

Sistema de carga modular y aterrizaje de precisión para enjambre de drones

**José Alberto Castro Villasana, José Eduardo Castro Villasana,
Ana Bárbara Quintero García**

Una Tesis presentada a la Facultad de Ingeniería en Conformidad con los
Requisitos para el Grado de:

Ingeniero en Tecnologías Electrónicas y Robótica, Ingeniero en Mecatrónica



Universidad de Monterrey
Departamento de Ingeniería y Tecnologías
San Pedro Garza García, México

30 de noviembre 2024

Asesor: Fermín Castro Aragón

Contents

1	Introducción	4
1.1	Problem Statement	4
1.1.1	Clear Problem Definition	4
1.1.2	Research Questions	4
1.2	Objectives	4
1.2.1	General Objective	4
1.2.2	Specific Objectives	5
1.3	Justification	5
1.3.1	Project Relevance	5
1.3.2	Potential Impact	5
1.4	Scope and Limitations	6
1.4.1	Project Scope	6
1.4.2	Study Limitations	6
1.5	Thesis Structure	6
2	Estado del Arte	7
2.1	Estado del Arte	7
2.2	Hardware	7
2.2.1	Drones	7
2.2.2	Motors	8
2.2.3	Propellers	9
2.2.4	LiPo Batteries	10
2.2.5	Aplicaciones de los Drones	10
2.2.6	Swarming Drones	11
2.2.7	Types of Drone Charging Stations	13
2.2.8	Types of Mechanisms	16
2.3	Electronics	17
2.3.1	Motors and ESC Modules	17
2.3.2	Flight Controllers	17
2.3.3	Companion Computers and Embedded Systems	18
2.3.4	Communication Protocols	19
2.3.5	Sensors	20
2.3.6	Types of Chargers and Charging	20
2.4	Software	21
2.4.1	FC Firmware	21
2.4.2	Ground Control Systems (GCS)	23
2.4.3	Operative Systems	23

2.4.4	ROS2 Distributions	24
2.4.5	ROS 2 Architecture	26
2.4.6	Computer Vision	28
2.4.7	What is an ArUco?	30
2.4.8	Aruco vs. Embedded Aruco	32
2.4.9	Aruco Detection	33
2.4.10	ROS2-Pixhawk Communication	35
2.4.11	Sensores para la Navegación en Ambientes Exteriores e Interiores	37
2.4.12	Fusión de Sensores usando el Filtro de Kalman en Pixhawk	38
3	Desarrollo	42
3.1	Diseño y Desarrollo de la Base de Carga	42
3.1.1	Especificaciones y Requerimientos	42
3.1.2	Lista de Materiales	42
3.1.3	Selección e Implementación del Mecanismo de Movimiento del Cajón	44
3.1.4	Diseño CAD de la Base de Carga	44
3.1.5	Proceso de Manufactura de la Estructura de la Base de Carga	45
3.1.6	Circuito Electrónico de la Estación de Carga	45
3.2	Instrumentación del Dron	46
3.2.1	Especificaciones y Requerimientos	46
3.2.2	Lista de Materiales para la Instrumentación del Dron	46
3.2.3	Diseños CAD del Dron	47
3.2.4	Circuito de Distribución de Energía	47
3.3	Software Configuration	48
3.3.1	Configuración de la Computadora Central en Tierra	48
3.3.2	Configuración de la Computadora Auxiliar del Dron	49
3.3.3	Configuración de la Estación de Control en Tierra	52
3.4	Sistema de Visión	53
3.4.1	Especificaciones y Requerimientos	53
3.4.2	Calibración de la Cámara	53
3.4.3	Generación de Marcadores ArUco	55
3.4.4	Detección de Marcadores e-Aruco en Tiempo Real	58
3.5	Sistema de Comunicación	59
3.5.1	Especificaciones y Requerimientos	59
3.5.2	Estructura de Comunicación	60
4	Resultados	61
4.1	Resultados del Diseño y Manufactura de la Base de Carga	61
4.1.1	Diseño CAD Final del 1er prototipo de la Estación de Carga	61
4.1.2	Diseño e Instrumentación del Drone	61
4.1.3	Movimiento del Cajón	61
4.1.4	Vuelo del Drone	61
4.1.5	Carga y Descarga de Batería	61
4.1.6	Detección del ArUco	61
4.1.7	Sistema de Comunicación	61
4.2	Ánalisis de Desempeño General	61
4.2.1	Resultados de Integración Global	61

Chapter 1

Introducción

1.1 Problem Statement

1.1.1 Clear Problem Definition

One of the main challenges in achieving continuous and autonomous operation of drone swarms is the lack of modular and autonomous charging stations capable of simultaneously recharging multiple drones. Current market solutions face significant limitations, such as the inability to efficiently manage the simultaneous charging of several drones and ensure precise landing in a shared space without human intervention. These limitations not only reduce drone autonomy but also impact operational efficiency, requiring frequent manual interventions to maneuver drones and taking up additional space due to the lack of modular solutions in existing market offerings.

1.1.2 Research Questions

- How can a modular charging station be developed to enable simultaneous charging of multiple drones?
- What localization and vision technologies can be implemented to ensure precise landing?
- What resources are required to implement a precise landing and charging station for a drone swarm?
- Which charging method is the most efficient and effective for drone recharging in terms of performance and reliability?

1.2 Objectives

1.2.1 General Objective

To design and manufacture a charging station for open-architecture quadcopters, develop a communication system between the drone and the charging station, and instrument the drone with

localization sensors and a vision system to ensure effective and safe integration between the quadcopter and its charging system.

1.2.2 Specific Objectives

- Design a CAD model for the charging station, ensuring compatibility with open-architecture quadcopters.
- Manufacture a charging station module for the drone swarm.
- Adapt the current design of the open-architecture drone to the charging station.
- Equip a quadcopter with an onboard computer, flight controller, vision camera, and localization sensors.
- Develop a communication system between the charging station and the quadcopter for the transfer of relevant interaction data.
- Program a system to control the drone from an onboard computer.
- Develop a computer vision system capable of detecting in real-time the pose of an embedded ArUco marker on the charging station.

1.3 Justification

1.3.1 Project Relevance

The continuous and autonomous operation of drone swarms faces various challenges, including the lack of efficient charging solutions that allow the simultaneous recharging of multiple drones autonomously and without human intervention. While this project does not aim to fully solve this challenge, it lays the foundational groundwork for developing a viable solution. By designing a modular and autonomous charging station with a precise landing system, this project provides a starting point for future research and improvements.

The project is particularly relevant to industries such as logistics, surveillance, and precision agriculture, where autonomous drone operations can generate significant benefits. By presenting a modular solution capable of recharging multiple drones within the same physical space, this work sets the stage for future large-scale implementations, potentially reducing operational costs and improving efficiency [?]. Although the problem is not entirely resolved in this work, the advancements presented here contribute to the development of practical solutions in the future.

1.3.2 Potential Impact

Although the system developed in this project is not a final solution, its implementation has the potential to influence future research and developments in the field of drone autonomy. The precise landing and simultaneous charging technologies explored in this work could be fundamental

in designing more advanced and scalable solutions to meet the operational needs of industries increasingly relying on drones.

The potential impact of this project lies in its ability to inspire research that improves drone swarm efficiency in applications such as parcel delivery, large-area surveillance, and resource optimization in agriculture. While current solutions face limitations, this work provides a solid platform on which significant improvements can be implemented in future stages [?].

1.4 Scope and Limitations

1.4.1 Project Scope

This project establishes the foundations for a modular and autonomous drone recharging solution. It focuses on the design, manufacture, and instrumentation of a charging station capable of managing multiple drones simultaneously, utilizing computer vision and localization sensors. The work will be conducted in a controlled environment and validated under simulated conditions to ensure the solution is technically feasible.

While the project does not address autonomous large-scale charging, it provides a preliminary step towards a more advanced solution. The system will include the design of a modular charging station capable of recharging multiple drones in the same space, with a functional prototype of one charging module being physically developed for a single drone. Testing will be carried out on an open-architecture quadcopter, incorporating design modifications to facilitate charging and accommodate the vision and localization sensors [?].

1.4.2 Study Limitations

This study is limited to the technical validation of the system in a controlled environment with a specific type of drone (open-architecture quadcopter). It does not address scalability to other types of drones or implementation under adverse weather conditions. Additionally, the project does not include prolonged testing or commercial application implementation.

The vision and localization technologies implemented will also be limited to the test conditions, and their accuracy in more complex scenarios will not be evaluated in this work. The objective is to demonstrate the technical feasibility of the key components rather than delivering a fully functional final solution [?].

1.5 Thesis Structure

Chapter 2 will present a detailed review of the state of the art in drone charging systems, covering mechanics, electronics, and programming. Chapter 3 will describe the methodology employed in developing the prototype. Chapter 4 will discuss the results obtained during testing, and Chapter 5 will include conclusions and recommendations for future work.

Chapter 2

Estado del Arte

2.1 Estado del Arte

El propósito de este capítulo es brindar una visión clara y detallada de los trabajos e investigaciones previas en el campo relacionado con la carga de drones. Se presenta una revisión de la literatura, las tecnologías existentes y las limitaciones identificadas que justifican la necesidad de la investigación propuesta.

2.2 Hardware

2.2.1 Drones

There are different categories or names for what can be considered a drone, such as UAV (Unmanned Aerial Vehicle), UAS (Unmanned Aerial Systems), RPAS (Remotely Piloted Aircraft System), or Multicopter (Figure 2.1). A specific definition is given by the official documentation [?], “A multicopter is a mechanically simple aerial vehicle whose motion is controlled by speeding or slowing multiple downward thrusting motor/propeller units.” It is known that each category has distinct functionalities, and therefore, a multicopter is not exactly the same as a UAV. According to the official documentation [?], “A multicopter becomes a UAV or Drone when it is capable of autonomous flight.”

Additionally, according to Piotr et al., ”Drones or Unmanned Aerial Systems (UAV - Unmanned Aerial Vehicle or UAS - Unmanned Aerial Systems) are the aircrafts, which are able to fly without a pilot and passengers on board” [?]. Another definition comes from The Office of the Secretary of Defense of the United States of America: “A powered, aerial vehicle that does not carry a human operator, uses aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or non-lethal payload.”

There are various configurations of drones, each with different capabilities and specific applications. Figure 2.2a shows the Quad Frame, one of the most common and basic configurations. Quadrotor drones, with four motors and propellers, are ideal for applications like photography,



Figure 2.1: Multicopter.

surveillance, and light tasks.

Figure 2.2b presents the Hexa Frame, with six motors and propellers, offering greater stability and payload capacity. These drones are suitable for tasks requiring the transport of heavier equipment, such as advanced sensors or high-resolution cameras.

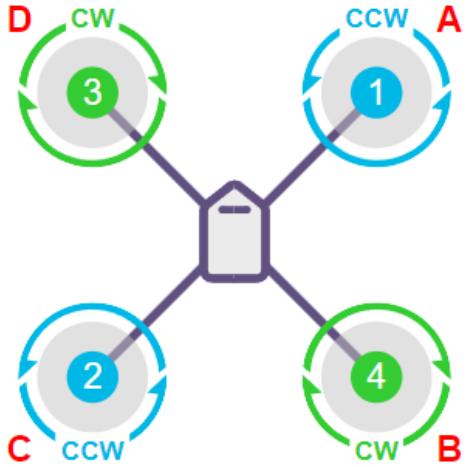
In Figure 2.2c, the Octo Frame configuration is shown, featuring eight motors and propellers. Drones with this configuration are ideal for industrial applications due to their higher payload capacity and redundancy, which improves safety in case of motor or propeller failures.

Finally, Figure 2.2d shows the Octo Quad Frame, combining the advantages of eight motors with the efficiency of a compact configuration. These types of drones are used when high payload capacity and precise stability control are required.

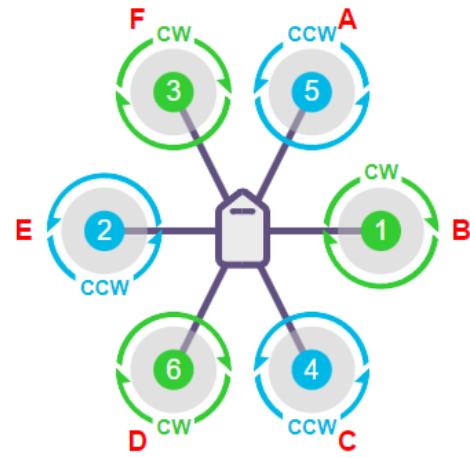
2.2.2 Motors

Motors are an essential component for the flight of a drone, and they are typically brushless motors due to their high efficiency and durability. According to the guide, the selection of the motor depends on factors such as the drone's size, total weight, and the desired type of flight. Key considerations include:

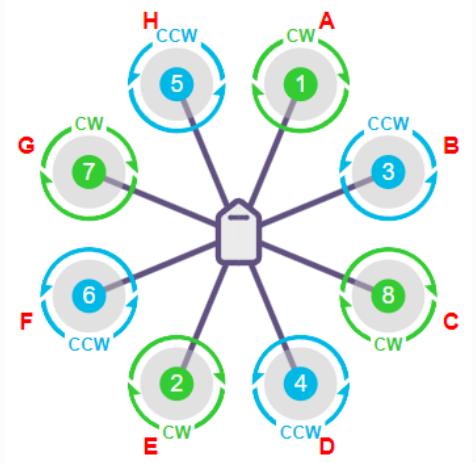
- **Kv Rating:** Indicates the revolutions per minute (RPM) per applied volt without load. A lower Kv provides more torque, making it ideal for larger drones with bigger propellers.
- **Compatibility:** The motor must be compatible with the electronic speed controller (ESC) and the propellers.
- **Mounting:** The drone's frame design should allow for a secure installation of the motors.



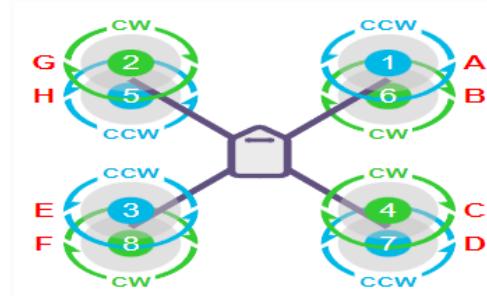
(a) Quad X Frame.



(b) Hexa X Frame.



(c) Octo X Frame.



(d) Octo Quad X Frame.

Figure 2.2: Drone configurations.

2.2.3 Propellers

Propellers generate the necessary lift for flight. According to the guide, they are a critical factor for the drone's performance and should be selected based on the motors and the size of the drone. Important characteristics include:

- **Size (diameter and pitch):** Larger propellers generate more lift but require more powerful motors. The "pitch" determines how much air the propeller displaces per revolution.
- **Material:** Propellers can be made of plastic, wood, or carbon fiber. Carbon fiber propellers are lighter and stronger, making them ideal for high-performance drones.
- **Balancing:** Poorly balanced propellers can cause vibrations, affecting stability and the quality of sensor data.



Figure 2.3: Esta imagen muestra el DJI AGRAS T50 siendo aplicado en la industria agricola. [?]

2.2.4 LiPo Batteries

LiPo (Lithium Polymer) batteries are the primary energy source for most drones. They are lightweight, high-capacity, and capable of delivering the current required by the motors and other components. Key aspects include:

- **Cells (S):** Each cell has a nominal voltage of 3.7V. For example, a 3S battery has 11.1V ($3 \times 3.7V$). The number of cells must be compatible with the motors and ESCs.
- **Capacity (mAh):** Determines the flight time. Higher capacity provides longer flights but also increases weight.
- **Discharge Rate (C):** Indicates how much current the battery can safely provide. It must meet the motors' requirements.
- **Charging and Safety:** LiPo batteries require specific chargers and must be handled carefully to prevent fires.

2.2.5 Aplicaciones de los Drones

Los drones tienen una amplia variedad de aplicaciones en la actualidad y conforme pasa el tiempo se incrementan estas mismas gracias a su versatilidad y capacidad para acceder a áreas de difícil acceso. De acuerdo a la documentación oficial [ardupilot] las aplicaciones de los drones son [1]: Aerial Photography and Videography, First Person View (FPV), Disaster Response, Search and Rescue, Agricultural Applications, Wildfire Mitigation, Hazard Mitigation, 3D Mapping and Geographic Information Systems (GIS), Inspection and Verification, Cargo Delivery, among others.



Figure 2.4: Se muestra un dron de la empresa Amazon, el cual hace entregas de paquetes a domicilio. [?]

2.2.6 Swarming Drones

A diferencia del control individual de un dron, el concepto de enjambre de drones permite la colaboración entre múltiples unidades para ejecutar tareas complejas de manera eficiente y coordinada. A esta tecnología se le llama enjambres de drones que de acuerdo a los autores [4] respresenta “the true revolution in this field is the control of drone swarms, enabling multiple units to collaborate in executing more complex tasks”.

El control de enjambres de drones requiere de sistemas sofisticados que incluyan comunicación bidireccional, tanto entre los drones como con la estación de control terrestre (Ground Control Station, GCS). Según los autores [4], estos sistemas permiten que los drones compartan información clave como posición, estado de batería y obstáculos detectados, lo que posibilita evitar colisiones, mantener formaciones específicas y adaptarse a condiciones dinámicas del entorno.

Además, los autores [4] destacan que un elemento fundamental para el éxito de esta tecnología es la capacidad de posicionamiento y localización precisa, lograda a través de sistemas GPS avanzados. Esta precisión permite a los drones mantener distancias seguras entre sí y operar de forma autónoma en misiones complejas. Otro aspecto relevante es la implementación de sistemas de redundancia que garantizan la continuidad de las operaciones en caso de fallos en alguna unidad del enjambre.

En el ámbito industrial, los enjambres de drones presentan un gran potencial en aplicaciones como inspecciones automatizadas, monitoreo ambiental, agricultura de precisión y gestión de desastres. Según los autores [4], el uso de enjambres de drones en estos escenarios no solo mejora la eficiencia operativa, sino que también abre nuevas oportunidades para el desarrollo de tecnologías autónomas en el contexto de la Industria 4.0.

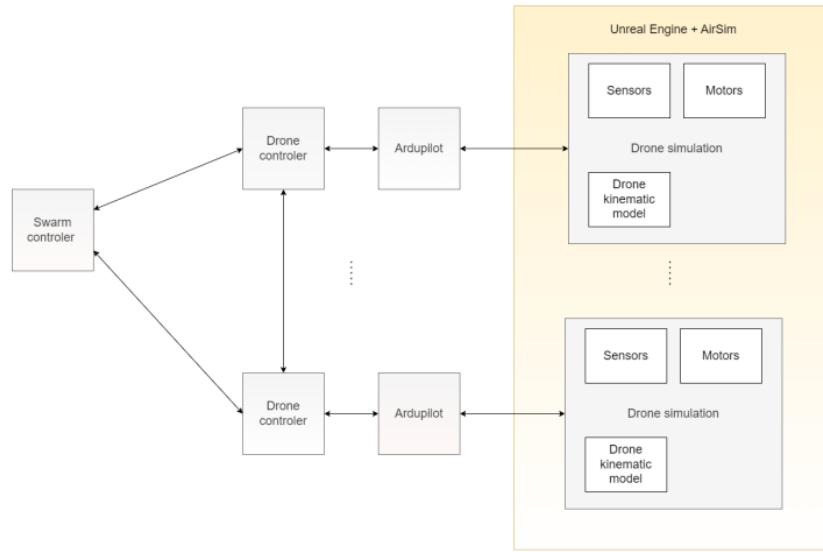


Figure 2.5: Diagrama que simula un modelo de enjambre de drones



Figure 2.6: Se muestra un ejambre de drones en la imagen. [?]

2.2.7 Types of Drone Charging Stations

Currently, there is a growing trend towards the concept of charging or docking stations for drones. According to Grlj et al., "A docking station for UAVs is a multipurpose system that enables them to land safely, take off, recharge and/or replace batteries, and transfer data and payload" [?].

In this context, various types of charging stations can be observed, classified according to the following specifications [?]:

- Mobility (fixed and mobile)
- Charging method (two electrodes, multiple electrodes, wireless, etc.)
- Automatic battery exchange (recharging spare batteries)
- Positioning (active and passive)
- Drone storage (yes or no)
- Package delivery (with storage, without storage)
- Type of landing (precision, vision-based, etc.)
- Type of landing platform (self-leveling)

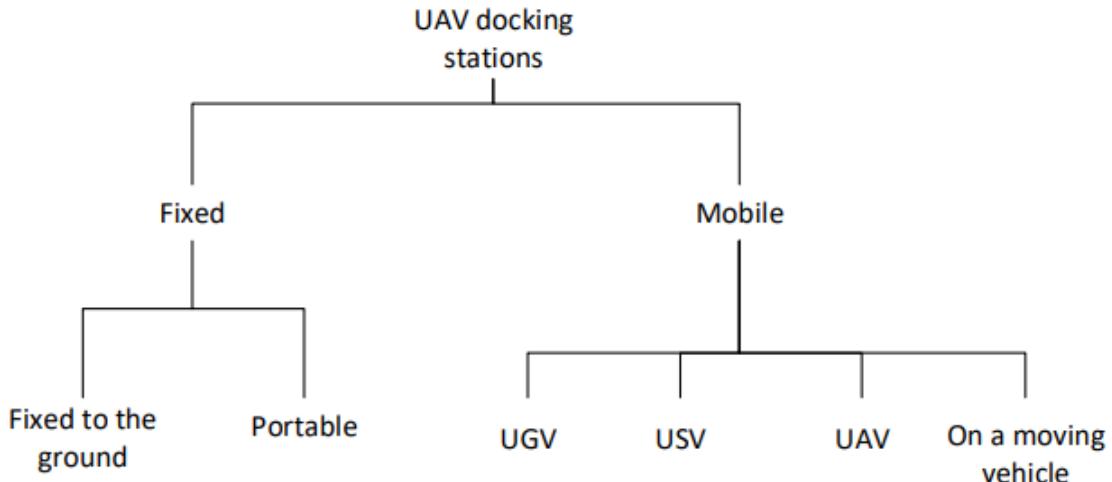


Figure 2.7: Drone Charging Stations Classification [?].

Fixed Charging Stations

The main objective of fixed charging stations is to provide a stable platform that allows drones to recharge without needing to move the infrastructure during the operation process. These stations are often simpler in design and are placed in strategic locations, such as cities or industrial areas.

According to Grlj et al., many of these fixed stations use an active or passive positioning system to ensure the drone lands accurately, and they may feature charging options through electrical contacts or even wireless charging [?].

Advantages of Fixed Charging Stations

Fixed stations are particularly useful in situations where the drone needs a precise landing in a constant location. These stations can use visual markers, such as ArUco [2] markers, to guide the drone during the final landing phase, ensuring high accuracy in the maneuver. Additionally, they may include self-leveling platform systems to guarantee the stability of the drone upon landing, especially on uneven terrain.

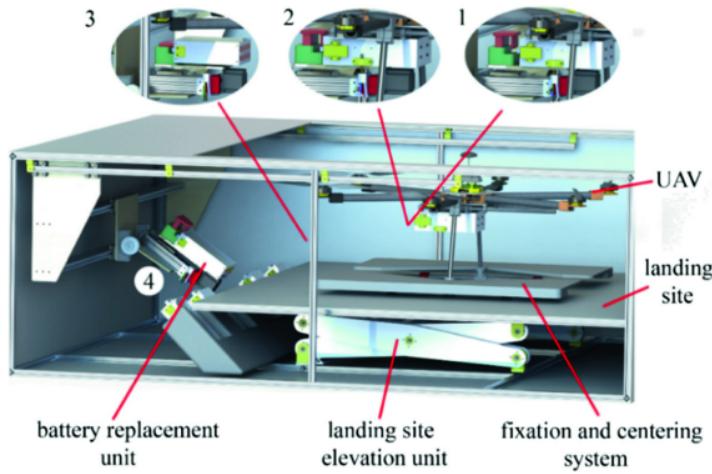


Figure 2.8: Fixed docking station with its subsystems. The labels (1–4) show the battery replacement mechanism in operation [?].



Figure 2.9: Battery replacement docking station [?].

Mobile Charging Stations

Mobile charging stations, on the other hand, provide greater operational flexibility, as they are mounted on surface vehicles, such as Unmanned Ground Vehicles (UGVs) or even Unmanned

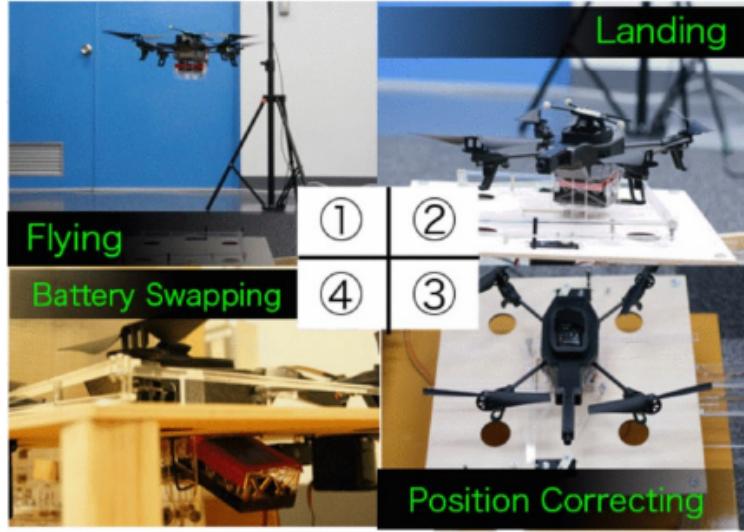


Figure 2.10: Endless Flyer [?], docking station with battery replacement mechanism. The labels in the center show the different landing stages and the order in which they are realized.

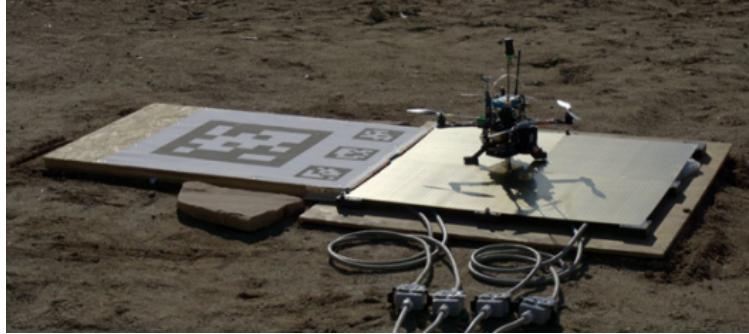


Figure 2.11: Docking station with recharging pad with the custom made UAV from [?]. In the figure, the QR codes used in the approach and landing phase can be seen .

Surface Vehicles (USVs). These stations extend the range of drones by moving to locations that require real-time logistical support, such as in search and rescue operations [?].

Advantages of Mobile Charging Stations

Mobile stations provide a dynamic alternative for supporting drones in missions that require continuous movement. For example, some stations are mounted on autonomous ground vehicles that can move to locations where the drone requires assistance, significantly increasing the operational range of the UAV. These types of stations can also incorporate complex systems to exchange batteries, such as robotic arms that ensure the safe removal and insertion of the drone's battery without needing to pause the mission for long recharge periods.



Figure 2.12: Moving docking station with storage system and battery replacement system. Labels in the figure (1–4) show the order in which the active positioning mechanism and storing mechanism take place [?].

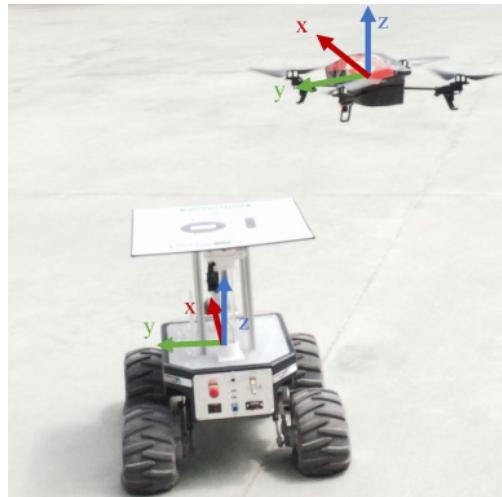


Figure 2.13: UGV–UAV proposed moving docking station solution [?].



Figure 2.14: USV–UAV landing system [?].

2.2.8 Types of Mechanisms

Automatic Drawer Mechanism

The automatic drawer mechanism in the charging station for drone storage is similar to the mechanisms found in other automatic storage solutions. Here, we'll discuss the main mechanisms that could be used in this context, including the advantages and disadvantages of each. The mechanism

chosen for this project was the belt and V-pulley due to its easy accessibility and low cost.

Pistons Pistons are a common choice for automated movement in many types of machinery. They use compressed air or hydraulic fluid to extend and retract, making them well-suited for precise and strong movements. For an automatic drawer for drones, pistons could provide a robust and stable means to open and close the drawer. However, this approach may have higher costs and complexity compared to other mechanisms, as it requires pneumatic or hydraulic infrastructure and regular maintenance.

Belt and Toothed Pulley This mechanism uses a toothed belt and pulley to move the drawer. It offers greater precision compared to V-belts because the toothed design prevents slipping, making it a suitable choice for environments where precise positioning is critical. However, the increased complexity and cost of toothed belts make them less accessible, and they may require more maintenance due to the increased friction between the teeth.

Belt and V-Pulley This mechanism involves a belt shaped like a V and a pulley, which ensures good friction and prevents slipping while moving the drawer. The belt and V-pulley mechanism is simple, inexpensive, and easy to source, which made it the choice for this project. It offers a balance between precision and accessibility, being a practical solution for the needs of an automatic drone storage drawer. Its main disadvantage is that it may lack the precise positioning capabilities of toothed pulleys, but for the current application, this is acceptable.

2.3 Electronics

2.3.1 Motors and ESC Modules

In drones, brushless motors are essential components for flight due to their high efficiency and durability. These motors require Electronic Speed Controller (ESC) modules to precisely control their speed and direction of rotation. ESCs convert the direct current (DC) from the battery into the alternating current (AC) needed by the motors, adjusting the frequency and voltage of the signal to control their rotation speed.

The operation of ESCs depends on frequency modulation technology, where the rate of change determines the motor's speed. As noted by Rohde and Schwarz, “modern ESCs not only regulate speed but also monitor temperature and voltage, protecting the motor from overloads” [?]. In the context of a modular drone charging system, ESCs enable agile and safe control of the propulsion system, which is critical for precise landing and docking maneuvers at the charging station.

2.3.2 Flight Controllers

Flight controllers are central components that manage stability, navigation, and the execution of flight commands in drones. Several models are available in the market, each with specific

characteristics and recommended applications.

Key Flight Controllers

- **DJI Naza:** Widely used in commercial drones due to its ease of configuration and enhanced stabilization system. However, it has limitations in customization and flexibility for advanced research projects.
- **APM (ArduPilot Mega):** Developed by ArduPilot, this platform is highly customizable and suitable for open-source projects. Nonetheless, it faces processing power limitations, restricting its use in complex applications.
- **Holybro Kakute:** This controller offers good sensor integration and is ideal for small racing drones but lacks the advanced processing capabilities needed for autonomous and heavy-load applications.

Pixhawk 6X

Among the available controllers, the **Pixhawk 6X** stands out as the most advanced and robust option. According to the Pixhawk development team, “the Pixhawk 6X was designed to provide high processing capabilities with maximum compatibility for advanced sensors and communication modules” [?]. The main advantages of the Pixhawk 6X include:

- **Processing and Stability:** Equipped with high-performance processors that support advanced stabilization and flight control algorithms.
- **Modularity:** Allows seamless integration with companion computers and external systems, essential for autonomous applications.
- **Integrated Sensors:** Includes high-precision accelerometers and gyroscopes, along with support for external sensors such as magnetometers and GPS.
- **Compatibility:** Highly compatible with communication systems like UART, I2C, and CAN, facilitating its integration into complex systems such as autonomous charging stations.

2.3.3 Companion Computers and Embedded Systems

In advanced drone applications, flight controllers often require the support of additional embedded systems, known as companion computers, to handle intensive data processing, artificial intelligence algorithms, or real-time video transmission.

Main Companion Computers

- **Raspberry Pi 4 and 5:** Both models are cost-effective and powerful options. Their 64-bit processors and 4 GB (or more) of RAM allow them to run complex operating systems

and handle intensive tasks, such as image processing and pattern recognition algorithms. According to Raspbian, the Raspberry Pi 5 offers a 30% performance improvement over its predecessor, making it ideal for high-performance drones at low cost [?].

- **Odroid XU4:** Featuring an Exynos 5422 octa-core processor, this option provides high performance for real-time applications. However, its higher energy consumption limits its use in lightweight drone applications.
- **Jetson Nano:** Developed by NVIDIA, it includes an integrated GPU that excels in artificial intelligence and computer vision tasks. Despite its outstanding graphical processing capabilities, its cost and integration complexity may outweigh its benefits in cost-sensitive projects.
- **Arduino:** In the context of a charging station, Arduino is ideal for handling specific low-processing tasks, such as controlling positioning motors. Its simplicity and low power consumption make it a perfect complement to a Raspberry Pi managing the central system.

2.3.4 Communication Protocols

Effective communication protocols are essential for connecting the various systems and modules in a modular charging station, enabling seamless interaction between the Raspberry Pi, Arduino, and Pixhawk flight controller.

Main Communication Protocols

- **UART (Universal Asynchronous Receiver-Transmitter):** Ideal for simple, low-cost connections, UART enables point-to-point asynchronous data transmission, useful for basic communication between the Raspberry Pi and Arduino [?].
- **SPI (Serial Peripheral Interface):** This protocol supports high-speed synchronous communication and is suitable for applications requiring fast and simultaneous data transfers, such as between the Raspberry Pi and Pixhawk [?].
- **I2C (Inter-Integrated Circuit):** With only two lines (SDA and SCL), I2C allows the connection of multiple devices, making it ideal for integrating sensors and additional peripherals in the charging system [?].

Challenges and Solutions

Latency, noise, and data integrity are common issues in serial communication, especially in drone applications where reliability is critical. For UART, faster baud rates and optimized software help address latency, while shielding and error-checking protocols improve data integrity. SPI mitigates noise through robust grounding and low-skew clock drivers, while I2C uses pull-up resistors and error-detection mechanisms to maintain reliable communication [?].

2.3.5 Sensors

Sensors are devices that detect and measure physical properties such as temperature, pressure, motion, or light, converting them into signals that can be interpreted by a system. They are crucial in gathering environmental data and enabling drones to respond to changes or perform specific functions.

Types of Sensors

- **Mechanical Sensors:** Rely on physical movement to detect changes and generate electrical signals.
- **Motion Sensors:** Detect movement through parameters like infrared radiation or sound waves.
- **Proximity Sensors:** Sense the presence of nearby objects without physical contact, using technologies like ultrasonic or magnetic fields.
- **Optical Sensors:** Detect changes in light intensity, wavelength, or polarization for measuring events or properties [?, ?, ?].

2.3.6 Types of Chargers and Charging

Technologies

For fast-charging technologies, the importance of amperage and voltage is the main output of the charger. The current or amperage is the amount of electricity flowing through the plug to the battery. The voltage is the strength of the electric current. In most cases, the fast charging cables or technologies change the voltage rather than the current to increase the amount of potential energy. For a long time, phones and other small portable devices were being charged by 5V/2.4A, but now, with the introduction of USB-C cables, these devices are being charged to up to 100W (20V/5A). Laptops, which are larger portable devices, can handle more power, hence faster charging and more amperage and voltage, than smaller devices. [?]

Smart charging systems are new technologies designed to optimize the charging process for devices, including electric vehicles, drones, and other rechargeable batteries. These technologies incorporate algorithms, communication, and remote interactions to efficiently manage energy usage during charging. In battery swapping scenarios, vehicles or devices can replace their empty or low batteries with a fully recharged battery that was pre-charged and ready for immediate use. In this case, there would always be a backup battery charging while the first battery is in use in the device. The backup battery would then replace the first battery, which would then be placed back to charge as the backup battery is in use. This minimizes downtime and ensures continuous operation, especially in high-demand environments where quick turnaround is essential. [?]

Wired vs. Wireless Charging

When comparing the wired vs. wireless types of charging in any rechargeable battery, it is essential to compare which method is more effective against which is more practical in everyday use. Wired charging is where a plug is directly in contact with the battery and then connected to a power adapter. The device draws power through the cable to charge its battery.

Wireless charging, as its name suggests, is where there are no cables involved in the charging process. In this case, power is transmitted via magnetic fields between two coils (one in the device and one in the charger) through a technology named inductive charging or resonant charging, where resonant frequencies are more efficient and can reach a longer distance.

When comparing both technologies, wireless is more convenient, eliminating the need for plugging and unplugging cables, but it can be slower and less efficient. [?]

Comparative Analysis

Wired charging is better for faster charging, efficient power transfer since there is less energy loss, more reliable since it does not depend on a specific position or distance, and more economic. However, dealing with cables can be difficult to deal with when they get tangled, lost or worn out, this technology is also inconvenient in a situation where it is frequently plugged or unplugged, and when charging, the device has limited mobility due to the need to stay connected to power through the cable.

Wired charging is more convenient since it was no need for cables which means it has reduced wear and tear, and its charging space is less cluttered due to the no cables. However, wireless charging tends to be slower, less efficient, requires precise alignment, generates more heat, and often comes at a higher cost compared to wired charging. [?]

2.4 Software

In this section, we will discuss all software-related topics researched for the development of this project. Topics include flight controller firmware, ground control stations, companion computer operating systems, ROS 2 documentation, computer vision, ArUco markers, and others.

2.4.1 FC Firmware

The definition of firmware, according to ..., is []. For flight controllers, firmware is the software that runs on the flight controller and is responsible for managing the sensors, actuators, and control algorithms required to maintain the stability and control of the vehicle. Several flight controller firmware options are available today, each with its unique characteristics and advantages. In this project, two of the most popular and widely used firmware in the drone and unmanned vehicle industry were investigated: PX4 and ArduPilot.

PX4 Firmware

PX4 firmware has been a fundamental component in the development of flight control systems for drones and unmanned vehicles, standing out for its open-source nature and flexibility for many applications. According to the official documentation, “*PX4 is a powerful open source autopilot flight stack running on the NuttX RTOS*” [1]. This system is widely recognized for its modular architecture, which allows the integration of new sensors, actuators, and control algorithms, enabling specific adaptations to meet the requirements of different projects [1].

Among its most notable features is its ability to support a wide variety of vehicle types. In this regard, PX4 “*supports many different vehicle frames/types, including: multicopters, fixed-wing aircraft (planes), VTOLs (hybrid multicopter/fixed-wing), ground vehicles, and underwater vehicles*” [1]. This multi-configuration capability is essential, as it allows the same framework to be applied to various platforms with minimal modifications.

Additionally, PX4 is an integral part of a broader ecosystem that includes ground control stations such as QGroundControl, Mission Planner, and specific hardware like Pixhawk. The documentation mentions that “*PX4 is a core part of a broader drone platform that includes the QGroundControl ground station, Pixhawk hardware, and MAVSDK for integration with companion computers, cameras, and other hardware using the MAVLink protocol*” [1]. This integration ensures seamless communication between the different system components, which is crucial for real-time monitoring and mission planning.

Finally, PX4 offers “*flexible and powerful flight modes and safety features*”, which are vital for projects requiring complex maneuvers, such as precise autonomous landing on charging stations [1]. These advanced control functionalities and its ability to integrate with computer vision technologies and external sensors establish PX4 as a key tool in the development of high-relevance autonomous vehicle technologies.

ArduPilot Firmware

ArduPilot, like PX4, is open-source software that runs on a wide variety of hardware, allowing the creation and use of autonomous systems for unmanned vehicles in different applications. According to the official documentation, “*ArduPilot provides a comprehensive suite of tools suitable for almost any vehicle and application. As an open source project, it is constantly evolving based on rapid feedback from a large community of users*” [2]. This flexibility and adaptability have made ArduPilot an essential tool for automation and robotics projects seeking a versatile and robust solution.

One of the most notable aspects of ArduPilot is that, unlike PX4, it does not manufacture hardware. Instead, “*ArduPilot firmware works on a wide variety of different hardware to control unmanned vehicles of all types*” [2]. This means it can integrate with different types of controllers, sensors, and devices, transforming virtually any mobile machine into an autonomous vehicle with the simple addition of an appropriate hardware package.

As mentioned earlier, firmware is the code that runs on the controller, and the choice of firmware depends on the vehicle and mission. As stated in the documentation, “*You choose the firmware to match your vehicle and mission: Copter, Plane, Rover, Sub, or Antenna Tracker*” [2]. This versatility allows developers to select the configuration most appropriate for their specific needs, optimizing the development and operation process.

ArduPilot is also complemented by ground control stations (GCS), which function as the interface between the user and the vehicle’s controller. One of the most comprehensive tools in this area is Mission Planner, described as “*a full-featured GCS supported by ArduPilot*”, offering easy and fast interaction with the firmware [2].

Lastly, it is worth mentioning that both PX4 and ArduPilot share similar functionalities such as sensor calibration, mission planning, parameter configuration, among others. However, the choice between the two will depend on the specific needs of the project and the compatibility with the available **hardware**.

2.4.2 Ground Control Systems (GCS)

Ground control systems (GCS) are auxiliary tools widely used for the configuration, calibration, and monitoring of drones. Although they are not the main focus of this project, they play an important role in system preparation, facilitating parameter configuration, sensor calibration, and supervision during test flights.

Among the tools used are QGroundControl and Mission Planner, which stand out for their compatibility with PX4 and ArduPilot firmware, respectively. These platforms provide intuitive interfaces for adjusting vehicle parameters, planning missions, and visualizing real-time telemetry data. In the context of this project, they have been key to ensuring that the drone is properly configured before being integrated with the modular charging station.

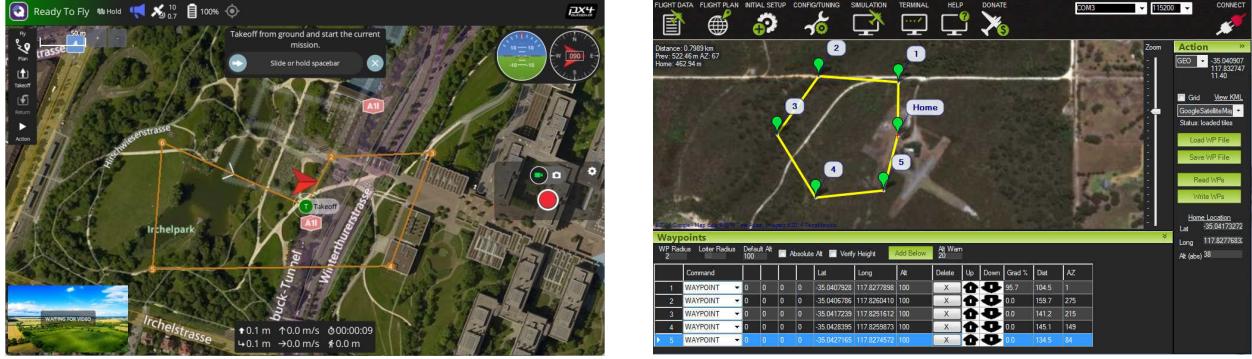
Despite their usefulness during the preparation and testing stages, the proposed system does not directly rely on these tools for its final operation. The autonomous operation of the drone and the charging station is based on direct interaction between the firmware, vision systems, and flight controllers, without requiring continuous intervention from a ground control station.

2.4.3 Operative Systems

To operate a companion computer, it is necessary to choose from the many available operating systems, with Ubuntu and Raspberry Pi OS being among the most popular options. Below, the features and advantages of each operating system are described.

Ubuntu

Ubuntu is an open-source operating system based on Linux, widely recognized for its stability and large support community. It is important to note that within Ubuntu and its versions, even-numbered releases are more stable. According to Ubuntu’s official documentation, this system is



(a) QGroundControl Interface.

(b) Mission Planner Interface.

Figure 2.15: Ground Control Systems used during configuration and testing stages.

“designed for security, reliability, and ease of use” [4]. In the context of companion computers for drones and other autonomous vehicles, Ubuntu is frequently used due to its compatibility with robotics tools such as ROS (Robot Operating System), facilitating the integration and development of advanced software for control and automation.

Ubuntu supports ARM architectures, allowing its installation and operation on devices such as Raspberry Pi 4 and 5. This capability is essential for projects that require efficient local processing, sensor data handling, and real-time communication. Additionally, Ubuntu’s flexibility allows for its environment to be customized to meet the specific needs of the project, whether for running flight control nodes or real-time image processing [4].

Raspberry Pi OS

Raspberry Pi OS is the official operating system developed and optimized for Raspberry Pi devices. The Raspberry Pi OS documentation describes it as *“a Debian-based operating system specifically tuned for the Raspberry Pi hardware”* [5]. Its main advantage is its optimization for Raspberry Pi hardware, ensuring optimal performance and efficient use of available resources.

Raspbian includes a series of pre-installed tools that facilitate development and prototyping, making it a preferred option for educational and research projects. Its compatibility with Python and other programming libraries makes it easier to implement scripts and software necessary for many robotics applications.

Compared to other operating systems, Raspbian is lightweight and allows for a **quick boot**, which is beneficial in scenarios where the system needs to start quickly.

2.4.4 ROS2 Distributions

The ROS 2 (Robot Operating System 2) distributions provide standardized environments for the development of robotic applications, each tailored to different needs and hardware capabilities. One of the advantages of ROS 2 is its ability to **facilitate cross-distribution communication** through the use of topics, allowing nodes in different ROS 2 versions to communicate and collaborate

effectively within the same project [6]. The following sections provide details about the most recent and stable ROS 2 distributions.

Humble Distribution

The **ROS 2 Humble Hawksbill** distribution is known for being a version with **long-term support (LTS)**, which guarantees consistent and reliable updates. This distribution is designed for projects requiring stability and high compatibility with different systems and packages. It is ideal for hardware such as the **Raspberry Pi 4** and is compatible with **Ubuntu 22.04** and earlier versions like Ubuntu 20.04, making it accessible for more standard hardware configurations.

According to the official ROS 2 documentation, Humble is one of the most recommended versions for projects seeking long-term consistency due to its focus on avoiding disruptive changes [?].

Key Features:

- **LTS (Long-Term Support):** Ensures extended support for updates and bug fixes.
- **Stability:** Proven improvements in node communication, ensuring reliability.
- **Broad compatibility:** Works with most libraries and packages within the ROS 2 ecosystem.
- **Optimized for ARM:** Ideal for platforms like the Raspberry Pi 4, efficiently utilizing limited resources.

Overall, Humble is the preferred choice for those prioritizing stability in research projects or long-term applications.

Jazzy Distribution

The **ROS 2 Jazzy Jalisco** distribution, newer than Humble, is designed to take advantage of modern hardware like the **Raspberry Pi 5** and is compatible with advanced operating systems such as **Ubuntu 24.04**. This distribution includes significant performance improvements and new functionalities but is considered more experimental compared to LTS versions.

According to Jazzy's documentation, its primary focus is to facilitate the development of robotic applications that leverage the latest tools and simulators while improving the speed of node-to-node communication [?].

Key Features:

- **New functionalities:** Introduction of experimental features and performance adjustments.
- **Lower latency:** Faster communication between nodes, crucial for real-time applications.
- **Advanced integration:** Improvements in simulation and debugging, leveraging tools like Gazebo and Rviz.

- **Hardware-oriented design:** Designed for devices such as the Raspberry Pi 5, offering greater processing capacity.

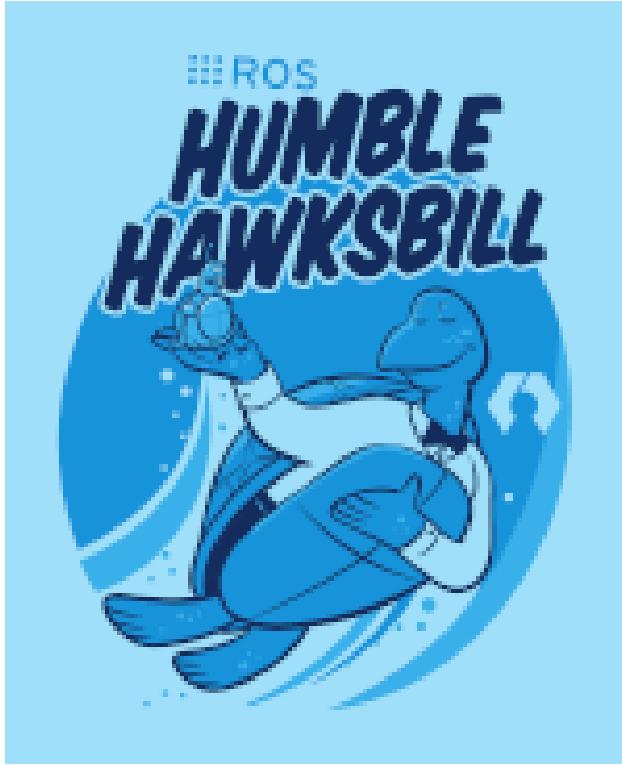


Figure 2.16: ROS 2 Humble

2.4.5 ROS 2 Architecture

ROS 2 (Robot Operating System 2) is a set of libraries and tools designed for developing robotics applications. It is the evolution of ROS 1, created to address issues related to scalability, security, and support for real-time systems. ROS 2 uses an architecture based on the middleware **DDS (Data Distribution Service)**, which facilitates communication between nodes in distributed systems, ensuring interoperability and low response times [6].

Basic Concepts of ROS 2

ROS 2 is organized around several fundamental elements that enable interaction between different parts of the robotic system. These elements, which define its modular architecture, are essential for understanding how a system based on ROS 2 is structured and functions [6]:

- **Nodes:** These are the basic execution units in ROS 2. Each node performs a specific function, such as collecting data from a sensor or controlling an actuator. Nodes are designed to be independent and communicate with each other through topics, services, or actions.



Figure 2.17: ROS 2 Jazzy Jalisco

- **Messages:** These are data structures sent between nodes to share information. Messages have a defined format that ensures all nodes understand the content.
- **Topics:** These represent communication channels for exchanging messages between nodes. Communication through topics is asynchronous, allowing nodes to publish and receive information without waiting for immediate responses.
- **Services:** These enable synchronous communication between nodes. Unlike topics, services operate on a request-and-response model, useful for specific operations.
- **Actions:** Similar to services, but designed for long-duration operations. They allow nodes to receive updates on the progress of a task and cancel it if necessary.
- **Launch Files:** These simplify the configuration and simultaneous startup of multiple nodes, facilitating the management of complex applications, especially in distributed environments.
- **Parameters:** Configurable values that nodes use to adjust their behavior without modifying the code.

The combination of these elements allows robotic systems designed in ROS 2 to be highly scalable and flexible, adapting to both small projects and complex systems.

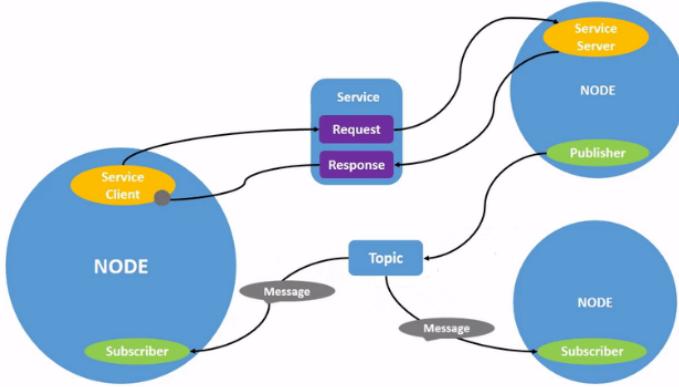


Figure 2.18: ROS 2 Architecture [6].

Key Benefits of ROS 2

The architecture of ROS 2 brings several advantages that make it ideal for modern distributed robotic systems. Among the key benefits are the following [6]:

Table 2.1: Key Benefits of ROS 2

Benefit	Description
Multithreading	Allows nodes to run in parallel, optimizing performance.
Use of topics	Asynchronous communication between nodes for data exchange.
Nodes and services	Nodes interact through services and topics for specific tasks.
Launch files	Configure and execute multiple nodes simultaneously.
Fast communication	Based on DDS, provides low-latency communication between nodes.
Custom messages	Enables defining and using specific structures for each application.
Real-time support	Enables critical applications where response time is essential.

2.4.6 Computer Vision

Computer vision is a programming field that allows systems to interpret and process visual information from the environment. This field is key in applications requiring real-time image and video analysis, such as autonomous robot navigation, object detection, and artificial intelligence systems. To facilitate the development of these applications, specialized tools and libraries such as OpenCV play a fundamental role in the practical implementation of computer vision [7].

OpenCV Library

OpenCV (Open Source Computer Vision Library) is one of the most widely used libraries in industry and academia for developing computer vision solutions. According to its official documentation, OpenCV is “*an open-source computer vision and machine learning software library containing more than 2500 optimized algorithms*” [7]. These tools enable tasks such as image processing, object recognition, and motion tracking, and are compatible with programming languages like Python and C++.

Thanks to its flexibility and ease of use, OpenCV is suitable for both beginners and experts. Moreover, its optimized algorithms enable real-time processes, making it an ideal tool for robotics projects and autonomous applications [7].

Camera Calibration

Before implementing any computer vision system in applications that demand spatial accuracy, it is essential to perform proper camera calibration. This process corrects inherent lens distortions and enables precise measurements of the environment. According to OpenCV documentation, “*Camera calibration is the process of estimating the parameters of the lens and the image sensor of an imaging device*” [?].

Calibration involves determining intrinsic parameters (such as focal length and principal point) and extrinsic parameters (relative to the camera’s position and orientation) that map 2D coordinates from captured images to real-world 3D coordinates.

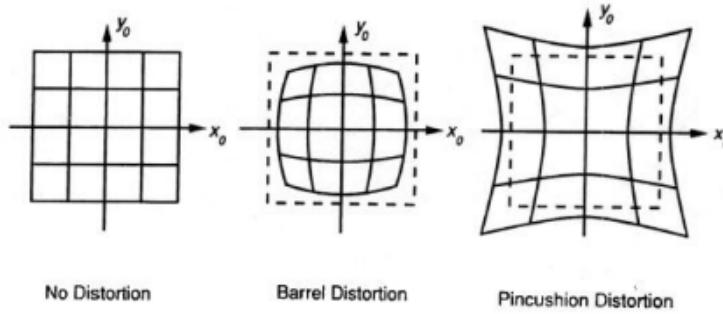


Figure 2.19: Types of camera distortions.

OpenCV provides functions that allow this process to be performed efficiently by detecting patterns in images, such as chessboards or circle grids. The typical calibration workflow includes:

- Capturing images of a known pattern from different angles.
- Identifying points of interest in the captured images (pattern corners).
- Using optimization algorithms to calculate intrinsic parameters and distortion coefficients.

The result of calibration corrects distortions in images and videos, improving the accuracy of computer vision applications. This is especially useful in projects requiring precise spatial analysis, such as autonomous navigation and real-time object positioning.

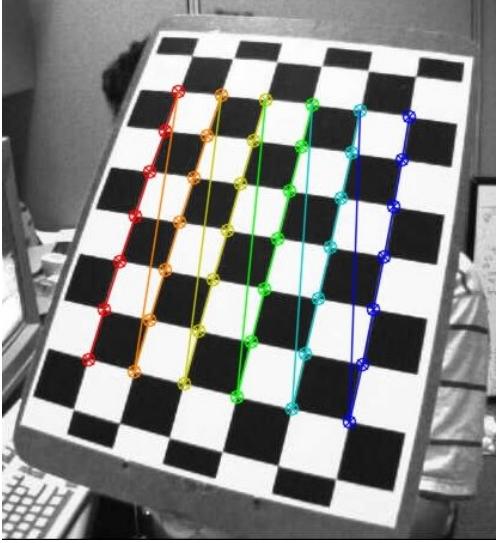


Figure 2.20: Chessboard pattern used in camera calibration with OpenCV.

2.4.7 What is an ArUco?

ArUco, whose name comes from the combination of "Artificial" and "Uco" (for the University of Córdoba, where it was developed), is an open-source library widely recognized in the field of computer vision for detecting fiducial markers in images. This technology is essential for estimating the camera pose relative to the markers when the camera has been previously calibrated. According to the documentation, "*ArUco is an OpenSource library for detecting squared fiducial markers in images*" [8]. Detecting these markers is crucial in applications requiring precise estimation of the position and orientation of objects in three-dimensional space.

History of ArUco Markers

ArUco markers were developed as a solution to overcome the limitations of other technologies for detecting patterns, colors, or shapes. The goal was to create a technique that provides high reliability even under partial occlusions and varying lighting conditions. Early studies focused on the automatic generation of markers with a design that ensured their uniqueness and ease of detection. These markers consist of a binary pattern surrounded by a black border, enhancing their visibility and robustness under different lighting conditions [8].

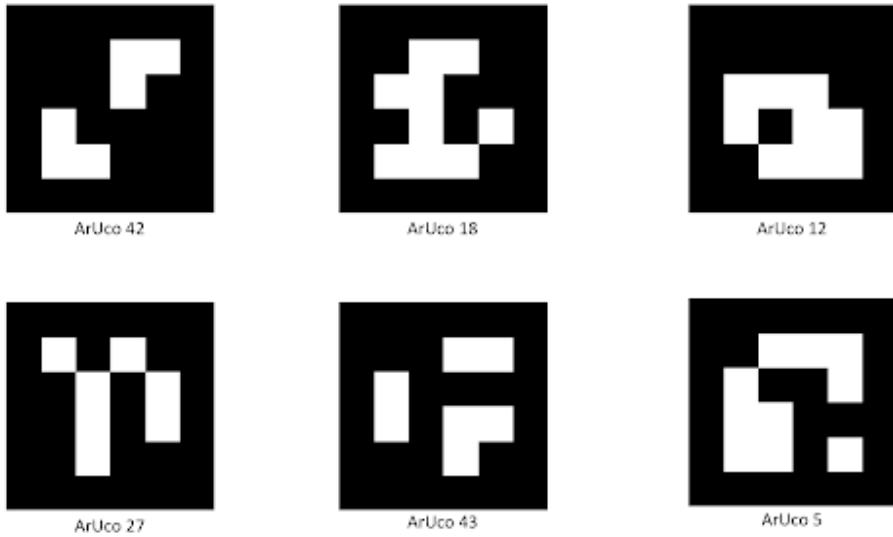


Figure 2.21: Examples of ArUco markers and IDs.

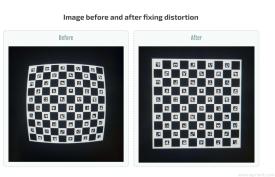


Figure 2.22: Camera Calibration with ArUco

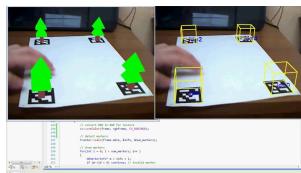


Figure 2.23: Augmented Reality with ArUco

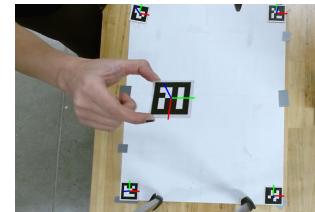


Figure 2.24: Real-Time Tracking with ArUco

Figure 2.25: Common applications of ArUco markers.

Common Applications

ArUco markers are used in various applications, including camera calibration, augmented reality, and robot and drone navigation and control. One of the advantages of using ArUco is their ability to act as reference points in 3D environments, enabling computer vision systems to calculate the camera pose. According to the documentation, “*Markers can be used as 3D landmarks for camera pose estimation*” [?]. This feature makes markers essential in tracking and positioning systems where precision is critical.

Marker Formats

ArUco markers consist of an outer black border and an internal region encoding a unique binary pattern. Depending on the dictionary being used, the number of bits in the marker varies, affecting the likelihood of confusion with other markers and the detection distance. A higher resolution allows markers to be detected from greater distances but may require more processing [8].

The ArUco library also supports creating custom dictionaries, allowing developers to adapt

markers to the specific needs of their projects. “*The design of a dictionary is important since the idea is that their markers should be as different as possible to avoid confusions*” [?]. This flexibility is especially useful in projects where ensuring the uniqueness and reliability of marker detection in complex environments is crucial.

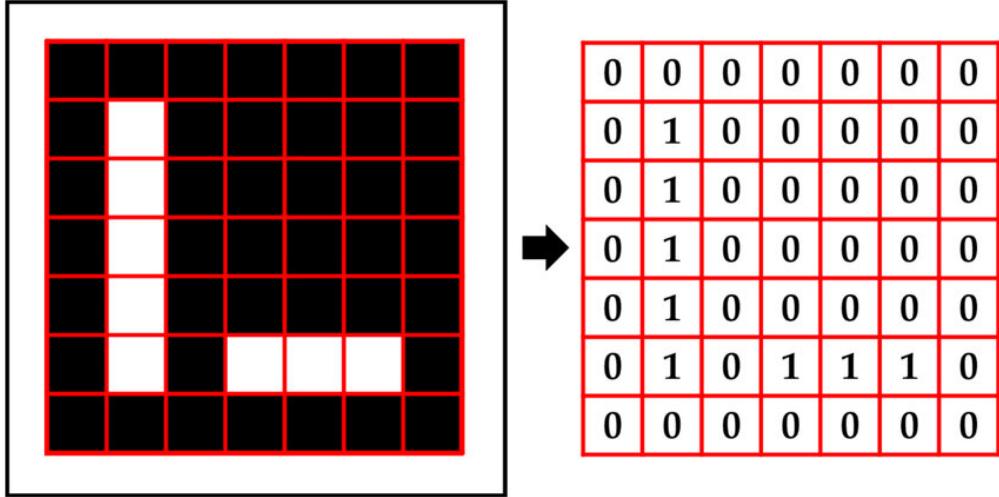


Figure 2.26: Bits of a 7x7 ArUco marker.

2.4.8 Aruco vs. Embedded Aruco

ArUco markers and Embedded ArUco markers (e-ArUco) are technologies used in computer vision for detection and pose estimation tasks. While they share a common basis in their design and detection algorithms, they have significant differences that make them suitable for different applications, particularly in high-precision operations.

Key Differences

The main difference between traditional ArUco markers and Embedded ArUco markers lies in the latter’s optimization for high-precision detection over a wide range of distances. According to Khazetdinov et al. (2021), “*a new type of fiducial marker called embedded ArUco (e-ArUco) was developed specially for a task of robust marker detection for a wide range of distances*” [9]. e-ArUco markers are designed to maintain detectability and precision in scenarios where standard ArUco markers might not perform as effectively, such as when millimeter-level accuracy is required in UAV landing applications.

Another significant difference is that e-ArUco markers are designed to improve detection robustness, minimizing errors that could arise from changing lighting conditions and partial occlusions. These markers build upon ArUco detection algorithms, allowing implementation without substantial changes to existing ArUco-based systems [9].



Figure 2.27: Large/Small ArUco Marker Generation.

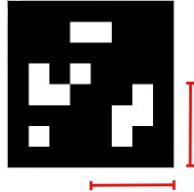


Figure 2.28: Calculating the Large ArUco Marker Center.

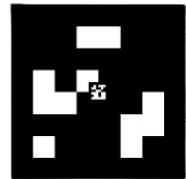


Figure 2.29: Overlaying the Small ArUco Marker on the Large Center.

Figure 2.30: Generation process of e-ArUco markers.

Use Cases

Traditional ArUco markers, as mentioned in the previous section, are commonly used in applications such as augmented reality, pose estimation, and robot and drone navigation. These markers are versatile and adaptable to various applications that do not require extreme precision, making them ideal for localization, augmented reality, etc.

On the other hand, Embedded ArUco markers (e-ArUco) are specifically designed for scenarios where higher precision is critical. A notable example is their use in UAV (Unmanned Aerial Vehicle) landing, where precise and reliable detection is needed across different distances. In a study by Khazetdinov et al., “*an average landing accuracy was 2.03 cm with a standard deviation of 1.53 cm*” was achieved using e-ArUco markers and a landing algorithm implemented in ROS and tested in the Gazebo simulator [9]. This capability makes e-ArUco markers ideal for environments requiring millimeter-level precision, such as in high-precision autonomous landing operations.

2.4.9 Aruco Detection

ArUco marker detection is an essential process in computer vision that enables the identification and pose estimation of markers in images. The following section details detection algorithms, OpenCV implementation, and parameters that affect detection accuracy.

Detection Algorithms

ArUco marker detection relies on computer vision algorithms that identify contours and specific patterns in images. According to OpenCV documentation, the detection process begins by identifying squares in the image and verifying if they contain a valid binary pattern corresponding to an ArUco marker [?]. The algorithm implemented in OpenCV uses contour segmentation and edge detection to find square regions, which are then checked to determine if they match the patterns in the ArUco dictionary.

Once a marker is identified, the algorithm calculates the camera’s pose relative to the marker using inverse projection. This process is particularly important in applications requiring the calculation of the camera’s position and orientation for robot and drone navigation and control.

OpenCV Implementation

OpenCV provides a robust implementation for detecting ArUco markers through the `cv::aruco` module. The main function for detection is `cv::aruco::detectMarkers`, which identifies markers in an image and returns their corners and corresponding IDs. OpenCV documentation highlights that “*the function detects the markers and returns their IDs and corner positions in the image*” [?].

The following example demonstrates how to use OpenCV to detect ArUco markers in Python:

```
import cv2
import cv2.aruco as aruco

# Load the image
image = cv2.imread('image_path.jpg')

# Define the ArUco dictionary
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)

# Detect markers
corners, ids, _ = aruco.detectMarkers(image, aruco_dict)

# Draw detected markers
if ids is not None:
    aruco.drawDetectedMarkers(image, corners, ids)
```

After obtaining the corners and IDs of the markers, this information can be used to calculate the camera’s pose relative to the detected marker(s). To calculate the center of an ArUco marker, the detected corners and marker size can be used to estimate its position in the image.

Accuracy Parameters

The accuracy of ArUco marker detection depends on several factors, including image quality, marker size, and camera calibration parameters. According to OpenCV documentation, precise camera calibration is crucial to minimize errors in pose estimation [?]. Key parameters affecting detection include:

- **Lens distortion:** Correcting lens distortion improves detection accuracy.
- **Marker resolution:** Higher-resolution markers allow for more accurate detection at greater distances but require more processing power.

- **Lighting and contrast:** Detection can be affected by varying lighting conditions, so it is important for the image to have good contrast between the marker and the background.

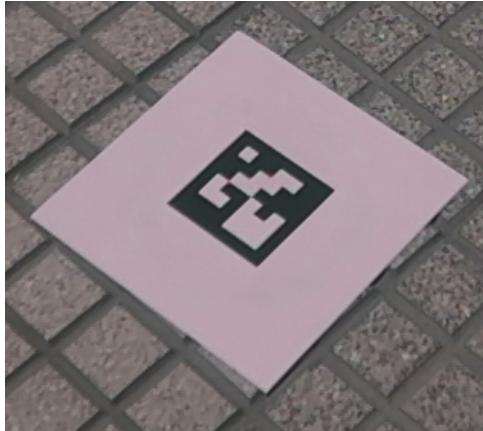


Figure 2.31: ArUco without pose estimation.

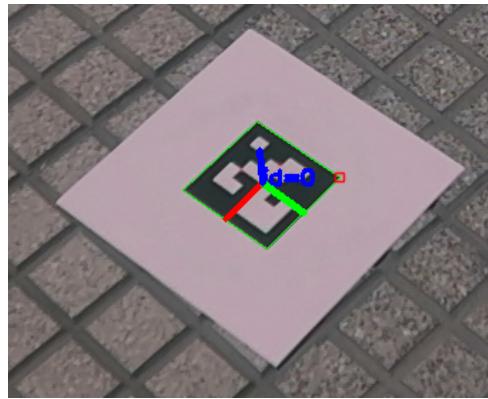


Figure 2.32: ArUco with pose estimation.

2.4.10 ROS2-Pixhawk Communication

Communication between ROS 2 and Pixhawk is based on the MAVLink protocol, a lightweight protocol that enables bidirectional data transfer between the flight controller and a companion computer. This integration is critical for sending real-time flight commands and monitoring the drone’s status.

MAVLink Protocol

MAVLink (Micro Air Vehicle Link) is a high-performance communication protocol designed for unmanned vehicle systems. According to the official documentation, “*MAVLink is a very lightweight, header-only message marshalling library for micro air vehicles*” [?]. This protocol uses a message-based packet system that facilitates data transmission between the ground control station and the unmanned vehicle.

MAVLink messages are structured into specific commands that control various aspects of flight and telemetry, including parameters such as position, speed, battery status, and flight control commands. Each message is identified by a unique ID, which simplifies processing and enables efficient real-time communication.

MAVLink Integration Options in ROS 2

To facilitate MAVLink integration with ROS 2, several communication options exist for sending and receiving data between a companion computer and the Pixhawk flight controller. The two most popular solutions are MAVROS and Micro XRCE-DDS.

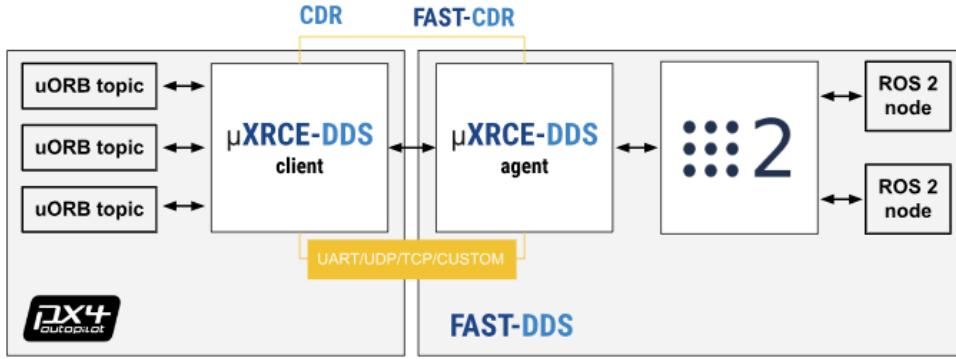


Figure 2.33: ROS 2 - Pixhawk communication using Micro XRCE-DDS.

MAVROS MAVROS is a set of ROS nodes that act as an interface between ROS and MAVLink, allowing ROS to communicate with the flight controller through topics and services. MAVROS enables the publication and subscription of MAVLink messages via ROS topics, facilitating flight command execution, telemetry data retrieval, and mission control.

MAVROS implements a series of predefined ROS topics, such as:

- `/mavros/setpoint_position/local`: To set position waypoints.
- `/mavros/state`: Provides the current state of the drone, including the flight mode and whether the vehicle is armed.
- `/mavros/imu/data`: IMU data for monitoring orientation and acceleration.
- `/mavros/battery`: Displays battery data, such as charge percentage and voltage.

With MAVROS, developers can send control commands, such as arming the drone, changing flight modes, or setting waypoints for autonomous navigation. This is achieved by publishing messages to the appropriate topics and adjusting parameters in real-time via the MAVLink interface [?].

Micro XRCE-DDS Micro XRCE-DDS is a lightweight implementation of DDS (Data Distribution Service) that allows efficient communication between ROS 2 and the flight controller in systems with resource constraints. This option is useful in contexts where system size and performance are critical, such as small drones or those with limited processing hardware.

With Micro XRCE-DDS, ROS 2 can directly communicate with PX4 using the DDS middleware. This enables the creation of a distributed and scalable system, where ROS 2 nodes can send and receive MAVLink messages via DDS topics, similar to MAVROS but with optimized processing overhead [?].

2.4.11 Sensores para la Navegación en Ambientes Exteriores e Interiores

Para lograr el control preciso de la posición del dron mediante los topics de ROS 2 como `/mavros/setpoint_position` es necesario contar con sensores que permitan obtener datos de posicionamiento confiables en función del entorno en el que opere el dron. A continuación, se describen los sensores comúnmente utilizados en ambientes exteriores e interiores para proporcionar información de posición.

Sensores para Ambientes Exteriores

En entornos exteriores, los drones suelen utilizar sensores de posicionamiento global (GPS) para determinar su ubicación en tiempo real. A continuación se presentan los sensores más comunes en exteriores:

- **GPS (Sistema de Posicionamiento Global):** El GPS es el sensor principal para la navegación en exteriores. Proporciona coordenadas de latitud, longitud y altitud que permiten al dron determinar su posición global. En ROS 2, estos datos se suelen publicar en topics como `/mavros/global_position/global` y se integran en los sistemas de control de posición.
- **RTK-GPS (Real-Time Kinematic GPS):** Para aplicaciones de alta precisión, como el aterrizaje de precisión o la navegación en áreas restringidas, se utiliza RTK-GPS. Este sistema mejora la precisión del GPS mediante la corrección de datos en tiempo real, alcanzando precisiones de centímetros. RTK-GPS es compatible con MAVLink y permite un control de posición altamente preciso en ROS 2.
- **IMU (Unidad de Medición Inercial):** Aunque no proporciona datos de ubicación directa, la IMU complementa al GPS detectando cambios en orientación y aceleración. Estos datos son cruciales para mantener la estabilidad y proporcionar información sobre la orientación del dron durante el vuelo.

Sensores para Ambientes Interiores

En ambientes interiores, donde las señales de GPS suelen ser débiles o inexistentes, se emplean otros sensores para determinar la posición y navegación del dron. Los sensores utilizados en interiores incluyen:

- **Cámaras de Visión (Monocular o Estéreo):** Las cámaras de visión permiten la localización visual mediante algoritmos de SLAM (Simultaneous Localization and Mapping) o detección de marcadores, como ArUco, para calcular la posición relativa del dron en interiores. Con ROS 2, estas cámaras pueden integrarse con librerías como OpenCV y utilizar los topics de imagen para el procesamiento en tiempo real.
- **LIDAR (Light Detection and Ranging):** Los sensores LIDAR emiten pulsos de luz para medir distancias y obtener mapas detallados del entorno. Estos sensores son útiles para la navegación y el mapeo en tiempo real en interiores y se integran en ROS 2 mediante

topics de nube de puntos. Son particularmente efectivos para evitar obstáculos y realizar posicionamiento preciso en interiores.

- **IMU (Unidad de Medición Inercial):** En interiores, la IMU sigue siendo un componente esencial para detectar cambios de orientación y movimiento. La fusión de datos de la IMU con otros sensores, como LIDAR o cámaras, ayuda a mantener la estabilidad y proporciona estimaciones de posición precisas mediante filtros de fusión, como el filtro de Kalman.

2.4.12 Fusión de Sensores usando el Filtro de Kalman en Pixhawk

La fusión de sensores es un proceso crucial en la navegación de drones, permitiendo integrar datos de múltiples sensores para obtener una estimación de posición y orientación más precisa. En el Pixhawk, este proceso se realiza mediante el filtro de Kalman extendido (EKF), el cual combina datos de sensores como GPS, IMU, Barómetro, Compass y otros sensores adicionales que pueden fusionarse. A continuación, se detallan los requisitos y configuraciones necesarias para que el sistema de fusión de sensores funcione adecuadamente en cada tipo de entorno.

Fusión de GPS e IMU en Exteriores

En entornos exteriores, el GPS es el sensor principal para la localización global, mientras que la IMU proporciona datos de orientación y aceleración. La fusión de estos sensores en Pixhawk se logra mediante el uso del EKF, que mejora la precisión y confiabilidad de la navegación al compensar posibles errores en las lecturas del GPS y la IMU.

- **Configuración de Pixhawk para GPS e IMU:** Para activar la fusión de GPS e IMU en Pixhawk, es necesario habilitar el EKF y configurar parámetros específicos, tales como:
 - `EKF2_GPS_CHECK`: Habilita la verificación de la calidad de la señal de GPS, garantizando que solo se utilicen datos de GPS cuando la señal sea suficiente.
 - `EKF2_HGT_MODE`: Establece la fuente de altitud para el filtro de Kalman. En exteriores, se recomienda configurarlo para usar datos de GPS o de barómetro.
 - `EKF2_AID_MASK`: Activa el soporte para sensores adicionales. Para la fusión de GPS e IMU, este parámetro debe incluir *GPS* y *IMU*.
- **Requisitos de Precisión de GPS e IMU:**
 - **GPS**:** Es ideal contar con un GPS de alta precisión o un sistema RTK-GPS en aplicaciones donde se requiera una precisión de centímetros.
 - **IMU**:** Es importante que la IMU esté correctamente calibrada y que el Pixhawk esté montado en una estructura estable para reducir el ruido de las lecturas.
- **Proceso de Fusión con EKF:** El EKF en Pixhawk realiza un cálculo continuo de la posición y velocidad del dron al combinar los datos de GPS e IMU. El GPS proporciona la ubicación

global (latitud, longitud y altitud), mientras que la IMU contribuye con datos de orientación y aceleración. El EKF utiliza estos datos para reducir el error acumulativo y generar una estimación de posición precisa y estable.

Desafíos en la Fusión de Sensores en Ambientes Dinámicos

La fusión de sensores en Pixhawk enfrenta desafíos en entornos dinámicos o con interferencias, como en interiores con superficies reflectantes para LIDAR o en exteriores con pérdida de señal GPS. Para minimizar estos problemas, es importante realizar calibraciones periódicas de todos los sensores y ajustar los parámetros del EKF según el entorno y las condiciones de vuelo.

Mediante la fusión de sensores con el EKF en Pixhawk, los drones pueden operar de forma autónoma y precisa tanto en exteriores como en interiores, adaptándose a las condiciones y requerimientos específicos de cada entorno.

Fusión de Sensores en Interiores sin GPS

Para lograr la navegación autónoma en interiores, donde el GPS no está disponible, se emplean sensores alternativos que permiten al dron estimar su posición relativa y mantener un vuelo estable. La fusión de estos sensores se realiza mediante el filtro de Kalman extendido (EKF) en el Pixhawk, que combina datos de diferentes sensores para obtener una estimación precisa de posición y orientación. A continuación se describen los sistemas de posicionamiento más destacados y los requisitos para lograr una fusión de sensores efectiva en interiores.

Sensores Alternativos para Navegación en Interiores

En entornos cerrados, el uso de tecnologías como cámaras, beacons, y sistemas de seguimiento óptico permite al dron obtener información de posición sin depender del GPS. Estos sistemas requieren que se defina manualmente el origen de la posición inicial, usando una estación de control en tierra (GCS) como Mission Planner o mediante scripts en Lua. Algunos de los sensores más destacados incluyen:

- **Intel RealSense T265 y Luxonis OAK-D:** Ambas son cámaras de visión estéreo que pueden proporcionar estimaciones de posición mediante técnicas de SLAM (Simultaneous Localization and Mapping). Estas cámaras permiten calcular la posición y la orientación del dron en tiempo real, y pueden integrarse en el Pixhawk usando el EKF para fusionar estos datos con la IMU. La configuración requiere calibrar correctamente las cámaras y establecer el origen de posición.
- **MarvelMind Beacons y Pozyx Beacons:** Estos sistemas basados en balizas (beacons) usan señales de radio para determinar la posición relativa del dron. Los beacons se instalan en ubicaciones fijas dentro del entorno, y el dron calcula su posición en función de la distancia a cada beacon. Este tipo de sistema es muy preciso y puede proporcionar una estimación

de posición estable, aunque requiere de una infraestructura inicial de beacons correctamente ubicados en el área.

- **ModalAI VOXL y Nokov Indoor Optical Tracking:** Estos sistemas combinan visión por computadora y algoritmos de seguimiento óptico para proporcionar datos de posicionamiento precisos en interiores. ModalAI VOXL, por ejemplo, es un módulo que integra una computadora de alto rendimiento con cámaras de visión para permitir una navegación autónoma sin GPS. El Pixhawk puede fusionar estos datos de posición y orientación en tiempo real mediante el EKF, mejorando la estabilidad del vuelo en interiores.
- **Vicon Positioning System y OptiTrack Motion Capture System:** Estos sistemas de captura de movimiento utilizan cámaras de alta precisión colocadas en el entorno para rastrear marcadores ubicados en el dron. Estos sistemas son muy efectivos para aplicaciones de navegación de alta precisión, como en laboratorios o áreas de prueba, y permiten obtener una posición con precisión milimétrica. Sin embargo, requieren un entorno controlado y la instalación de múltiples cámaras en el área.
- **Optical Flow:** Este sensor detecta el movimiento relativo entre el dron y la superficie debajo de él. Aunque no proporciona una posición absoluta, el flujo óptico es útil para mantener la estabilidad del dron en interiores y para pequeñas correcciones de posición. La combinación de datos de flujo óptico con la IMU permite una navegación precisa a nivel local.

Requisitos para la Fusión de Sensores sin GPS

La fusión de sensores sin GPS en interiores requiere de un entorno controlado donde estos sistemas de posicionamiento puedan operar de manera efectiva. Los siguientes son algunos de los requisitos necesarios para configurar y utilizar estos sensores en Pixhawk:

- **Definir el Origen de la Posición:** Al utilizar sensores de visión, beacons o sistemas de captura de movimiento, el origen de la posición debe definirse manualmente a través del GCS o con un script en Lua para que el dron pueda interpretar correctamente su posición en relación con el entorno.
- **Calibración de Sensores de Visión y Beacons:** La precisión de los datos de posición depende de una calibración adecuada de los sistemas de visión y beacons. Esto incluye la alineación de cámaras y beacons, así como la configuración de parámetros en el EKF para optimizar la precisión de los datos.
- **Calidad de IMU:** La IMU es fundamental para la fusión de sensores en interiores, ya que proporciona datos de orientación y aceleración que complementan los datos de posicionamiento relativo. Las IMUs de bajo costo pueden experimentar un alto nivel de deriva, por lo que es recomendable utilizar IMUs de mayor calidad o realizar una calibración frecuente.

- **Uso del Filtro de Kalman (EKF) en Pixhawk:** El EKF en Pixhawk debe configurarse para aceptar las entradas de los sensores seleccionados. Esto incluye activar los parámetros necesarios en el Pixhawk, tales como `EKF2_AID_MASK` para habilitar `VISION_POSITION`, `RANGE_FINDER`, o `FLOW` dependiendo del sensor utilizado, y ajustar los valores de ruido según las especificaciones del sistema de posicionamiento.

Estos sensores y configuraciones permiten que el dron mantenga un vuelo autónomo y preciso en entornos interiores sin GPS, proporcionando alternativas para aplicaciones de navegación autónoma en espacios cerrados.

Chapter 3

Desarrollo

Este capítulo describe el proceso de diseño, implementación y configuración de la estación de carga modular y la instrumentación del dron. Se detallan las metodologías y herramientas utilizadas para llevar a cabo el desarrollo, con un enfoque en la creación de una solución eficaz para el sistema de carga.

3.1 Diseño y Desarrollo de la Base de Carga

3.1.1 Especificaciones y Requerimientos

A continuación se presentan las especificaciones técnicas y los requisitos para la base de carga, garantizando compatibilidad y eficiencia para un sistema de carga en enjambres de drones:

1. Diseño modular y apilable para facilitar el almacenamiento de múltiples drones en un mismo metro cuadrado.
2. Las dimensiones mínimas requeridas para la estación de carga debe ser de mínimo 10 porciento mayores en ancho, largo y alto que las del dron.
3. La estacion de carga deberá contar con un mecanismo de carga para la recarga de la batería del dron.
4. Debe contar con por lo menos un método de posicionamiento visible para el dron.

3.1.2 Lista de Materiales

- Perfiles de Aluminio
 - **30 mm:**
 - * 4 x Perfiles de 100 mm
 - * 4 x Perfiles de 810 mm
 - * 6 x Perfiles de 750 mm

– **40 mm:**

- * 2 x Perfiles de 1040 mm
- * 4 x Perfiles de 560 mm
- * 4 x Perfiles de 830 mm
- * 4 x Perfiles de 1120 mm
- 16 x Codos para perfil de aluminio impresos en 3D
- Tronillería para perfil de aluminio
- 2 x Riel Corredora Telescópica
- 1 x Motor de 12V
- 1 x Soportes de Motor impresos en 3D
- 1 x Banda de 1 m
- 2 x Polea tipo A con diámetro interno de 1/2"
- 2 x Balero de DI: 15 mm y DE: 35 mm
- 2 x Chumacera impresa en 3D
- 2 x MDF de 9 mm de 75x75 cm
- 8 x Separadores de MDF impresos en 3D
- 1 x Fuente de Poder de 12V
- 1 x Arduino Mega
- 1 x Raspberry Pi 4
- 1 x Puente H BTS7960
- 2 x Sensor de Proximidad Inductivo
- 1 x Joystick
- 1 x Router WiFi
- 1 x BT3 Pro Compact Charger
- 1 X Cable Carrier 1 m

3.1.3 Selección e Implementación del Mecanismo de Movimiento del Cajón

En la sección 2.2.3 se explican en detalle los diferentes mecanismos de movimiento que se consideraron para la estación de carga, evaluando sus ventajas y desventajas en términos de precisión, durabilidad, costo y facilidad de integración en el diseño modular. Se analizaron opciones como pistones lineales, bandas dentadas y otros sistemas de transmisión, con el objetivo de seleccionar una solución que cumpliera con los requisitos de funcionalidad sin elevar demasiado los costos. Tras esta evaluación, se optó por un mecanismo de Polea y Banda tipo V, el cual, además de ser una opción económica, utiliza componentes ya disponibles, como el motor. Este sistema permite un movimiento horizontal suficiente para posicionar el dron con la precisión necesaria para el proceso de carga, a la vez que facilita el apilamiento modular de la estación. La orientación del movimiento no obstruye la estructura general de la estación, lo cual asegura que el diseño cumpla con el requisito de apilabilidad y compatibilidad en espacios compartidos, manteniéndose compacto y adaptable sin comprometer la estructura modular de la estación.

Para este mecanismo, se seleccionaron diferentes componentes, incluyendo una banda de alta resistencia para soportar las cargas previstas, poleas (con baja fricción) para optimizar el movimiento y reducir el desgaste del sistema en operaciones continuas, y un motor ... que tiene un torque de ... y cumple con el torque necesario para mover los 3 kg de peso del drone.

Diseños CAD y Componentes

1. Motor de 12V
2. Poleas
3. Banda
4. Baleros
5. Chumaceras
6. Soporte del Motor

Mecanismo Polea y Banda tipo V

(Imagen del Mecanismo)

3.1.4 Diseño CAD de la Base de Carga

Para cumplir con los requerimientos de diseño modular / apilable y compatibilidad con el drone en cuestión a las dimensiones de este. Se llevó a cabo el siguiente diseño en CAD:

- El diseño de la estructura principal de la estación de carga se realizó en Fusion 360, considerando las dimensiones mínimas requeridas para el dron y el mecanismo de carga. Imagen

- Se tomó en cuenta la dimensión necesaria para que el drone tenga un espacio de por lo menos 10 cm de separación con la pared del cajón, esto para evitar alguna colisión del drone con la estructura de la base de carga al momento de intentar aterrizar, además se disminuyó la altura del cajón para que el drone pueda aterrizar sin problemas. Por otro lado se usarán rieles que se obtuvieron previamente reutilizados, por lo cual la estación de carga física se encuentra salida de la base de carga por unos 5 cm al momento de estar cerrada, pero en el diseño CAD el diseño si se encuentra cerrado por completo. Imagen
- Se añadió un espacio de 25 cm para colocar la parte electrónica del cajón. Imagen

3.1.5 Proceso de Manufactura de la Estructura de la Base de Carga

1. **Corte del Material:** Para iniciar el proceso de manufactura, se realizaron cortes precisos de los perfiles de aluminio y las placas de MDF según las dimensiones especificadas en la lista de materiales. Este paso es fundamental para asegurar que todas las piezas se ensamblen correctamente en el diseño modular.

Imagen del proceso de corte de perfiles de aluminio y MDF

2. **Ensamblaje de la Estructura:** Una vez cortadas las piezas, se procedió a ensamblar la estructura principal de la estación de carga utilizando tornillos y tuercas para fijar los perfiles de aluminio y las placas de MDF usando los separadores impresos. Se verificó que todas las piezas estuvieran alineadas y niveladas para garantizar la estabilidad y resistencia de la base de carga.

Imagen del proceso de ensamblaje de la estructura de la base de carga

3. **Instalación del Mecanismo de Movimiento:** Despues de ensamblar la estructura principal, se instaló el mecanismo de polea y banda tipo V para permitir el movimiento horizontal del cajón. Se colocaron las poleas, la banda y el motor en las ubicaciones previamente definidas, asegurando que el sistema de movimiento funcionara correctamente y sin obstrucciones.

Imagen del proceso de instalación del mecanismo de movimiento

3.1.6 Circuito Electrónico de la Estación de Carga

Diagrama de Conexiones

Añadir diagrama de conexiones

Construcción del Circuito

Añadir imágenes de la construcción del circuito

Programación del Circuito

Explicar el código de programación del circuito y su funcionamiento, mencionar que en los anexos se encuentra el código fuente del arduino y raspberry pi 4.

3.2 Instrumentación del Dron

3.2.1 Especificaciones y Requerimientos

Para garantizar la compatibilidad y funcionalidad en el sistema de carga, el dron debe cumplir con las siguientes especificaciones:

- Las dimensiones de las piezas deberán adaptarse al frame del drone cuadricoptero de arquitectura abierta que fue comprado.
- El dron deberá tener un circuito de carga compatible con la estacion de carga.
- El drone deberá contar con una cámara y un sistema de visión para la detección de marcadores Aruco.
- Se deberán integrar sensores de localización para la navegación en exterior.
- Se deberá integrar algún microprocesador como computadora auxiliar para el procesamiento de datos y la comunicación con la estación de carga.

3.2.2 Lista de Materiales para la Instrumentación del Dron

- 1 x Frame de cuadricóptero abierto (especificar modelo)
- 1 x Controlador de vuelo (especificar modelo)
- 1 x Cámara (compatible con detección Aruco)
- 1 x Raspberry Pi 4 (Companion Computer)
- 1 x Sensor de proximidad (modelo de preferencia)
- 1 x Módulo GPS (compatible con el controlador de vuelo)
- Cableado y conectores
- Material de montaje impreso en 3D (soportes específicos para cada componente)

3.2.3 Diseños CAD del Dron

Se presentan a continuación los diseños CAD de las piezas que se han desarrollado para el dron, adaptadas a su frame original para integrar los componentes electrónicos necesarios y cumplir con los requisitos de carga y posicionamiento.

- Se diseñaron nuevos soportes para el microprocesador y la cámara, asegurando una integración estable y precisa en el frame.
- Se implementó un espacio de montaje para los sensores de localización y el circuito de carga.

Imagen de los diseños CAD del dron con las piezas modificadas

3.2.4 Circuito de Distribución de Energía

El circuito de distribución de energía proporciona alimentación segura y estable a los componentes del dron. La selección de baterías y reguladores de voltaje fue optimizada para asegurar la duración de vuelo y protección de cada componente.

- Se seleccionaron baterías de litio-polímero (LiPo) de alta capacidad para maximizar la autonomía.
- Reguladores de voltaje se añadieron para adaptar el suministro a la Raspberry Pi, la cámara y el controlador de vuelo.

Imagen del circuito de distribución de energía en el dron

Integración de Componentes Electrónicos

Cada componente fue ensamblado y cableado de acuerdo con el diseño modular del dron. A continuación se describen los pasos del proceso de integración:

1. **Montaje de la Cámara:** La cámara se fijó en la parte de abajo del dron, permitiendo una visibilidad óptima para la detección de marcadores Aruco.

Imagen del montaje de la cámara en el dron

2. **Instalación del Controlador de Vuelo y Módulo GPS:** El controlador de vuelo se instaló en el centro del frame para optimizar el balance y calibración, y el módulo GPS se colocó en una de las patas.

Imagen del montaje del controlador de vuelo y módulo GPS

- 3. Instalación del Microprocesador:** La Raspberry Pi 4 se integró en un soporte impreso en 3D en la parte superior, y el microprocesador se colocó sobre este soporte. Un cable de comunicación serial se conectó entre la Raspberry Pi y el controlador de vuelo que se encuentra debajo de el microprocesador.

Imagen del montaje del microprocesador en el dron

3.3 Software Configuration

3.3.1 Configuración de la Computadora Central en Tierra

La computadora central se configuró para supervisar y procesar la información proveniente de la computadora auxiliar del dron. A continuación, se describen los pasos realizados para esta configuración:

- **Instalación de ROS 2 Humble:** Dado que la computadora central utiliza Ubuntu 22.04, se instaló ROS 2 Humble siguiendo las instrucciones oficiales. Los comandos utilizados fueron:

```
sudo apt update && sudo apt install -y software-properties-common  
sudo add-apt-repository universe  
sudo apt update  
sudo apt install -y ros-humble-desktop
```

Esto incluyó la instalación de las herramientas necesarias para desarrollar y ejecutar aplicaciones en ROS 2 Humble.

- **Configuración del entorno:** Para facilitar el uso de ROS 2, se configuró el archivo `/.bashrc`. Se agregó la siguiente línea al final del archivo:

```
source /opt/ros/humble/setup.bash
```

Posteriormente, se ejecutó:

```
source ~/.bashrc
```

- **Clonación del repositorio de GitHub:** Se creó un espacio de trabajo para ROS 2 y se clonó el repositorio correspondiente. Los pasos realizados fueron:

```
mkdir -p ~/ros2_ws/src  
cd ~/ros2_ws/src  
git clone <URL_del_repositorio>  
cd ..  
colcon build
```

- **Configuración del ROS_DOMAIN_ID:** Para garantizar una correcta comunicación entre la computadora central y la computadora auxiliar del dron, se configuró el `ROS_DOMAIN_ID` con el valor 10. Esto se realizó añadiendo la siguiente línea al archivo `/.bashrc`:

```
export ROS_DOMAIN_ID=10
```

Luego, se ejecutó:

```
source ~/.bashrc
```

- **Verificación del entorno de ROS 2:** Finalmente, se verificó que el entorno estuviera correctamente configurado utilizando los siguientes comandos:

```
ros2 doctor
```

Esto permitió confirmar que todas las dependencias necesarias para ROS 2 Humble estuvieran instaladas y funcionando correctamente.

Imagen de la configuración del entorno en la computadora central.

3.3.2 Configuración de la Computadora Auxiliar del Dron

La Raspberry Pi 5 se configuró como computadora auxiliar para el procesamiento de datos y comunicación en tiempo real con la estación de carga. A continuación, se detallan los pasos realizados para su configuración:

- **Instalación de Ubuntu 24.04:** Se utilizó la herramienta Raspberry Pi Imager para instalar Ubuntu Server 24.04 LTS (64-Bit) en la tarjeta SD. Durante la configuración inicial, se habilitó SSH, se definió un usuario con contraseña y se conectó la Raspberry Pi a la red WiFi.
- **Conexión inicial y SSH:** Después de insertar la tarjeta SD en la Raspberry Pi y conectarla a un monitor y teclado, se encendió el dispositivo e ingresaron las credenciales configuradas. Se obtuvo la dirección IP mediante el comando:

```
hostname -I
```

Con esta información, se estableció una conexión SSH desde una computadora externa utilizando:

```
ssh <usuario>@<IP>
```

Esto permitió continuar con la configuración de la Raspberry Pi de forma remota.

- **Actualización del sistema:** Se realizó una actualización completa del sistema operativo con los siguientes comandos:

```
sudo apt update && sudo apt upgrade -y
```

También se instaló `raspi-config` para habilitar el puerto serial. Esta opción se configuró en **Interface Options > Serial Port**, seleccionando **No** y luego **Yes**.

- **Modificación de bashrc y configuración del ID:** Para garantizar la comunicación entre todos los dispositivos, se configuró el `ROS_DOMAIN_ID` con el valor 10. Se editó el archivo `/.bashrc` con:

```
sudo vim ~/.bashrc
```

Al final del archivo, se agregaron las siguientes líneas:

```
export ROS_DOMAIN_ID=10
source /opt/ros/jazzy/setup.bash
```

Posteriormente, se guardaron los cambios y se ejecutó:

```
source ~/.bashrc
```

- **Instalación de ROS 2 Jazzy:** Se siguieron las instrucciones oficiales para instalar ROS 2 Jazzy en Ubuntu 24.04. Esto incluyó los comandos:

```
sudo apt install -y software-properties-common
sudo add-apt-repository universe
sudo apt update
sudo apt install -y ros-jazzy-desktop
```

- **Clonación del repositorio de GitHub:** Se creó un espacio de trabajo en ROS 2 para integrar los nodos personalizados. Los pasos realizados fueron:

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws/src
git clone <URL_del_repositorio>
cd ..
colcon build
```

- **Instalación de MAVROS y MAVProxy:** Para la comunicación con el Pixhawk, se instalaron MAVROS y MAVProxy. Los comandos utilizados fueron:

```
sudo apt install ros-jazzy-mavros ros-jazzy-mavros-extras
sudo rosdep init
rosdep update
sudo apt install python3-mavproxy
```

Además, se configuraron los complementos geográficos necesarios:

```
sudo apt install geographiclib-tools
sudo geographiclib-get-geoids egm96-5
```

- **Verificación de la comunicación:** Para garantizar que MAVROS y MAVProxy funcionaran correctamente, primero se inició MAVProxy con:

```
mavproxy.py --master=/dev/ttyAMA0 --baudrate 921600
```

Posteriormente, se lanzó MAVROS utilizando:

```
ros2 launch mavros px4.launch fcu_url:=serial:///dev/ttyAMA0:921600
```

Finalmente, se verificaron los tópicos disponibles con:

```
ros2 topic list
```

y se confirmó la comunicación observando los mensajes de los tópicos relevantes.

Imagen de la configuración del entorno en la Raspberry Pi 5.

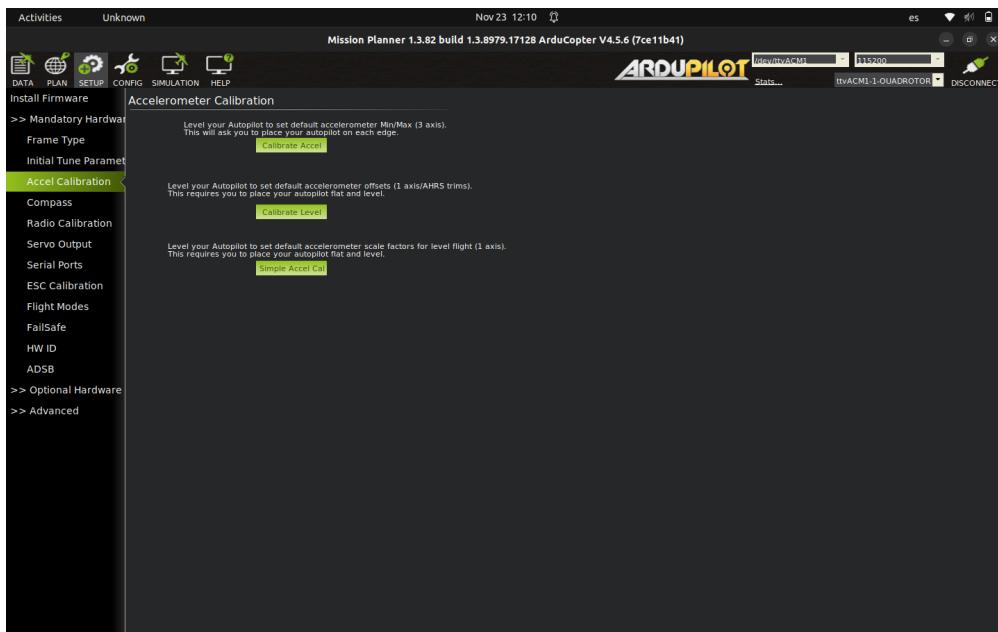


Figure 3.1: Imagen de la configuración de sensores en Mission Planner.

3.3.3 Configuración de la Estación de Control en Tierra

Se detalla la configuración realizada para la estación de control en tierra utilizando las herramientas QGroundControl y Mission Planner. Ambas aplicaciones son similares en funcionalidad, permitiendo la visualización y gestión de parámetros de vuelo. Sin embargo, dado que se utilizó ArduPilot como firmware, se decidió priorizar Mission Planner debido a su optimización para este sistema.

- **Instalación de Mission Planner y QGroundControl:** Se instalaron ambas herramientas en la computadora central para la gestión y monitoreo del Pixhawk. Mission Planner se utilizó principalmente por su compatibilidad directa con ArduPilot, mientras que QGroundControl fue útil en ciertas configuraciones iniciales como algunas calibraciones redundantes.
- **Calibración de sensores:** La calibración de los sensores del Pixhawk que se realizó desde Mission Planner, siguiendo estos pasos:
 - Acceder a la sección de calibración en la pestaña *Initial Setup*.
 - Seleccionar la opción de calibración para cada sensor:
 - * **Acelerómetro:** Se colocó el Pixhawk en diferentes orientaciones según las instrucciones en pantalla para calibrar correctamente.
 - * **Giroscopio:** Se mantuvo el Pixhawk inmóvil durante el proceso de calibración.
 - * **GPS:** Se verificó la recepción de satélites y se ajustaron los parámetros necesarios para una correcta ubicación.

- **Parámetros de comunicación:** Para establecer la comunicación entre la Raspberry Pi y el Pixhawk a través de MAVROS y MAVLink, se realizaron los siguientes ajustes en los parámetros utilizando Mission Planner:
 - SERIAL2_PROTOCOL: Configurado en 2 para habilitar MAVLink.
 - SERIAL2_BAUD: Ajustado a 921600 para coincidir con la configuración de la Raspberry Pi.
 - SYS_ID: Configurado en el ID correspondiente para garantizar una comunicación adecuada con el sistema.

3.4 Sistema de Visión

3.4.1 Especificaciones y Requerimientos

El sistema de visión debe cumplir con los siguientes requisitos para garantizar la detección precisa de marcadores Aruco:

- Obtener las matrices de calibración intrínsecas y extrínsecas de la cámara.
- Generar y detectar marcadores Aruco de distintos tamaños en tiempo real.

3.4.2 Calibración de la Cámara

El proceso de calibración de la cámara se realizó utilizando un script en Python con la biblioteca OpenCV. Este procedimiento se detalla a continuación, combinando fragmentos del código y las imágenes obtenidas en cada etapa. El código completo se encuentra en los anexos del documento.

1. **Captura de imágenes:** Se recopilaron múltiples imágenes del tablero de ajedrez desde diferentes ángulos y distancias para abarcar todo el campo de visión de la cámara. Estas imágenes presentaban distorsiones propias de la lente, como se observa en la Figura 3.2.

Figure 3.2: Ejemplo de imagen descalibrada tomada durante el proceso de captura.

2. **Detección de esquinas:** El script detectó las esquinas internas del tablero de ajedrez mediante la función `cv2.findChessboardCorners`. Posteriormente, se refinaron las esquinas detectadas utilizando `cv2.cornerSubPix`, y se visualizaron las líneas superpuestas en las imágenes de calibración, como se muestra en la Figura 3.3.

```
ret, corners = cv2.findChessboardCorners(gray, (ncols, nrows), None)
corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
```

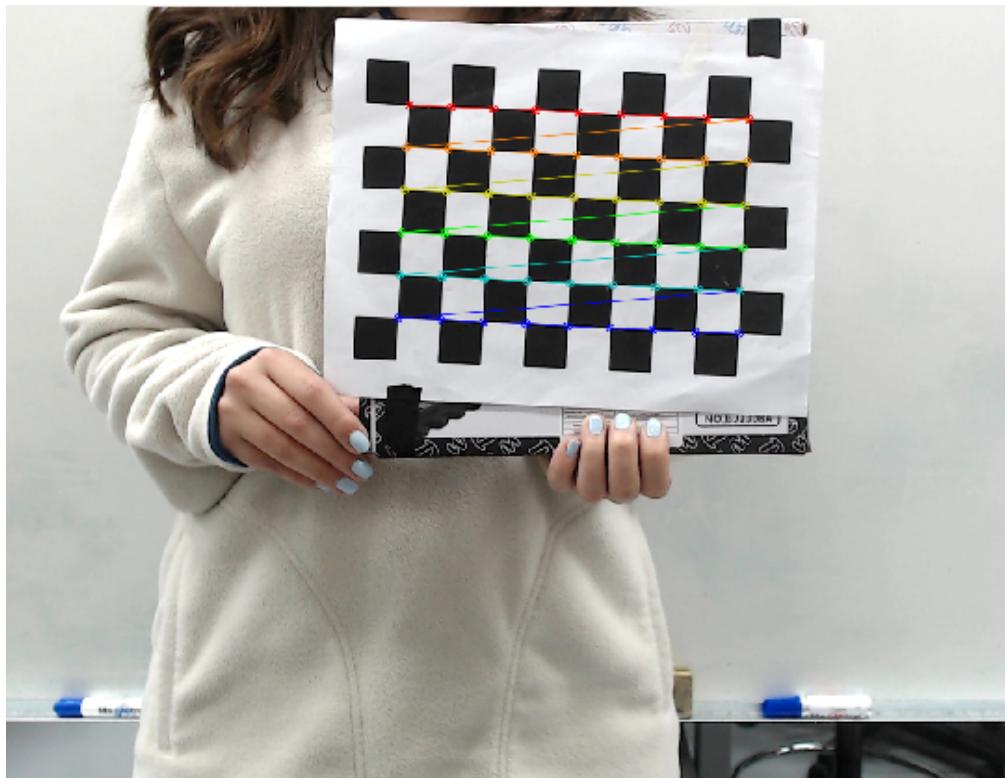


Figure 3.3: Detección y refinamiento de esquinas en el tablero de ajedrez.

3. **Calibración de la cámara:** Utilizando `cv2.calibrateCamera`, se calcularon los parámetros intrínsecos de la cámara y los coeficientes de distorsión. Estos parámetros permiten corregir las distorsiones en las imágenes capturadas. La Figura 3.4 muestra el resultado después de aplicar estos parámetros.

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
                                                 imgpoints,
                                                 img_size,
                                                 None,
                                                 None)
```

Donde:

- **mtx:** Matriz intrínseca de la cámara.
- **dist:** Coeficientes de distorsión.
- **rvecs, tvecs:** Rotaciones y traslaciones.

Figure 3.4: Imagen calibrada después de aplicar los parámetros obtenidos.

4. **Cálculo del error de calibración:** Se verificó la precisión del modelo calculando el error promedio mediante `cv2.projectPoints`. Este cálculo compara las esquinas detectadas y proyectadas.

```
error = cv2.norm(imgpoints[i], imgpoints2, cv2.NORM_L2) / len(imgpoints2)
print("Total error: {}".format(mean_error / len(objpoints)))
```

5. **Almacenamiento de parámetros:** Los parámetros de calibración obtenidos se guardaron en un archivo JSON, lo que permite su reutilización en futuros procesos de corrección de imágenes.

```
data = {"camera_matrix": mtx.tolist(),
        "distortion_coefficients": dist.tolist()}
with open(output_path, "w") as file:
    json.dump(data, file, indent=4)
```

3.4.3 Generación de Marcadores ArUco

Se generaron marcadores ArUco adaptados al sistema, incluyendo tamaños y patrones específicos. En particular, se crearon marcadores embebidos, donde un marcador interno se incrusta dentro de un marcador externo para maximizar la precisión en la detección. A continuación, se describe el proceso paso a paso, destacando las partes importantes del código utilizado. El código completo se encuentra en los anexos del documento.

1. **Carga del diccionario de ArUco:** Se utilizó el diccionario predefinido `DICT_6X6_250` de OpenCV, adecuado para generar marcadores con diferentes patrones y niveles de detalle.

```
aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_6X6_250)
```

2. **Generación del marcador externo:** Se definió un tamaño de 220 píxeles para el marcador externo (`outer_marker_size`) y se generó el marcador con un ID específico (`outer_marker_id`). Esto permitió crear un marcador grande que sirviera como contenedor.

```
outer_marker = aruco.generateImageMarker(aruco_dict,  
                                         outer_marker_id,  
                                         outer_marker_size)
```

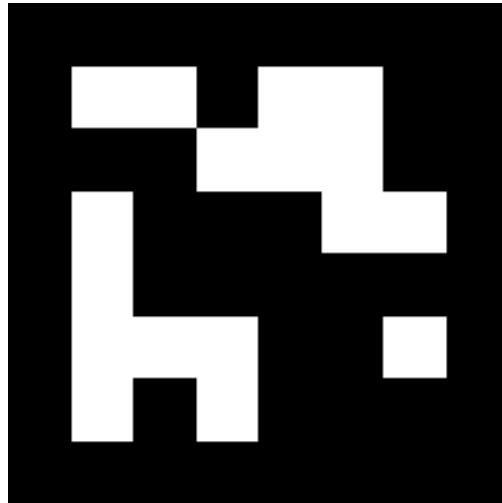


Figure 3.5: Marcador ArUco externo generado.

3. **Generación del marcador interno:** El marcador interno se generó con un tamaño de 50 píxeles (`inner_marker_size`) y un ID específico (`inner_marker_id`). Este marcador se diseñó para ser incrustado dentro del marcador externo.

```
inner_marker = aruco.generateImageMarker(aruco_dict,  
                                         inner_marker_id,  
                                         inner_marker_size)
```

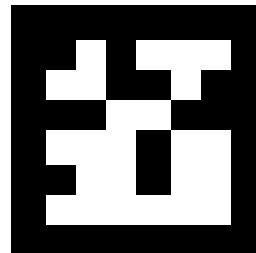


Figure 3.6: Marcador interno con borde blanco añadido.

- 4. Creación del borde blanco alrededor del marcador interno:** Se añadió un borde blanco de un píxel alrededor del marcador interno utilizando `cv2.copyMakeBorder`, lo que facilita su incrustación y aumenta la visibilidad en diferentes condiciones de iluminación.

```
inner_marker_with_border = cv2.copyMakeBorder(
    inner_marker,
    top=border_size,
    bottom=border_size,
    left=border_size,
    right=border_size,
    borderType=cv2.BORDER_CONSTANT,
    value=255 # Blanco
)
```

- 5. Incrustación del marcador interno en el marcador externo:** Se calculó la posición central para incrustar el marcador interno con borde en el marcador externo. El marcador externo se modificó para incluir un fondo negro en el centro antes de insertar el marcador interno con borde.

```
center_position = (outer_marker_size - inner_marker_with_border_size) // 2
e_aruco_marker[center_position:center_position + inner_marker_with_border_size,
               center_position:center_position + inner_marker_with_border_size] =
e_aruco_marker[center_position:center_position + inner_marker_with_border_size,
               center_position:center_position + inner_marker_with_border_size] =
```

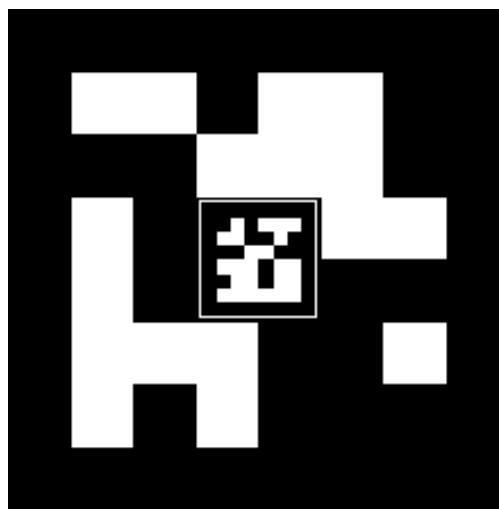


Figure 3.7: Marcador ArUco embebido generado.

6. **Almacenamiento del marcador embebido:** Finalmente, el marcador generado se guardó como una imagen PNG para su uso posterior.

```
cv2.imwrite(f'embedded_aruco_marker_{outer_marker_id}_{inner_marker_id}.png',  
          e_aruco_marker)
```

3.4.4 Detección de Marcadores e-Aruco en Tiempo Real

El sistema fue diseñado para detectar marcadores Aruco en tiempo real, utilizando imágenes capturadas por una cámara conectada al dron. Este proceso se llevó a cabo mediante un nodo en ROS2 que procesa las imágenes y estima la pose de los marcadores con respecto a la cámara. El código completo se encuentra en los anexos del documento. A continuación, se describe el paso a paso del sistema:

1. **Publicación de imágenes desde el dron:** La computadora auxiliar en el dron (companion computer) publica imágenes comprimidas capturadas por la cámara a un tópico de ROS 2 (`/camera_image/compressed`). Estas imágenes se envían a la computadora en tierra para su procesamiento.

2. **Recepción de imágenes en la computadora en tierra:** Un nodo en la computadora en tierra se suscribe a las imágenes publicadas por el dron y realiza las siguientes tareas:

- Convierte las imágenes comprimidas en matrices utilizables para el procesamiento con OpenCV.
- Redimensiona las imágenes y las convierte a escala de grises para optimizar la detección.

3. **Detección de marcadores Aruco:** Se utiliza la biblioteca OpenCV para detectar los marcadores Aruco presentes en la imagen. Los pasos incluyen:

- Uso del diccionario `DICT_6X6_250` para identificar marcadores específicos.
- Cálculo de las esquinas de los marcadores detectados mediante `aruco.detectMarkers`.
- Dibujo de los marcadores detectados para su visualización.

Figure 3.8: Detección de marcadores Aruco en la imagen procesada.

4. **Estimación de la pose:** Para cada marcador detectado, se calculó su posición y orientación en el espacio con respecto a la cámara. Esto se realizó utilizando la función `aruco.estimatePoseSingleMarker` que devuelve:

- **tvec:** Vector de traslación (x, y, z) en metros.
- **rvec:** Vector de rotación que se convierte en ángulos de Euler (*roll, pitch, yaw*) mediante matrices de rotación.

Figure 3.9: Representación gráfica de la posición y orientación calculada de un marcador Aruco.

5. **Visualización y publicación de resultados:** Los resultados obtenidos, incluyendo la pose de los marcadores, se visualizaron en tiempo real y se publicaron en un nuevo tópico de ROS 2 (`/aruco_detection/compressed`). Las imágenes procesadas contenían:

- Marcadores detectados con esquinas resaltadas.
- Ejes X, Y y Z proyectados para mostrar la orientación de cada marcador.

Figure 3.10: Imagen con marcadores detectados y ejes proyectados.

6. **Correcciones de posición:** Basándose en los valores de **tvec**, se calcularon las correcciones necesarias para ajustar la posición del dron con respecto al marcador, incluyendo movimientos laterales (x), verticales (y) y de profundidad (z).
7. **Interacción con el sistema:** Los resultados de la detección de marcadores Aruco se utilizaron para visualizar la posición y orientación del drone en tiempo real, lo que permitirá posteriormente que el sistema de control pueda ajustar las mismas para su aterrizaje en la estación de carga.

3.5 Sistema de Comunicación

3.5.1 Especificaciones y Requerimientos

El sistema de comunicación entre el dron y la estación de carga fue diseñado para garantizar la transferencia eficiente y en tiempo real de datos críticos. Los requisitos principales son:

- **Red local basada en WiFi:** Proporcionar una comunicación efectiva entre los dispositivos sin necesidad de acceso a Internet.
- **Middleware DDS de ROS 2:** Utilizar la arquitectura de nodos y tópicos de ROS 2 para gestionar la comunicación de datos entre los dispositivos conectados.

Figure 3.11: Diagrama de la red local WiFi utilizada en el sistema.

Figure 3.12: Diagrama de comunicación entre los dispositivos utilizando ROS 2.

3.5.2 Estructura de Comunicación

Red Local WiFi

La red local fue creada utilizando un router WiFi que conecta todos los dispositivos involucrados en el sistema, incluyendo el dron, la estación de carga y la computadora en tierra. Esta configuración eliminó la necesidad de Internet, proporcionando un entorno seguro y aislado para la transmisión de datos. El dron, equipado con una computadora auxiliar, transmitió datos mediante tópicos de ROS 2, como imágenes comprimidas y mensajes de estado, hacia la estación de carga y la computadora en tierra.

Comunicación mediante ROS 2

El sistema utilizó la arquitectura de nodos y tópicos de ROS 2 para gestionar la comunicación entre los dispositivos. Cada dispositivo en la red cumplió funciones específicas relacionadas con la publicación y suscripción de datos relevantes:

- **Dron:** Publicó imágenes de la cámara (`/camera_image/compressed`) y datos de estado relacionados con la batería, posición y otros parámetros críticos.
- **Computadora en tierra:** Contiene una interfaz que se suscribió a los tópicos publicados por el dron para:
 - Procesar imágenes y calcular la posición de los marcadores Aruco.
 - Mostrar datos de estado del dron, como nivel de batería y modo de vuelo, etc. en tiempo real.
 - Publicar comandos hacia la estación de carga, como abrir o retraer el cajón de la estación, mediante tópicos específicos.
- **Estación de carga:** Se suscribió a los comandos enviados desde la interfaz de la computadora en tierra para ejecutar las acciones necesarias, como la apertura y cierre del cajón.

Chapter 4

Resultados

Este capítulo presenta los resultados obtenidos en el desarrollo y prueba de la base de carga modular y el dron instrumentado para la interacción con la estación de carga.

4.1 Resultados del Diseño y Manufactura de la Base de Carga

4.1.1 Diseño CAD Final del 1er prototipo de la Estación de Carga

4.1.2 Diseño e Instrumentación del Drone

4.1.3 Movimiento del Cajón

4.1.4 Vuelo del Drone

4.1.5 Carga y Descarga de Batería

4.1.6 Detección del ArUco

4.1.7 Sistema de Comunicación

Documentación de los tiempos de respuesta, confiabilidad y estabilidad de la comunicación. Las métricas de latencia y tasa de éxito en la transferencia de datos se presentan en gráficos para facilitar el análisis.

4.2 Análisis de Desempeño General

4.2.1 Resultados de Integración Global

Comparación entre el desempeño esperado y el desempeño real en la interacción entre el dron y la estación de carga.

Chapter 5

Conclusiones

Texto de introduccion: Estabamos buscando hacer esto y llegamos a esto

Evaluacion de resultados

Posibles mejoras

Bibliography

- [1] PX4 Documentation, “PX4 is a powerful open source autopilot flight stack running on the NuttX RTOS.” Disponible en: . [*Último acceso: mes, año*].
- [2] ArduPilot Documentation, “ArduPilot firmware works on a wide variety of different hardware to control unmanned vehicles of all types.” Disponible en: <https://ardupilot.org/ardupilot/index.html>. [*Último acceso: mes, año*].
- [3] QGroundControl Documentation, “QGroundControl provides full flight control and vehicle setup for PX4 or ArduPilot powered vehicles.” Disponible en: <https://docs.qgroundcontrol.com/en/>. [*Último acceso: mes, año*].
- [4] Ubuntu Documentation, “Designed for security, reliability, and ease of use.” Disponible en: <https://ubuntu.com>. [*Último acceso: mes, año*].
- [5] Raspberry Pi OS Documentation, “A Debian-based operating system specifically tuned for the Raspberry Pi hardware.” Disponible en: <https://www.raspberrypi.org/documentation/raspbian/>. [*Último acceso: mes, año*].
- [6] ROS 2 Documentation, “Benefits and architecture of ROS 2 for distributed systems in robotics.” Disponible en: <https://docs.ros.org/en/>. [*Último acceso: mes, año*].
- [7] OpenCV Documentation, “An open-source computer vision and machine learning software library containing more than 2500 optimized algorithms.” Disponible en: <https://docs.opencv.org/>. [*Último acceso: mes, año*].
- [8] ArUco Documentation, “ArUco is an OpenSource library for detecting squared fiducial markers in images.” Disponible en: . [*Último acceso: mes, año*].
- [9] Khazetdinov, M., et al., “Embedded ArUco Markers for UAV Landing: High Accuracy Detection in Wide Distance Range,” 2021. DOI: [DOI number or URL if available].