenta el procedimiento para conectar el simulador a otras estaciones terrestres (como MP), y leer/escribir en el puerto de telemetría para la ejecución de *scripts* de *Python* usando *Dronekit*.

C.1. Instalación y configuración

Para instalar el simulador SITL en *Ubuntu* siga los siguientes pasos:

1. Actualice los paquetes disponibles en el sistema. Utilice el Código C.1 para esta acción.

```
sudo apt-get update
sudo apt-get upgrade
```

Código C.1: Actualizar paquetes del sistema

2. Clone la carpeta principal de *ArduPilot* desde su repositorio oficial usando el Código C.2.

```
git clone https://github.com/ArduPilot/ardupilot.git
```

Código C.2: Actualizar paquetes del sistema

3. Instale algunos paquetes de *Python* requeridos. Emplee el Código C.3 para efectuar este proceso.

```
sudo apt-get install python-matplotlib python-serial python-wxgtk3.0 python-
    lxml
sudo apt-get install python-scipy python-opencv ccache gawk python-pip
    python-pexpect
```

Código C.3: Instalar paquetes de *Python*

4. Instale *PyMAVLink* y *MAVProxy* usando *pip*. Use el Código C.4 para este propósito.

```
sudo pip install pymavlink mavproxy
```

Código C.4: Instalar *PyMAVLink* y *MAVProxy*

5. Diríjase a la carpeta de *ArduPilot* y actualice los submódulos del proyecto usando el Código C.5. Este proceso puede tardar algunos minutos.

```
cd ardupilot
git submodule update --init --recursive
```

Código C.5: Actualizar submódulos de *ArduPilot*

6. Diríjase a la carpeta *environment install* y ejecute el *script* de instalación de paquetes requeridos acorde al sistema donde esté instalando SITL. El Código C.6 contiene los comandos necesarios para lograr este objetivo. Este proceso tarda varios minutos.

```
cd Tools/environment_install/
./install-prereqs-ubuntu.sh -y
```

Código C.6: Instalar prerrequisitos para *Ubuntu*

7. Vuelva a cargar la ruta del archivo *profile* usando el Código C.7.

```
. ~/.profile
```

Código C.7: Cargar la ruta de archivo *profile*

8. Cierre sesión e iníciela otra vez para que estos cambios se vuelvan permanentes.

A continuación, debe configurar el tipo de vehículo y placa del piloto automático. Existe diversidad de placas de piloto automático que puede configurar en los diferentes tipos de vehículos de *ArduPilot* [85]. En este caso, configuraremos el vehículo *Copter* en la placa *Pixhawk1*. Emplee el Código C.8 para este propósito.

```
cd ardupilot/
./waf configure --board Pixhawk1
./waf copter
```

Código C.8: Configurar *ArduCopter* en placa *Pixhawk1*

Para cargar los parámetros predeterminados del tipo de vehículo, inicie el simulador usando el Código C.9. La opción `w` borra la Electrically Erasable Programmable Read-Only Memory (EEPROM) virtual y carga los parámetros. Esta opción solo debe ser usada una vez. Después de que los parámetros se carguen detenga el simulador usando `Ctrl + C`.

```
cd ArduCopter/
sim_vehicle.py -w
```

Código C.9: Iniciar simulación del vehículo por primera vez

Ahora puede iniciar el simulador con normalidad. Emplee el Código C.10 para iniciar el simulador conjuntamente con la consola y el mapa de ubicación del vehículo. La Figura C.1 presenta el resultado esperado al ejecutar este comando.

```
sim_vehicle.py --console --map
```

Código C.10: Iniciar simulador con mapa y consola

Figura C.1: Simulación de *quadCopter* en SITL

Por defecto, el vehículo aparecerá ubicado junto a la autopista *Monaro* en *Queanbeyan*, Australia. Puede modificar la ubicación inicial del vehículo usando la opción `L` al iniciar el simulador. Debe definir la nueva ubicación en el archivo *locations* dentro de la carpeta de *ArduPilot*. Utilice el Código C.11 para abrir el archivo de ubicaciones.

```
nano ardupilot/Tools/autotest/locations.txt
```

Código C.11: Agregar ubicación inicial para el simulador

Dentro de este archivo debe asignar un nombre a la ubicación acompañado del valor de latitud, longitud, altitud absoluta y rumbo (*heading*). Las coordenadas deben estar expresadas en grados decimales, la altitud y longitud debe estar en metros sobre el nivel del mar y el rumbo en grados con respecto al Norte. Por ejemplo, `BalzayCuenca=-2.890840425,-79.03705134,2607.981261,0`. Finalmente, inicie el simulador empleando el Código C.12.

```
sim_vehicle.py -L BalzayCuenca --console --map
```

Código C.12: Iniciar simulador en una ubicación particular

C.2. Conexión con Dronekit y estaciones terrestres adicionales

Una desventaja de *MAVProxy* es la reducida actualización con la que cuentan sus mapas. Para nuestra región solo cuenta con mapas con varios años de antigüedad. Por lo tanto, si se desea trabajar con mapas con información más actualizada se debe conectar *Mission Planner* a la simulación del vehículo.

Existe dos formas de lograr este propósito. La primera es definir directamente una salida a través del puerto de telemetría al iniciar la simulación del vehículo. El Código C.13 presenta la sintaxis a seguir. En este caso, se emplea como protocolo de transporte a UDP en el puerto 14550. Además, es necesario definir la dirección IP del *host* donde se encuentra la GCS.

```
sim_vehicle.py -L BalzayCuenca --console --out=udp:192.168.0.2:14550
```

Código C.13: Agregar salida de telemetría para *Mission Planner* como opción

De forma similar, se puede agregar una salida desde línea de comandos durante la ejecución del simulador. Espere a que el simulador establezca *STABILIZE* como modo de vuelo por defecto y utilice el Código C.14 para definir la dirección IP y puerto de la salida de la estación terrestre.

```
output add 192.168.0.2:14550
```

Código C.14: Agregar salida de telemetría para *Mission Planner* desde línea de comando

El proceso para conectar *Dronekit* con el simulador es similar al expuesto con anterioridad. En este caso, debe definir como parámetro de la opción `connect` el protocolo de transporte, la dirección IP y el puerto en donde se encuentra ejecutándose SITL. El Código C.15 presenta el comando para la ejecución de un *script* de *Python* conectado a SITL en el mismo equipo.

```
python script.py --connect udp:127.0.0.1:14550
```

Código C.15: Conectar SITL con *Dronekit*

SITL define por defecto dos salidas en el puerto de telemetría establecidas en la dirección de *loopback* del equipo en el puerto 14550 y 14551.

Bibliografía

[1] J. B. Ernst, "Energy-efficient next-generation wireless communications," in *Handbook of Green Information and Communication Systems*. Elsevier Inc., 2013, pp. 371–392.

[2] D. Herring, "Precision Farming," jan 2001. [En línea]. Disponible: https://earthobservatory.nasa.gov/images/1139/precision-farming

[3] A. McBratney, B. Whelan, T. Ancev, y J. Bouma, "Future directions of precision agriculture," in *Precision Agriculture*, vol. 6, num. 1, feb 2005, pp. 7–23.

[4] C. Cruz, A. González, y E. Peñaherrera, "ütilización de datos multiespectrales aeroportados en agricultura de precisión"," *Mapping, ISSN 1131-9100, Nº 105, 2005, pags. 68-77*, 01 2005.

[5] K. Fall, K. L. Scott, S. C. Burleigh, L. Torgerson, A. J. Hooke, H. S. Weiss, R. C. Durst, y V. Cerf, "Delay-Tolerant Networking Architecture."

[6] E. Mahoney, "Disruption Tolerant Networking: Reliable Solar System Internet Connection," 2016. [En línea]. Disponible: https://www.nasa.gov/content/dtn

[7] L. Amondaray y H. R. Sánchez, "Las redes tolerantes al retardo (DTN). Una solución a las comunicaciones rurales en Cuba," *Telématique*, vol. 9, num. 2, pp. 29–42, 2010. [En línea]. Disponible: http://www.redalyc.org/articulo.oa?id=78415900003

[8] U. R. Mogili y B. B. V. L. Deepak, "Review on Application of Drone Systems in Precision Agriculture," *Procedia Computer Science*, vol. 133, pp. 502–509, jan 2018. [En línea]. Disponible: https://www.sciencedirect.com/science/article/pii/S1877050918310081

[9] ArduPilot, "Copter Home — Copter documentation." [En línea]. Disponible: http://ardupilot.org/copter/

[10] ——, "Planning a Mission with Waypoints and Events — Mission Planner documentation." [En línea]. Disponible: http://ardupilot.org/planner/docs/common-planning-a-mission-with-waypoints-and-events.html{#}common-planning-a-mission-with-waypoints-and-events

[11] J. A. Pardo, W. G. Aguilar, y T. Toulkeridis, "Wireless communication system for the transmission of thermal images from a UAV," in *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies, CHILECON 2017 - Proceedings*, vol. 2017-January. Institute of Electrical and Electronics Engineers Inc., dec 2017, pp. 1–5.

[12] D. Ballaria, D. Orellana, E. Acostaa, A. Espinoza, y V. Morocho, "UAV MONITORING FOR ENVIROMENTAL MANAGEMENT IN GALAPAGOS ISLANDS," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLI-B1, pp. 1105–1111, jun 2016.

[13] J. Sani, A. Morillo, y A. Tierra, "VehÍculos aÉreos no tripulados – uav para la elaboraciÓn de cartografÍa escalas grandes referidas al marco de referencia sirgas-ecuador," 06 2015.

[14] M. Becerra, "Discover Ecuador | Ecuador.com | Ecuador Launches Locally Made Drone," 2014. [En línea]. Disponible: https://www.ecuador.com/blog/ecuador-launches-locally-made-drone/

[15] P.-J. Bristeau, F. Callou, D. Vissière, y N. Petit, "The Navigation and Control technology inside the AR.Drone micro UAV," Tech. Rep., 2011. [En línea]. Disponible: http://cas.ensmp.fr/{~}petit/papers/ifac11/PJB.pdf

[16] J. Karpowicz, "Enabling Safe Operation Through All Phases of a Drone Flight," 2016. [En línea]. Disponible: https://www.expouav.com/news/latest/exploring-technology-will-ensure-safe-operation-phases-drone-flight/

[17] J.-M. Dilhac y V. Boitier, "Wireless Sensor Networks," in *Energy Autonomy of Batteryless and Wireless Embedded Systems*. Elsevier, jan 2016, pp. 1–11. [En línea]. Disponible: https://linkinghub.elsevier.com/retrieve/pii/B9781785481239500012

[18] J. Portilla, A. Otero, V. Rosello, J. Valverde, Y. E. Krasteva, E. de la Torre, y T. Riesgo, "Wireless Sensor Networks: From Real World to System Integration - Alternative Hardware Approaches," in *Comprehensive Materials Processing*. Elsevier Ltd, may 2014, vol. 13, pp. 353–373.

[19] S. Fedor y M. Collier, "On the problem of energy efficiency of multi-hop vs one-hop routing in wireless sensor networks," in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, vol. 2, 2007, pp. 380–385.

[20] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, y E. Cayirci, "Wireless sensor networks: A survey," *Computer Networks*, vol. 38, num. 4, pp. 393–422, mar 2002.

[21] A. Vahdat y D. Becker, "Epidemic Routing for Partially-Connected Ad Hoc Networks," Tech. Rep., 2009.

[22] S. Burleigh, "Interplanetary overlay network: An implementation of the dtn bundle protocol," in *2007 4th IEEE Consumer Communications and Networking Conference*, 2007, pp. 222–226.

[23] Jet Propulsion Laboraroty y D. Young, "Interplanetary Overlay Network (ION) - Design and Operation," California, pp. 1–78, 2018. [En línea]. Disponible: https://sourceforge.net/projects/ion-dtn

[24] J. A. Fraire, P. Madoery, S. Burleigh, M. Feldmann, J. Finochietto, A. Charif, N. Zergainoh, y R. Velazco, "Assessing Contact Graph Routing Performance and Reliability in Distributed Satellite Constellations," *Journal of Computer Networks and Communications*, vol. 2017, p. 2830542, 2017. [En línea]. Disponible: https://doi.org/10.1155/2017/2830542

[25] RAE, "dron | Definición | Diccionario de la lengua española | RAE - ASALE." [En línea]. Disponible: https://dle.rae.es/dron?m=form

[26] M. Hassanalian y A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," pp. 99–131, may 2017.

[27] T. I. C. Aviation, "RPAS/3 Symposium." [En línea]. Disponible: https://www.icao.int/Meetings/RPAS3/Pages/default.aspx

[28] A. Restas, "Drone Applications for Supporting Disaster Management," *World Journal of Engineering and Technology*, vol. 03, num. 03, pp. 316–321, 2015.

[29] M. Corcoran, "Drone Journalism: Newsgathering applications of Unmanned Aerial Vehicles (UAVs) in covering conflict, civil unrest and disaster," Tech. Rep.

[30] B. Laszlo, R. Agoston, y Q. Xu, "Conceptual Approach of Measuring the Professional and Economic Effectiveness of Drone Applications Supporting Forest fire Management," in *Procedia Engineering*, vol. 211. Elsevier Ltd, jan 2018, pp. 8–17.

[31] S. Lee y Y. Choi, "Reviews of unmanned aerial vehicle (drone) technology trends and its applications in the mining industry," pp. 197–204, jul 2016.

[32] Dempsey Maritin, *U.S Army Unmanned Aircraft Systems Roadmap 2010-2035*, 2013, vol. 53, num. 9.

[33] Federal Aviation Administration y S. Zaidman, "Global Positioning system wide area augmentation system (waas) performance standard," Federal Aviation Administration, Department of Transportation United States of America, Washington, DC 20591, Tech. Rep., oct 2008. [En línea]. Disponible: https://web.archive.org/web/20170427033332/http://www.gps.gov/technical/ps/2008-WAAS-performance-standard.pdf

[34] B. Hofmann-Wellenhof, H. Lichtenegger, y E. Wasle, *GNSS – Global Navigation Satellite Systems*, 1ra ed. Vienna: Springer-Verlag Wien, 2008. [En línea]. Disponible: https://www.springer.com/gp/book/9783211730126

[35] A. Angrisano, M. Petovello, y G. Pugliano, "Benefits of combined GPS/GLONASS with low-cost MEMS IMUs for vehicular urban navigation," *Sensors*, vol. 12, num. 4, pp. 5134–5158, apr 2012. [En línea]. Disponible: /pmc/articles/PMC3355462/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC3355462/

[36] A. team gps, "Beitian BN-220 GPS / BN-880 GPS y brújula - Documentación del helicóptero." [En línea]. Disponible: https://ardupilot.org/copter/docs/common-beitian-gps.html?highlight=glonass

[37] Ardupilot Team Parameters, "Complete Parameter List — Copter documentation." [En línea]. Disponible: https://ardupilot.org/copter/docs/parameters-Copter-stable-V3.6.1.html

[38] ArduPilot Dev Team, "Documentación ArduPilot ." [En línea]. Disponible: https://ardupilot.org/ardupilot/

[39] ArduPilo Dev Team, "ArduPilot Project: ArduPlane, ArduCopter, ArduRover source." [En línea]. Disponible: https://github.com/ArduPilot/ardupilot

[40] ArduPilot, "Mission Planner Overview — Mission Planner documentation." [En línea]. Disponible: https://ardupilot.org/planner/docs/mission-planner-overview.html

[41] "SiK Telemetry Radio — Copter documentation." [En línea]. Disponible: https://ardupilot.org/copter/docs/common-sik-telemetry-radio.html

[42] MAVLink Developer Team, "Descripción general · Guía para desarrolladores de MAVLink." [En línea]. Disponible: https://mavlink.io/en/about/overview.html

[43] A. Koubaa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, y M. Khalgui, "Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey," *IEEE Access*, vol. 7, num. 8, pp. 87 658–87 680, 2019.

[44] MAVLink Developer Team, "Messages (common) · MAVLink Developer Guide." [En línea]. Disponible: https://mavlink.io/en/messages/common.html

[45] 3D Robotics, "Sobre DroneKit." [En línea]. Disponible: https://dronekit-python.readthedocs.io/en/latest/about/overview.html

[46] A. Tridgell, P. Barker, y S. Dade, "MAVProxy - documentación de MAVProxy 1.6.4." [En línea]. Disponible: http://ardupilot.github.io/MAVProxy/html/index.html

[47] R. Vega Astorga, "Simulation of a Quadrotor Unmanned Aerial Vehicle," Ph.D. dissertation, Carlos III de Madrid, 2016.

[48] OpenCV, "OpenCV: Introduction." [En línea]. Disponible: https://docs.opencv.org/master/d1/dfb/intro.html

[49] R. academia española, "fiducia | Definición | Diccionario de la lengua española | RAE - ASALE." [En línea]. Disponible: https://dle.rae.es/fiducia?m=form

[50] D. B. Dos Santos Cesar, C. Gaudig, M. Fritsche, M. A. Dos Reis, y F. Kirchner, "An evaluation of artificial fiducial markers in underwater environments," sep 2015.

[51] O. S. C. Vision, "OpenCV: Detection of ArUco Markers." [En línea]. Disponible: https://docs.opencv.org/trunk/d5/dae/tutorial{_}aruco{_}detection.html

[52] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, y M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, num. 6, pp. 2280–2292, jun 2014.

[53] B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic Imaging*, vol. 13, num. 1, p. 146, jan 2004. [En línea]. Disponible: http://electronicimaging.spiedigitallibrary.org/article.aspx?doi=10.1117/1.1631315

[54] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, num. 1, pp. 62–66, jan 1979. [En línea]. Disponible: http://ieeexplore.ieee.org/document/4310076/

[55] A. Zormpas, K. Moirogiorgou, K. Kalaitzakis, G. A. Plokamakis, P. Partsinevelos, G. Giakos, y M. Zervakis, "Power Transmission Lines Inspection using Properly Equipped Unmanned Aerial Vehicle (UAV)," in *2018 IEEE International Conference on Imaging Systems and Techniques (IST)*. IEEE, oct 2018, pp. 1–5. [En línea]. Disponible: https://ieeexplore.ieee.org/document/8577142/

[56] Tianqu Zhao y Hong Jiang, "Landing system for AR.Drone 2.0 using onboard camera and ROS," in *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*. IEEE, aug 2016, pp. 1098–1102. [En línea]. Disponible: http://ieeexplore.ieee.org/document/7828941/

[57] V. Sudevan, A. Shukla, y H. Karki, "Vision based autonomous landing of an Unmanned Aerial Vehicle on a stationary target," in *2017 17th International Conference on Control, Automation and Systems (ICCAS)*. IEEE, oct 2017, pp. 362–367. [En línea]. Disponible: http://ieeexplore.ieee.org/document/8204466/

[58] P. Smyczynski, L. Starzec, y G. Granosik, "Autonomous drone control system for object tracking: Flexible system design with implementation example," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE, aug 2017, pp. 734–738. [En línea]. Disponible: http://ieeexplore.ieee.org/document/8046919/

[59] J. Janousek y P. Marcon, "Precision landing options in unmanned aerial vehicles," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*. IEEE, may 2018, pp. 58–60. [En línea]. Disponible: https://ieeexplore.ieee.org/document/8388325/

[60] K. T. Putra, R. O. Wiyagi, y M. Y. Mustar, "Precision Landing System on H-Octocopter Drone Using Complementary Filter," in *2018 International Conference on Audio, Language and Image Processing (ICALIP)*. IEEE, jul 2018, pp. 283–287. [En línea]. Disponible: https://ieeexplore.ieee.org/document/8455553/

[61] M. F. Sani y G. Karimian, "Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors," in *2017 International Conference on Computer and Drone Applications (IConDA)*. IEEE, nov 2017, pp. 102–107. [En línea]. Disponible: http://ieeexplore.ieee.org/document/8270408/

[62] K. Fujii, K. Higuchi, y J. Rekimoto, "Endless flyer: A continuous flying drone with automatic battery replacement," in *Proceedings - IEEE 10th International Conference on Ubiquitous Intelligence and Computing, UIC 2013 and IEEE 10th International Conference on Autonomic and Trusted Computing, ATC 2013*. IEEE, dec 2013, pp. 216–223. [En línea]. Disponible: http://ieeexplore.ieee.org/document/6726212/

[63] C. Giannini, A. A. Shaaban, C. Buratti, y R. Verdone, "Delay Tolerant Networking for smart city through drones," in *2016 International Symposium on Wireless Communication Systems (ISWCS)*. IEEE, sep 2016, pp. 603–607. [En línea]. Disponible: http://ieeexplore.ieee.org/document/7600975/

[64] N. Uchida, N. Kawamura, T. Ishida, y Y. Shibata, "Proposal of Autonomous Flight Wireless Nodes with Delay Tolerant Networks for Disaster Use," in *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, jul 2014, pp. 146–151. [En línea]. Disponible: http://ieeexplore.ieee.org/document/6975455/

[65] C. Caillouet, F. Giroire, y T. Razafindralambo, "Efficient data collection and tracking with flying drones," *Ad Hoc Networks*, vol. 89, pp. 35–46, jun 2019.

[66] C. Wang, F. Ma, J. Yan, D. De, y S. K. Das, "Efficient Aerial Data Collection with UAV in Large-Scale Wireless Sensor Networks," *International Journal of Distributed*

Sensor Networks, vol. 11, num. 11, p. 286080, nov 2015. [En línea]. Disponible: http://journals.sagepub.com/doi/10.1155/2015/286080

[67] J. R. Martinez-De Dios, K. Lferd, A. De San Bernabé, G. Núñez, A. Torres-González, y A. Ollero, "Cooperation between UAS and wireless sensor networks for efficient data collection in large environments," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 70, num. 1-4, pp. 491–508, apr 2013.

[68] J J O'Connor y E F Robertson, "Light through the ages: Ancient Greece to Maxwell," aug 2002. [En línea]. Disponible: http://mathshistory.st-andrews.ac.uk/HistTopics/Light{_}1.html

[69] OpenCV, "Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 documentation." [En línea]. Disponible: https://docs.opencv.org/2.4/modules/calib3d/doc/camera{_}calibration{_}and{_}3d{_}reconstruction.html

[70] Z. Zhang, "A Flexible New Technique for Camera Calibration; a typo in Appendix B) (last updated on Aug A Flexible New Technique for Camera Calibration," Tech. Rep., 2008. [En línea]. Disponible: http://research.microsoft.com/~zhanghttp://research.microsoft.com/~zhang

[71] A. Mordvintsev y K. Abid, "OpenCV-Python Tutorials Documentation Release 1," Tech. Rep., nov 2017. [En línea]. Disponible: https://opencv-python-tutroals.readthedocs.io/{_}/downloads/en/latest/pdf/

[72] P. Kumar, R. Kumar, S. Anand, D. , y V. Prithiviraj, "Quad band signal strength monitoring system using quadcopter and quad phone," *Journal of Green Engineering*, vol. 5, pp. 1–22, 04 2015.

[73] H. Ji, L. Ma, G. Ai, y M. Wang, *The Distributions of HDOP and VDOP in GNSS and a Corresponding New Algorithm of Fast Selecting Satellites*, 01 2012, vol. 160, pp. 411–421.

[74] Ardupilot Buzzer, "Buzzer (aka Tone Alarm) — Copter documentation." [En línea]. Disponible: https://ardupilot.org/copter/docs/common-buzzer.html

[75] FrSky, "Instruction Manual For FrSky RX8R," Tech. Rep. [En línea]. Disponible: https://www.frsky-rc.com/wp-content/uploads/2017/07/Manual/RX8R161216.pdf

[76] Tarot LTD Team, "2-AXIS BRUSHLESS GIMBAL SYSTEM SOFTWARE FOR GOPRO - FLYING MODEL AIRPLANE." [En línea]. Disponible: http://www.tarotrc.com/Download/Detail.aspx?Lang=en{&}Id=cbcd93bb-13c9-4a2a-a38c-4a8d79422033

[77] Ardupilot, "Combinación de GPS (también conocido como GPS dual): documentación del helicóptero." [En línea]. Disponible: https://ardupilot.org/copter/docs/common-gps-blending.html

[78] ——, "Barometer (external) — Copter documentation." [En línea]. Disponible: https://ardupilot.org/copter/docs/common-baro-external.html

[79] Raspbian Org, "FrontPage - Raspbian." [En línea]. Disponible: https://www.raspbian.org/FrontPage

[80] T. Ylonen, SSH Communication Security Corp, y C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," Tech. Rep., 1992. [En línea]. Disponible: https://tools.ietf.org/html/rfc4251

[81] 3D Robotics, "Dronekit / dronekit-sitl: SITL para DroneKit." [En línea]. Disponible: https://github.com/dronekit/dronekit-sitl{#}dronekit-sitl

[82] Raspberry Pi Foundation, "GPIO - Documentación de Raspberry Pi." [En línea]. Disponible: https://www.raspberrypi.org/documentation/usage/gpio/

[83] ArduPilot Dev Team, "Pre-Arm Safety Checks — Copter documentation." [En línea]. Disponible: https://ardupilot.org/copter/docs/common-prearm-safety-checks.html

[84] P. Cordes, "c++ - Is copying in a loop less efficient than memcpy()? - Stack Overflow." [En línea]. Disponible: https://stackoverflow.com/questions/35385157/is-copying-in-a-loop-less-efficient-than-memcpy

[85] ArduPilot, "Building ArduPilot." [En línea]. Disponible: https://github.com/ArduPilot/ardupilot/blob/master/BUILD.md