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

UDEM

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 State of Art . . . 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 Drone Applications . . . 10
  - 2.2.6 Swarming Drones . . . 11
  - 2.2.7 Types of Drone Charging Stations . . . 11
  - 2.2.8 Types of Mechanisms . . . 15
- 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 . . . 34
2.4.11 Sensors for Navigation in Outdoor and Indoor Environments . . . 36

3 Desarrollo **38**
3.1 Diseño y Desarrollo de la Estación de Carga . . . 38
3.1.1 Requerimientos . . . 38
3.1.2 Lista de Materiales . . . 38
3.1.3 Diseño CAD de la Base de Carga . . . 40
3.1.4 Selección e Implementación del Mecanismo de Movimiento del Cajón . . . 42
3.1.5 Proceso de Manufactura de la Estructura de la Base de Carga . . . 46
3.1.6 Electronic Circuit of the Charging Station . . . 47
3.2 Instrumentación del Dron . . . 49
3.2.1 Especificaciones y Requerimientos . . . 49
3.2.2 Lista de Materiales para la Instrumentación del Dron . . . 49
3.2.3 Diseños CAD del Dron . . . 50
3.2.4 Circuito de Distribución de Energía . . . 50
3.3 Software Configuration . . . 51
3.3.1 Configuración de la Computadora Central en Tierra . . . 51
3.3.2 Configuración de la Computadora Auxiliar del Dron . . . 52
3.3.3 Configuración de la Estación de Control en Tierra . . . 55
3.4 Sistema de Visión . . . 55
3.4.1 Especificaciones y Requerimientos . . . 55
3.4.2 Calibración de la Cámara . . . 56
3.4.3 Generación de Marcadores ArUco . . . 57
3.4.4 Detección de Marcadores e-Aruco en Tiempo Real . . . 59
3.5 Sistema de Comunicación . . . 60
3.5.1 Especificaciones y Requerimientos . . . 60
3.5.2 Estructura de Comunicación . . . 60

4 Resultados **67**
4.1 Resultados del Diseño y Manufactura de la Base de Carga . . . 67
4.1.1 Diseño CAD Final del 1er protopipo de la Estación de Carga . . . 67
4.1.2 Diseño e Instrumentación del Drone . . . 67
4.1.3 Movimiento del Cajón . . . 67
4.1.4 Vuelo del Drone . . . 67
4.1.5 Carga y Descarga de Batería . . . 67
4.1.6 Detección del ArUco . . . 67
4.1.7 Sistema de Comunicación . . . 67
4.2 Análisis de Desempeño General . . . 68
4.2.1 Resultados de Integración Global . . . 68

5 Conclusiones **69**

6 Bibliography **70**

Chapter 1

Introducción

1.1 Problem Statement

1.1.1 Clear Problem Definition

One of the primary challenges in achieving continuous and autonomous operation of drone swarms lies in the lack of modular and autonomous charging stations capable of simultaneously recharging multiple drones. Current market solutions present significant limitations, including the inability to efficiently manage the concurrent charging of several drones and ensure precise landing in shared spaces without human intervention. These shortcomings not only reduce drone autonomy but also hinder operational efficiency, as they necessitate frequent manual interventions for maneuvering drones. Additionally, the absence of modular solutions in existing market offerings results in increased space requirements, further complicating implementation.

1.1.2 Research Questions

- How can a modular and stackable charging station be developed to enable simultaneous charging of multiple drones?
- What localization and vision technologies can be implemented to ensure precise landing in a specific setting?
- Which charging method is the most reliable in terms of fast, independent charging for a drone?

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

- To design a CAD model for the charging station, ensuring compatibility with open-architecture quadcopters.
- To manufacture a charging station module for the drone swarm.
- To adapt the current design of the open-architecture drone to the charging station.
- To equip a quadcopter with an onboard computer, flight controller, vision camera, and localization sensors.
- To develop a communication system between the charging station and the quadcopter for the transfer of relevant interaction data.
- To program a system to control the drone from an onboard computer.
- To 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 pd2resent 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 State of Art

The purpose of this chapter is to provide a clear and detailed overview of previous work and research in the field related to drone charging. A review of the literature, existing technologies, and identified limitations that justify the need for the proposed research is presented.

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 [22], "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 [22], "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) is the aircraft, which is able to fly without a pilot and passengers on board" [21]. 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 3.21a 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, surveillance, and light tasks.

Figure 2.1: Multicopter.

Figure 3.21b 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.

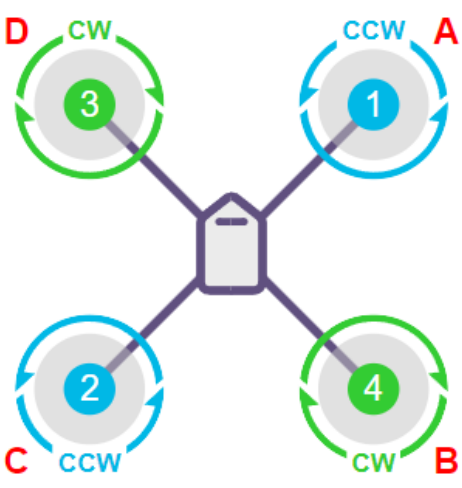
In Figure 3.21c, the Octo Frame configuration is shown, featuring eight motors and propellers. Drones with this configuration are ideal for industrial applications due to their higher payload capacity and redundancy, which improves safety in case of motor or propeller failures.

Finally, Figure 3.21d 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.

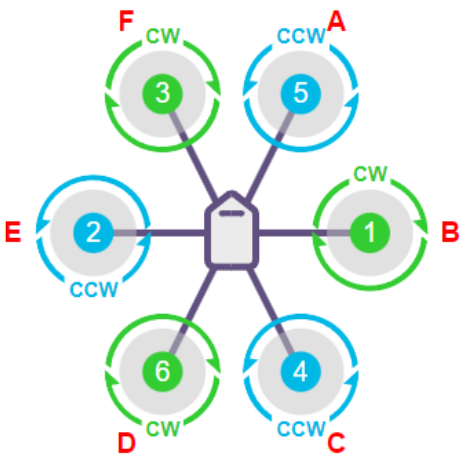
2.2.2 Motors

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:

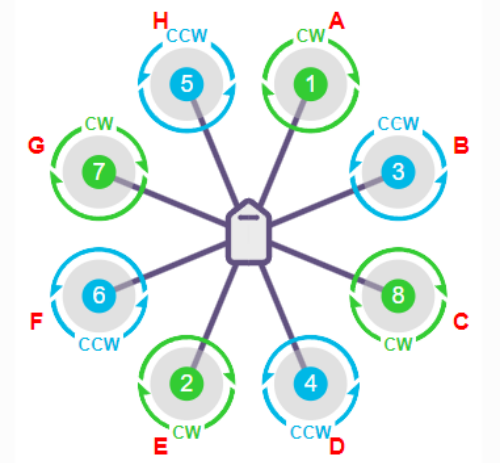
- **Kv Rating:** Indicates the revolutions per minute (RPM) per applied volt without load. A lower Kv provides more torque, making it ideal for larger drones with bigger propellers.
- **Compatibility:** The motor must be compatible with the electronic speed controller (ESC) and the propellers.
- **Mounting:**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



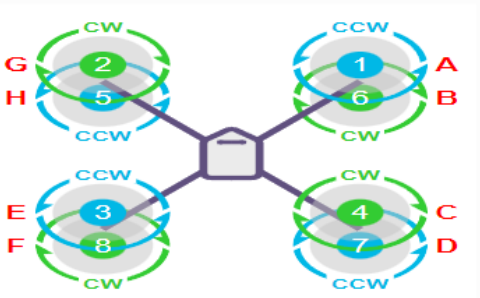
(a) Quad X Frame.



(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: This image shows the DJI AGRAS T50 being applied in the agricultural industry. [17]

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 Drone Applications

Drones have a wide variety of applications nowadays and as time goes by these applications are increasing thanks to their versatility and ability to access difficult to reach areas. According to the official documentation [ardupailot] the applications of drones are [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: An Amazon drone is shown, which makes home package deliveries. [17]

2.2.6 Swarming Drones

Unlike the individual control of a drone, the drone swarm concept allows collaboration between multiple units to execute complex tasks in an efficient and coordinated manner. This technology is called drone swarm, which according to the authors [4] represents "the true revolution in this field is the control of drone swarm, enabling multiple units to collaborate in executing more complex tasks".

The control of drone swarms requires sophisticated systems that include bidirectional communication, both between the drones and the Ground Control Station (GCS). According to the authors [4], these systems allow drones to share key information such as position, battery status, and obstacles detected, making it possible to avoid collisions, maintain specific formations, and adapt to dynamic environmental conditions.

In addition, the authors [4] emphasize that a fundamental element for the success of this technology is the capacity for precise positioning and localization, achieved through advanced GPS systems. This precision allows drones to maintain safe distances from each other and operate autonomously in complex missions. Another relevant aspect is the implementation of redundancy systems that guarantee the continuity of operations in case of failure in any swarm unit.

In the industrial domain, drone swarm present great potential in applications such as automated inspections, environmental monitoring, precision agriculture and disaster management. According to the authors [4], the use of drone swarm in these scenarios not only improves operational efficiency, but also opens new opportunities for the development of autonomous technologies in the context of Industry 4.0.

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

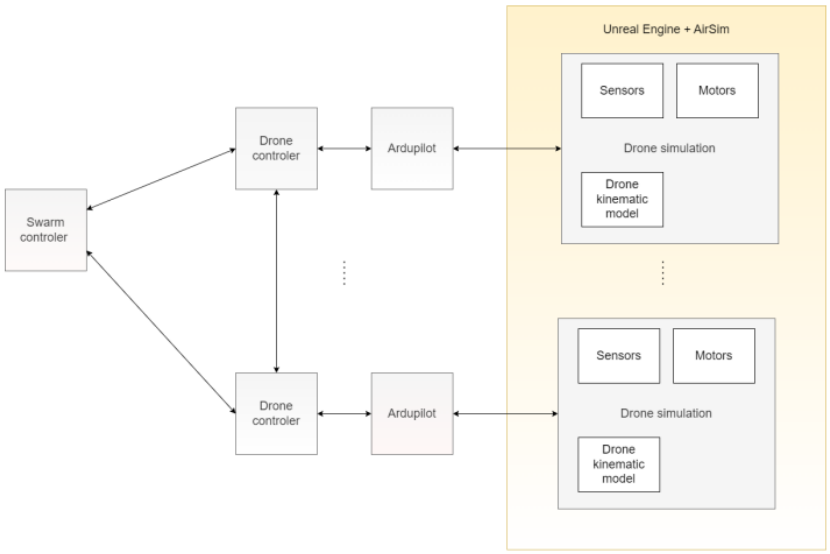


Figure 2.5: Diagram simulating a drone swarm model

Figure 2.6: An example of drone swarm is shown in the image. [19]

to land safely, take off, recharge and/or replace batteries, and transfer data and payload" [16].

In this context, various types of charging stations can be observed, classified according to the following specifications [16]:

- 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)

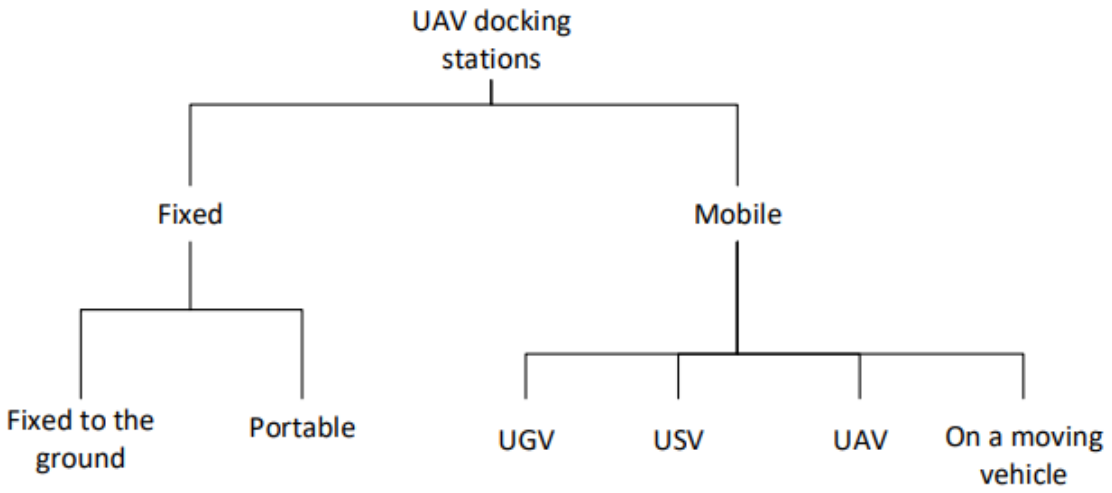


Figure 2.7: Drone Charging Stations Classification [16].

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 [16].

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.

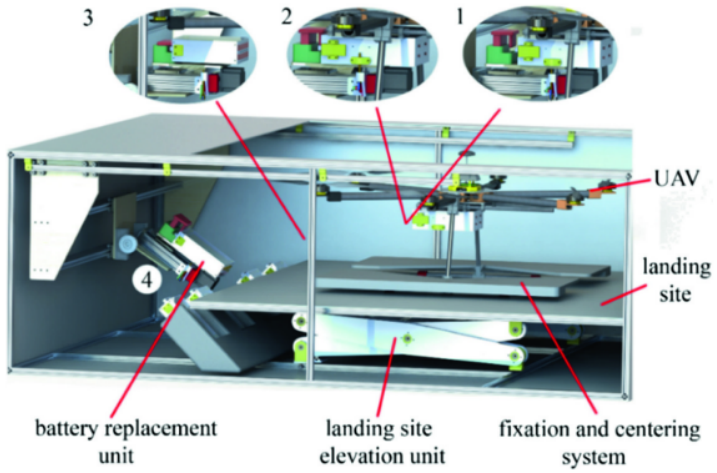


Figure 2.8: Fixed docking station with its subsystems. The labels (1–4) show the battery replacement mechanism in operation [16].

Figure 2.9: Battery replacement docking station [16].

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 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 [16].

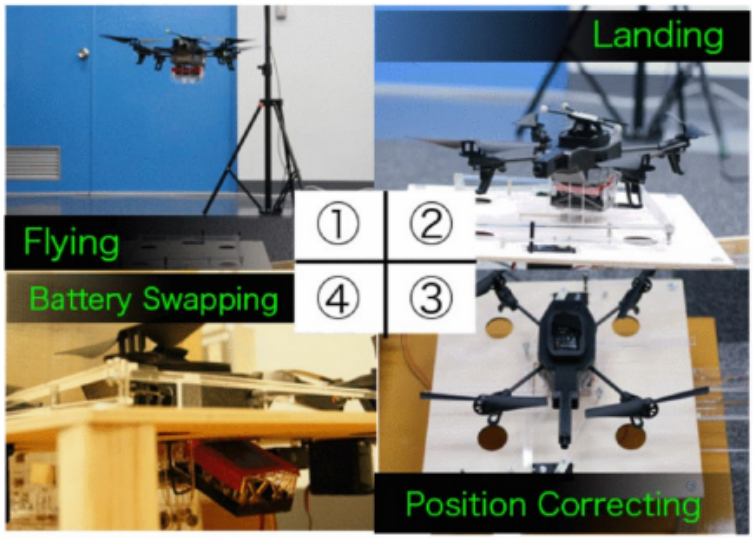


Figure 2.10: Endless Flyer [16], docking station with battery replacement mechanism. The labels in the center show the different landing stages and the order in which they are realized.

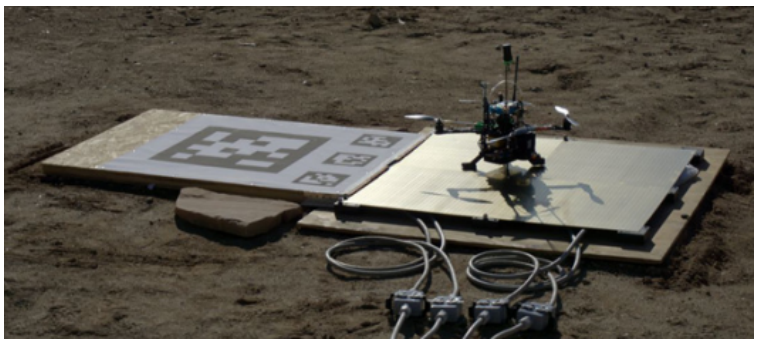


Figure 2.11: Docking station with recharging pad with the custom made UAV from [16]. In the figure, the QR codes used in the approach and landing phase can be seen .

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.

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

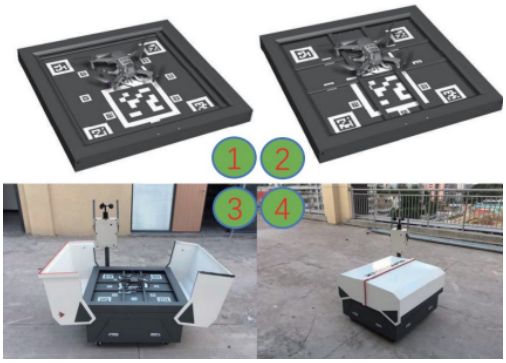


Figure 2.12: Moving docking station with storage system and battery replacement system. Labels in the figure (1–4) show the order in which the active positioning mechanism and storing mechanism take place [16].

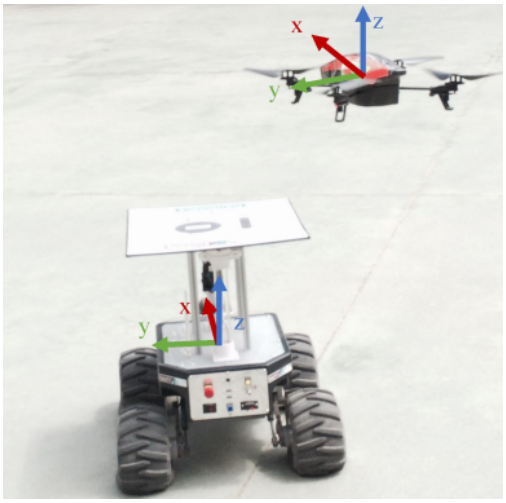


Figure 2.13: UGV–UAV proposed moving docking station solution [16].

Figure 2.14: USV–UAV landing system [16].

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 The pistons used 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 ro-

bust 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" [32]. 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" [33]. 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 [34].

- **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 [12].
- **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 [13].
- **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 [14].

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 [29].

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. [35]

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. [30]

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. [31]

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. [28]

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.

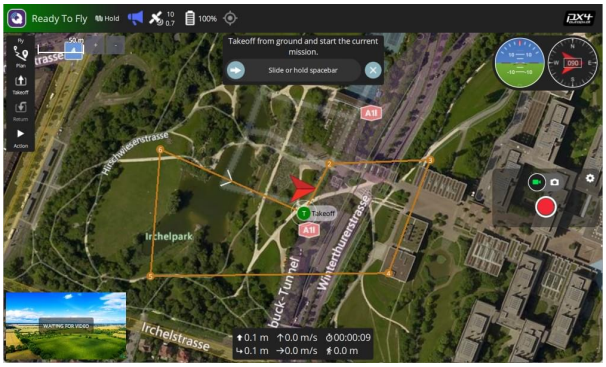
Despite their usefulness during the preparation and testing stages, the proposed system does not directly rely on these tools for its final operation. The autonomous operation of the drone and the charging station is based on direct interaction between the firmware, vision systems, and flight controllers, 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 [36].

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 [37].

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

Figure 2.17: ROS 2 Jazzy Jalisco

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.
- **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.

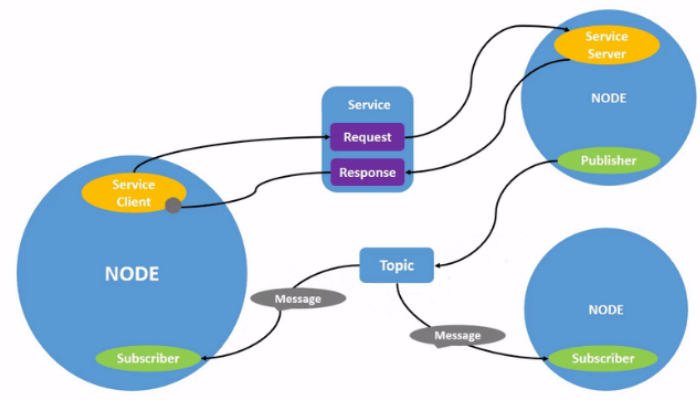


Figure 2.18: ROS 2 Architecture [6].

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"* [38].

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.

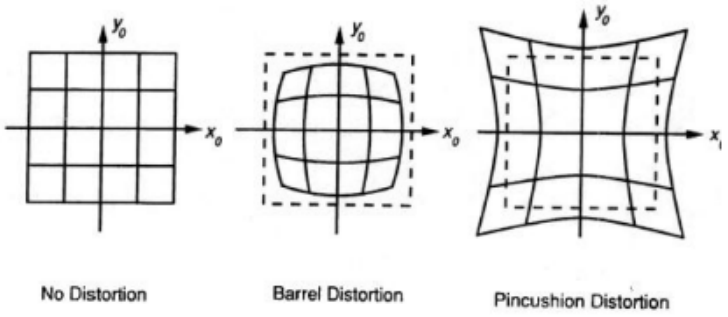


Figure 2.19: Types of camera distortions.

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.

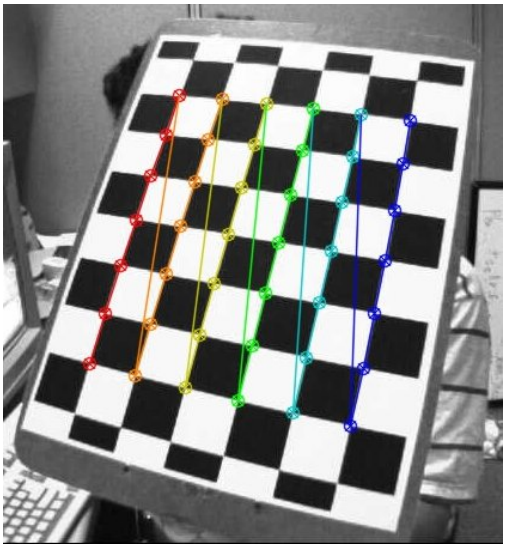


Figure 2.20: Chessboard pattern used in camera calibration with OpenCV.

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].

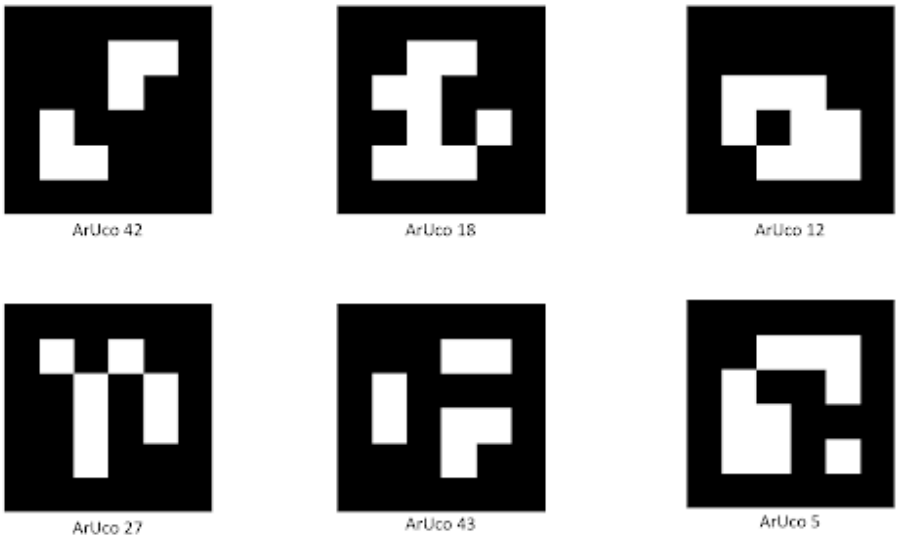


Figure 2.21: Examples of ArUco markers and IDs.

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*

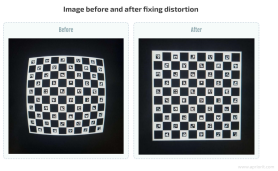


Figure 2.22: Camera Calibration with ArUco

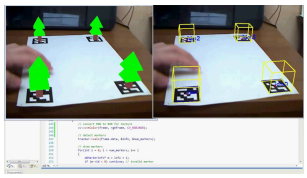


Figure 2.23: Augmented Reality with ArUco

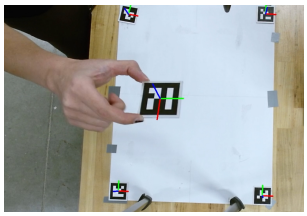


Figure 2.24: Real-Time Tracking with ArUco

Figure 2.25: Common applications of ArUco markers.

pose estimation" [39]. 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"* [39]. This flexibility is especially useful in projects where ensuring the uniqueness and reliability of marker detection in complex environments is crucial.

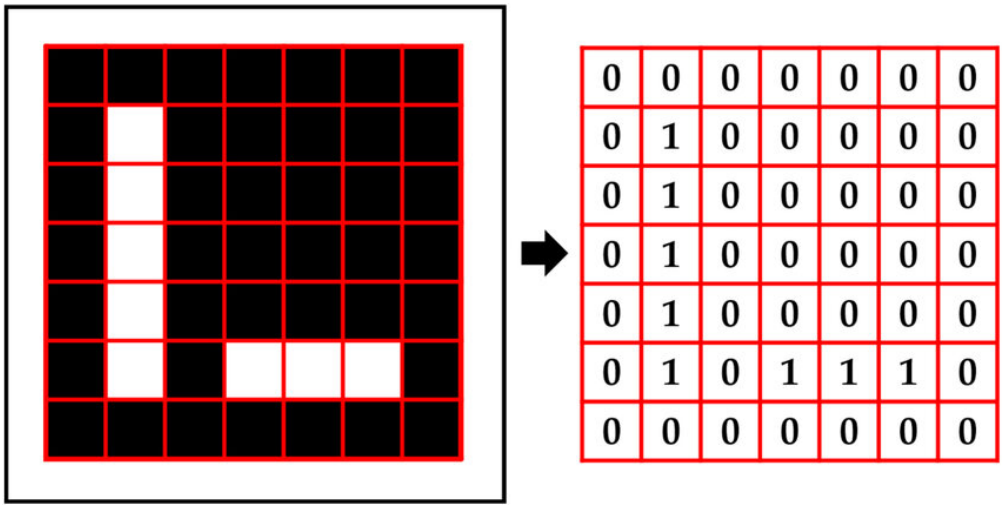


Figure 2.26: Bits of a 7x7 ArUco marker.

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.

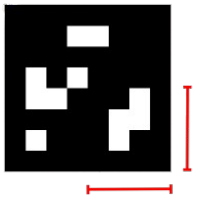


Figure 2.28: Calculating the Large ArUco Marker Center.

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 [42]. 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"* [43].

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 [42]. 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.

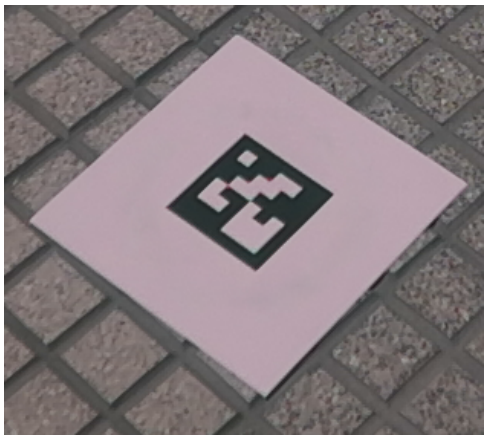


Figure 2.31: ArUco without pose estimation.

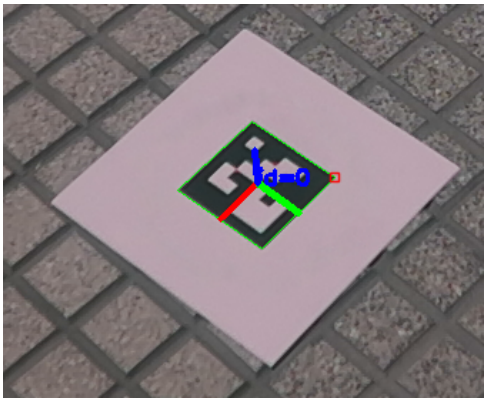


Figure 2.32: ArUco with pose estimation.

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"* [40]. 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.

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 [41].

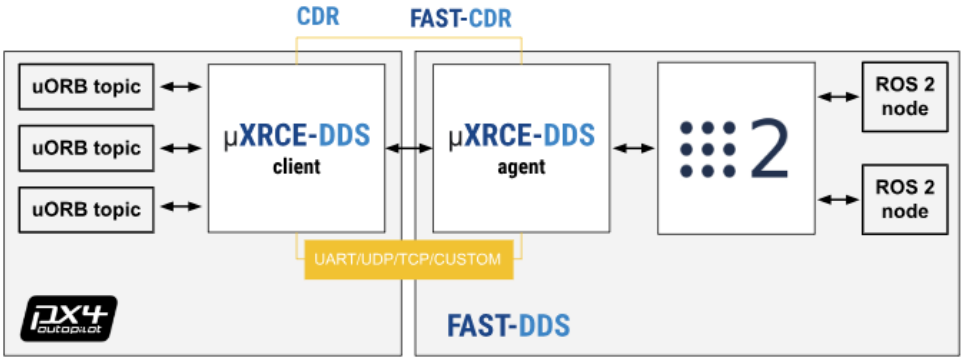


Figure 2.33: ROS 2 - Pixhawk communication using Micro XRCE-DDS.

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 [41].

2.4.11 Sensors for Navigation in Outdoor and Indoor Environments

To achieve precise control of the drone's position using ROS 2 topics such as Òtexture/mavros/setpoint/position/loc it is necessary to have sensors that provide reliable positioning data depending on the environment in which the drone is operating. Sensors commonly used in outdoor and indoor environments to provide position information are described below.

Outdoor Environment Sensors

In outdoor environments, drones often use global positioning sensors (GPS) to determine their location in real time. The following are the most common outdoor sensors:

- **GPS (Global Positioning System):** GPS is the primary sensor for outdoor navigation. It provides latitude, longitude, and altitude coordinates that allow the drone to determine its global position. In ROS 2, this data is usually published in topics such as `/mavros/global_position/global` and integrated into the position control systems [10].
- **RTK-GPS (Real-Time Kinematic GPS):** For high-precision applications, such as precision landing or navigation in restricted areas, RTK-GPS is used. This system improves the accuracy of GPS by correcting data in real time, reaching centimeter accuracies. RTK-GPS is compatible with MAVLink and enables highly accurate position control in ROS 2 [10].

- **IMU (Inertial Measurement Unit):** Besides it does not provide direct location data, the IMU complements GPS by detecting changes in orientation and acceleration. This data is crucial for maintaining stability and providing information about the drone's orientation during flight [10].

Sensors for Indoor Environments

In indoor environments, where GPS signals are often weak or non-existent, other sensors are used to determine the drone's position and navigation. Sensors used indoors include:

- **Vision Cameras (Monocular or Stereo):** Vision cameras allow visual localization using SLAM (Simultaneous Localization and Mapping) algorithms or marker detection, such as ArUco, to calculate the relative position of the drone indoors. With ROS 2, these cameras can be integrated with libraries such as OpenCV and use the image topics for real-time processing [11].
- **LIDAR (Light Detection and Ranging):** LIDAR sensors emit pulses of light to measure distances and obtain detailed maps of the environment. These sensors are useful for real-time indoor navigation and mapping and are integrated into ROS 2 using point cloud topics. They are particularly effective for obstacle avoidance and precise indoor positioning [11].
- **Vicon Motion Capture System:** Vicon cameras are a high-precision motion capture system that utilize multiple cameras placed around a testing environment to track reflective markers attached to the drone. By triangulating the position of these markers, the Vicon system provides accurate 3D position and orientation data. This data can be used in ROS 2 for real-time drone control and navigation, typically by publishing the position data as a topic. The Vicon system is ideal for precise indoor positioning, particularly in controlled environments where high accuracy is required [11].
- **IMU (Inertial Measurement Unit):** Indoors, the IMU remains an essential component for detecting changes in orientation and motion. Fusion of IMU data with other sensors, such as LIDAR or cameras, helps maintain stability and provides accurate position estimates using fusion filters, such as the Kalman filter [11].

Chapter 3

Desarrollo

This chapter describes the process of design, implementation and configuration of the modular charging station and drone instrumentation. It details the methodologies and tools used to carry out the development, with a focus on creating an effective solution for the charging system.

3.1 Diseño y Desarrollo de la Estación de Carga

3.1.1 Requerimientos

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

1. El sistema deberá contar con un diseño que sea modular y apilable para la integración de multiples estaciones de carga.
2. El sistema deberá tomar en cuenta y adaptarse a las dimensiones del dron propuesto para la aplicación.
3. El sistema deberá de contar con un mecanismo automático de apertura y cierre para el almacenamiento del dron.
4. El sistema deberá contar con un mecanismo de carga para la batería del dron.
5. El sistema deberá contener método de posicionamiento por visión para el dron.

3.1.2 Lista de Materiales

A continuación, se presenta la lista de materiales utilizados para la construcción de la estación de carga modular. Cabe destacar que el diseño y desarrollo de este proyecto se basaron en gran medida en los recursos y materiales disponibles previamente, lo que permitió optimizar costos y aprovechar de manera eficiente los insumos existentes.

- 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 Corredera Telescópica
- 1 x Motor de 12V
- 1 x Soportes de Motor impresos en 3D
- 1 x A V-Belts 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
- 1 X Eje de (falta poner dimensiones)

3.1.3 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:

Como se muestra en la Fig 1. Se diseñó la base conocida como "cajón interno", que sirve como plataforma para el despegue y aterrizaje del dron. Este diseño se realizó considerando las dimensiones del dron y asegurando la inclusión de tolerancias adecuadas para prevenir posibles colisiones. Se utilizó perfil de aluminio de 30 mm en este diseño aprovechando que es material con el que se contaba previamente. También se le integró otros perfiles por el medio del cajón y de manera horizontal, lo que sería el soporte de los rieles.

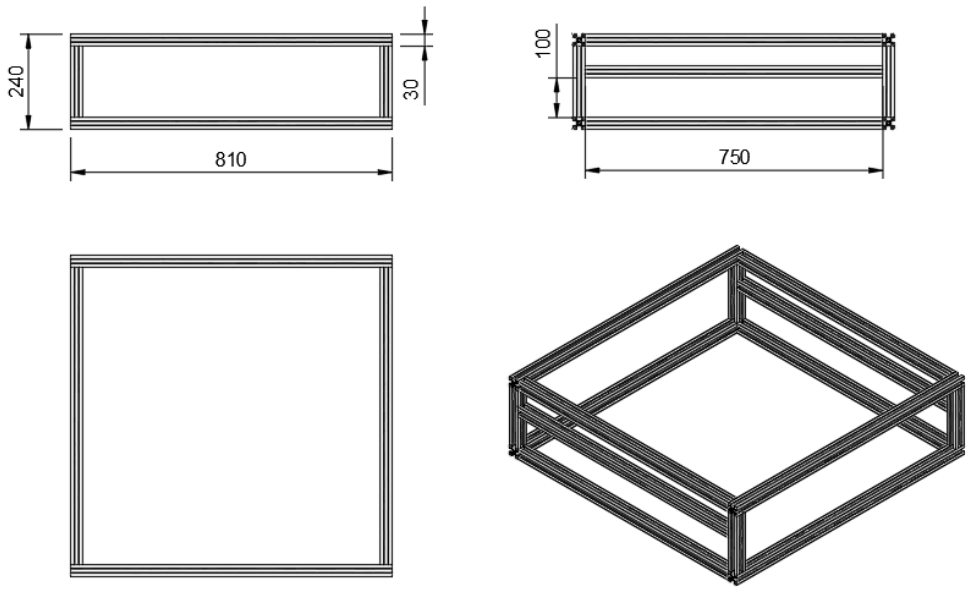


Figure 3.1: Plano del cajón interno de la Estación de Carga

Por consiguiente, tomando en cuenta las dimensiones del plano anterior, se realizó lo que es el "cajón externo" en el cual sirve para almacenar el dron y la electrónica de la estación de carga. Esto se puede observar en la Fig 2.

Después, se tomaron en cuenta las dimensiones de las landing gear (patas del dron) para imprimir en 3D unas guías mecánicas (Fig 3.) que ayuden a estas mismas a hacer una correcta conexión magnética con el sistema de carga.

Tomando en cuenta los diseños de la Fig. 1 y la Fig. 3 se desarrolló por completo el diseño de el cajón interno (Fig. 4), cuenta con dos cortes de MDF de 9 mm de 810 mm x 810 mm y separadores internos con espacio de 20 mm de altura para proteger los cables del sistema de carga.

Se desarrollaron los diseños CAD necesarios como se muestran en la Fig 5. y Fig 6. para garantizar la correcta integración y adaptación de los distintos componentes de la estación de carga. Estos diseños permitieron establecer las dimensiones, tolerancias y características esenciales para que cada elemento se ajustara de manera precisa, asegurando que la estación de carga cumpliera

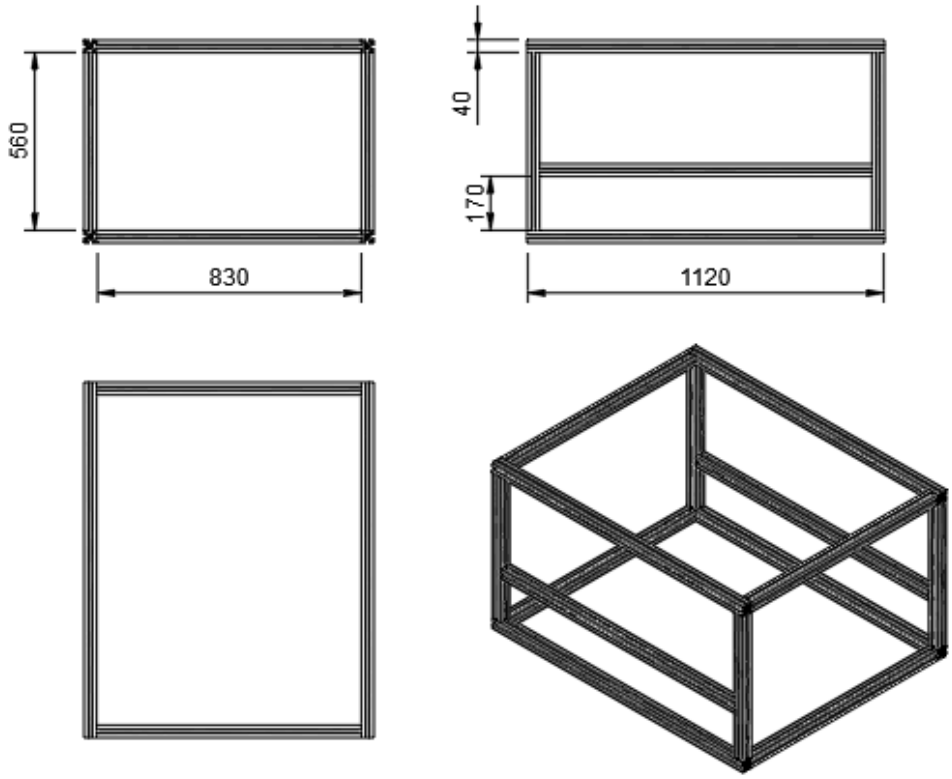


Figure 3.2: Plano del cajón externo de la Estación de Carga

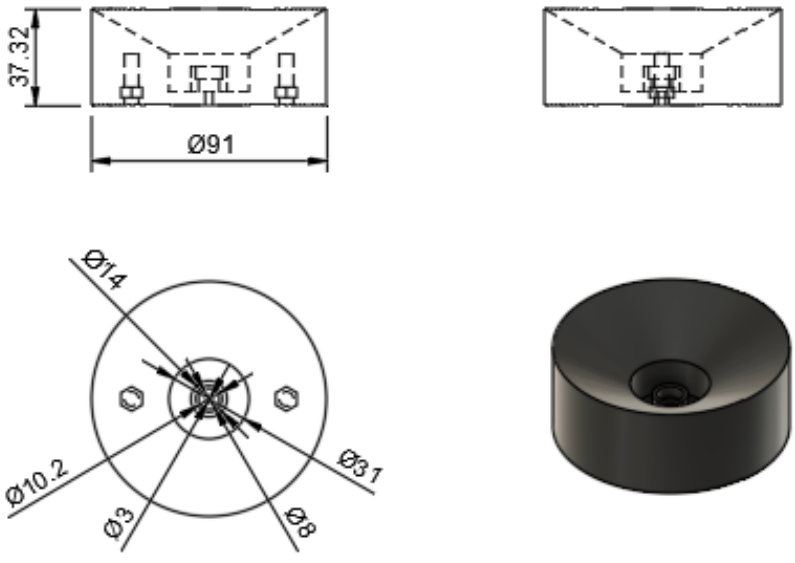


Figure 3.3: Plano de guía mecánica para la Estación de Carga

Figure 3.4: Vista del cajón interno de la Estación de Carga.

con su funcionalidad de forma eficiente y confiable.

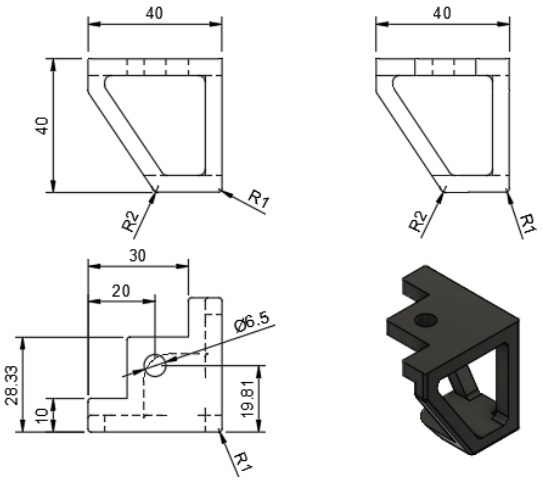


Figure 3.5: Soportes de piso para la estabilidad de la Estación de Carga.

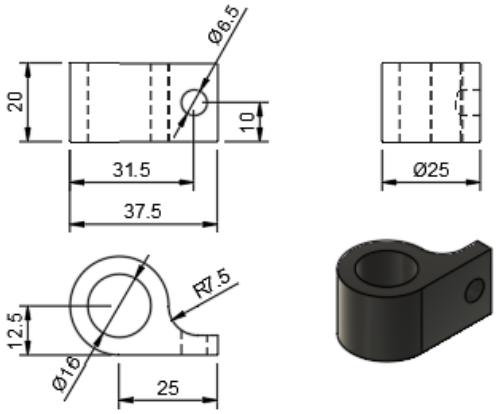


Figure 3.6: Sujetador impreso en 3D para los sensores de proximidad inductivos de la Estación de Carga.

Por último del proceso de diseño, se desarrolló un modelo CAD que integra los componentes principales necesarios para el correcto funcionamiento de la estación de carga. Este diseño conceptual final, mostrado en la figura 7, representa la consolidación de las ideas iniciales, las iteraciones realizadas y las decisiones tomadas a lo largo del desarrollo. El modelo asegura la funcionalidad, compatibilidad y eficiencia de la estación, sirviendo como base para la fabricación e implementación del sistema.

3.1.4 Selección e Implementación del Mecanismo de Movimiento del Cajón

El diseño del mecanismo de cajón automático para la estación de carga pasó por varias etapas de conceptualización, considerando diferentes opciones para garantizar un movimiento eficiente y económico. Inicialmente, se planteó utilizar una banda dentada similar al mecanismo de las impresoras 3D, permitiendo un desplazamiento preciso hacia adelante y hacia atrás; sin embargo, esta solución resultaba costosa y compleja debido a las dimensiones requeridas. Como alternativa, se evaluó el uso de pistones, pero estos incrementaban aún más los costos del proyecto. Finalmente, se optó por implementar un sistema de banda y polea tipo V, clasificación A, debido a su rápida disponibilidad y considerable reducción en costos, logrando así un equilibrio entre funcionalidad, simplicidad y economía.

Componentes y Diseños CAD

1. Motor de 12V

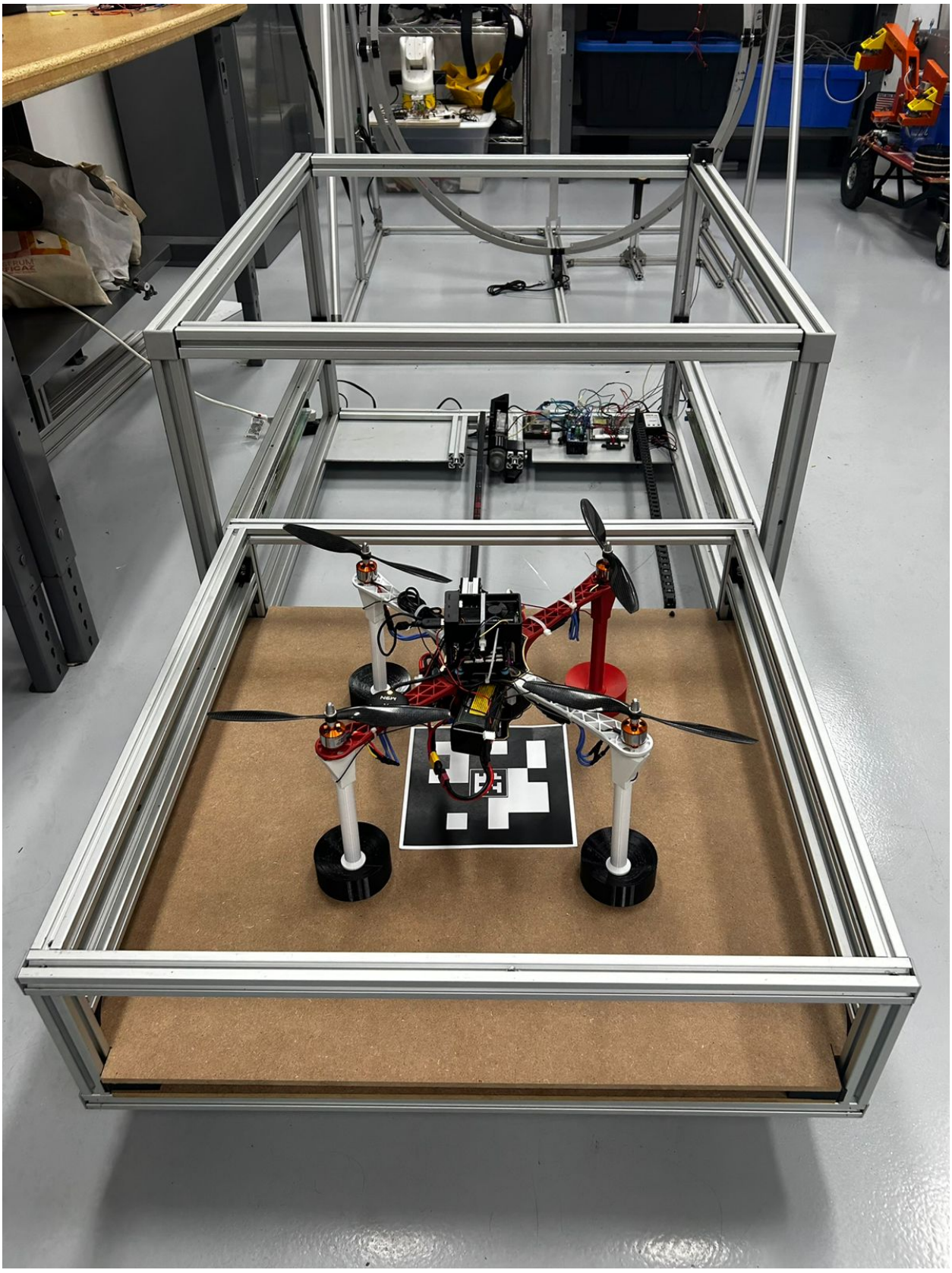


Figure 3.7: Vista de la Estación de Carga.

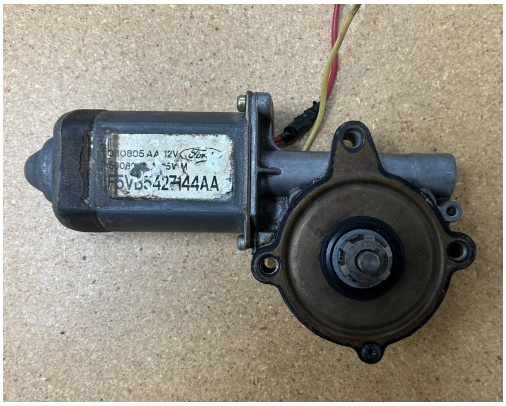


Figure 3.8: Motor de 12 Volts utilizado para la Estación de Carga.

2. Poleas

Figure 3.9: Polea de aluminio con diametro interno de 1/2”

3. Banda V Tipo A

Figure 3.10: V-Belt Type

4. Baleros
5. Chumaceras
6. Sujetadores de Motor 12 V
7. Sujetador de Banda V para cajón interno.

Figure 3.11: Baleros de..

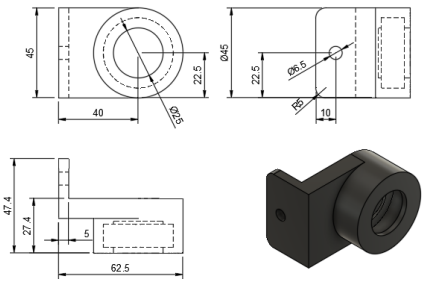


Figure 3.12: Plano de chumaceras para el mecanismo de movimiento del cajón interno.

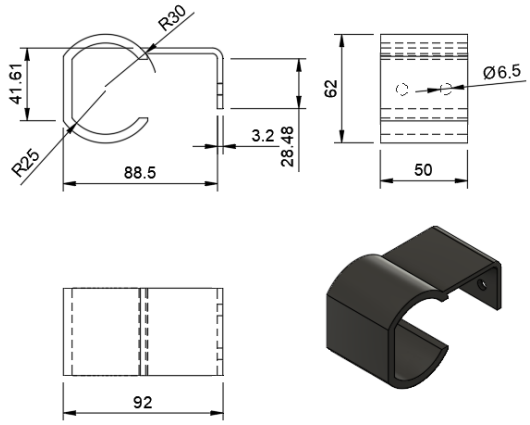


Figure 3.13

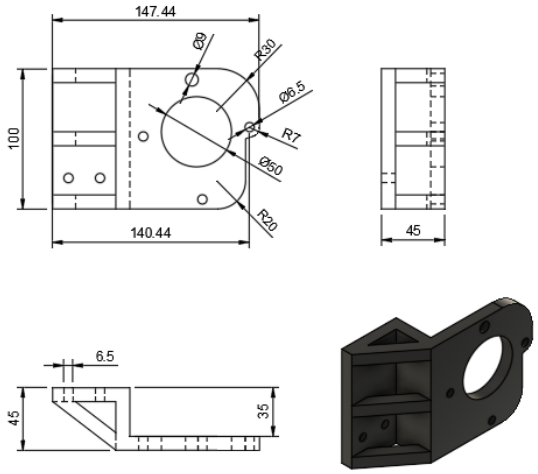


Figure 3.14

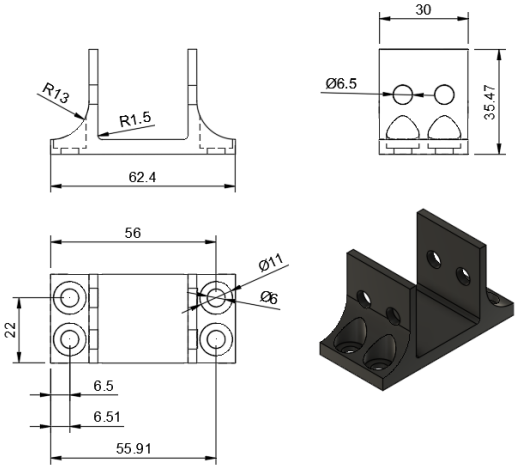


Figure 3.15

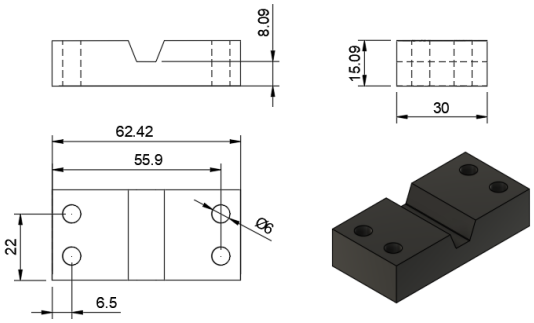


Figure 3.16

Mecanismo Polea y Banda tipo V

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. Se puede observar este preoceso en la Figura ? a)

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 separadores de 20 mm impresos en 3D. Se verificó que todas las piezas estuvieran alineadas y niveladas para garantizar la estabilidad y resistencia de la base de carga. Esto se muestra en las Figura ? b) y c).

3. **Instalación del Mecanismo de Movimiento:** Después 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. Figura ? d).

3.1.6 Electronic Circuit of the Charging Station

Conection Diagram

add diagram circuit

Circuit Construction

add images

Circuit Programming

General Description: The described circuit uses a joystick to control a motor that moves a drawer back and forth. The joystick is connected to a microcontroller (Arduino Mega 2560), which reads the joystick's Y-axis values to determine the direction in which the motor should move. Depending on the position of the joystick, the motor moves forward, backward or stops. In addition, limit switches are used to ensure that the motor stops when the drawer reaches its extreme positions.

Main Components

- **Arduino:** Microcontroller that manages the reading of the joystick values and controls the motor.
- **Joystick:** Analog input device that allows the user to control the direction of movement.
- **Motor and H-Bridge:** Driver system that moves the drawer. The H-bridge allows controlling the direction and speed of motion of the motor.
- **Limit Switches:** Sensors that detect when the drawer has reached its extreme positions (forward and reverse).
- **Power Supply:** Supplies power to the motor. The Arduino and RaspberryPi4 are supplied power by a plug-in charger.

Arduino Code The Arduino code performs the following functions:

1. **Pin initialization:**
   - The joystick pins (A1 for the Y-axis) are configured as inputs.
   - Motor pins 9 and 8 are configured as outputs.
   - The limit switch pins (10 and 11) are configured as inputs with internal pull-up resistors.
2. **Reading of Joystick:**
   - At each loop cycle, the Arduino reads the analog value from the joystick's Y-axis.
   - Depending on the value read, the Arduino decides whether the motor should move forward, backward or stop.
3. **Motor Control:**
   - If the value of the joystick Y-axis is greater than 512 (middle position) and the forward limit switch is not pressed, the motor moves forward.
   - If the Y-axis value is less than 512 and the backward limit switch is not pressed, the motor moves backward.
   - If the Y-axis value is approximately 512 or any limit switch is pressed, the motor stops.
4. **Limit Switch Verification:**
   - If the motor is moving forward and the forward limit detects metal (the drawer), the motor stops.
   - If the motor is moving backward and the backward limit switch detects metal (the drawer), the motor stops.

Circuit Operation

- Initialization: At startup, the Arduino configures the pins and establishes serial communication for debugging.
- Continuous Readout: In the main loop, the Arduino continuously reads the value of the joystick Y-axis.
- Dynamic Control: Based on the value read, the Arduino controls the motor to move the drawer forward, backward or stop it.
- Safety Check: Limit switches ensure that the motor automatically stops when the drawer reaches its extreme positions, protecting the system from possible damage.
- Debugging: Joystick values and motor status are printed on the serial monitor for easy debugging and system tuning.

Annex The complete source code for both the Arduino and Raspberry Pi 4 is included in the appendices of this document, making it easy to review and modify as needed for implementation and control of the system.

This approach ensures that the drawer moves in a precise and controlled manner based on the joystick position, providing an efficient solution for linear motion control using a motor and belt.

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 fue diseñado para proporcionar una alimentación segura y estable a los componentes críticos del dron, incluyendo los motores, el controlador de vuelo, la Raspberry Pi y la cámara. A continuación, se describe el funcionamiento de cada sección del circuito:

1. **Batería LiPo 3S:** Se seleccionó una batería de litio-polímero (LiPo) de 3 celdas con una capacidad de 5200 mAh, un voltaje nominal de 11.1 V y una tasa de descarga de 50C. Esta batería es ideal para proporcionar la corriente necesaria para los motores y los componentes electrónicos sin comprometer la duración de vuelo.
2. **Distribución de energía a los motores:** La energía de la batería se distribuye a través de una placa de distribución de energía, que conecta la batería a los cuatro controladores electrónicos de velocidad (ESC). Cada ESC regula la energía enviada a su respectivo motor A2212 10T, permitiendo un control preciso de la velocidad de los motores.
3. **Regulador de voltaje XL4005 DC-DC:** Este regulador convierte el voltaje de 11.1 V de la batería a 5 V, proporcionando una alimentación estable y segura para la Raspberry Pi y la cámara conectada.
4. **Controlador de vuelo Pixhawk 6X RT:** El controlador de vuelo está conectado directamente a la placa de distribución de energía para recibir la alimentación necesaria. Además, gestiona la comunicación con los ESC y otros sensores del dron, como el módulo GPS (M9N) conectado al puerto GPS1.
5. **Raspberry Pi 5:** La Raspberry Pi está alimentada a través del regulador de voltaje y se conecta al controlador de vuelo mediante un puerto serial para recibir datos y enviar comandos al sistema. Además, gestiona la cámara conectada por USB, que captura imágenes para el procesamiento en tiempo real.

6. **Módulo GPS M9N:** Este módulo se conecta al controlador Pixhawk a través del puerto GPS1 para proporcionar datos de posicionamiento en tiempo real en exterior, esenciales para la navegación del 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.

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 redundates.
- **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 3.24.

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 3.25.

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

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 3.26 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.

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)
```

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)
```

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] =
```

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.

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.estimatePoseSingleMarke` 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.

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.

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.

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.

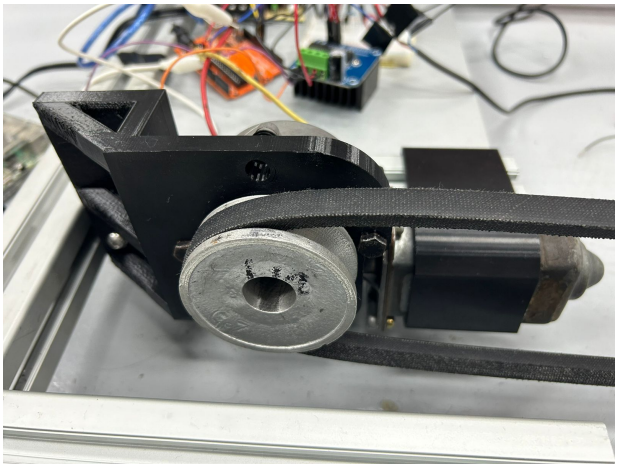


Figure 3.17: Vista del mecanismo para el movimiento del cajón interno, donde se muestran los sujetadores del motor impresos en 3D y el motor con la polea y la banda integradas al sistema.

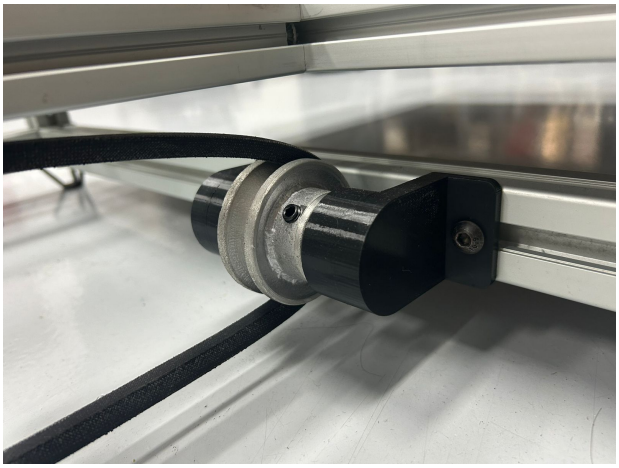


Figure 3.18: Vista del mecanismo para el movimiento del cajón interno, donde se muestran las chumaceras impresas en 3D, con...

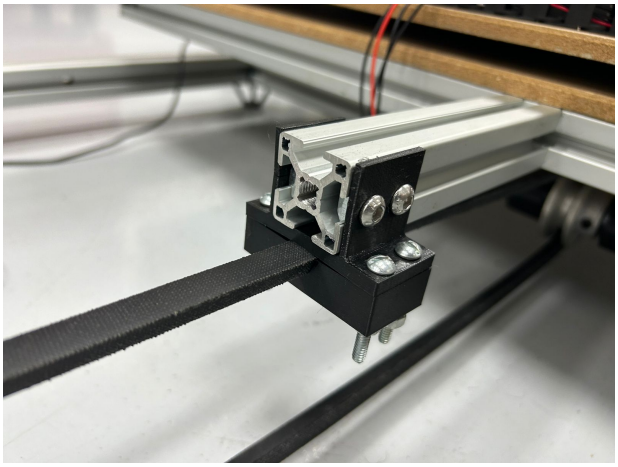


Figure 3.19: Vista del mecanismo para el movimiento del cajón interno, donde se muestran la sujeción que se imprimió en 3D para abrazar la banda con el cajón interno.

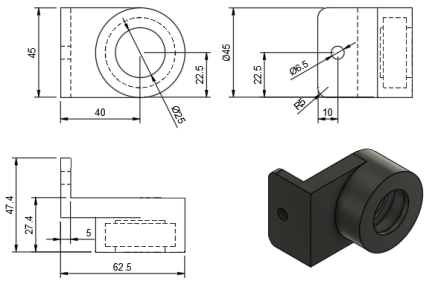
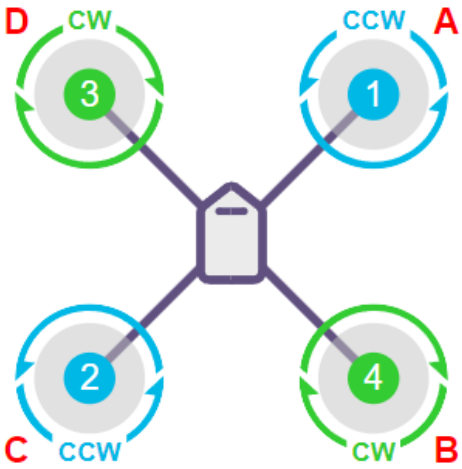
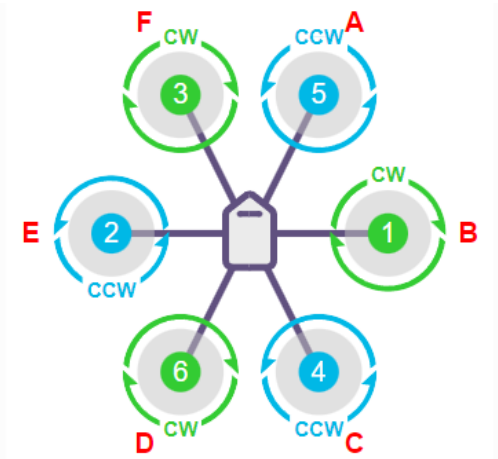


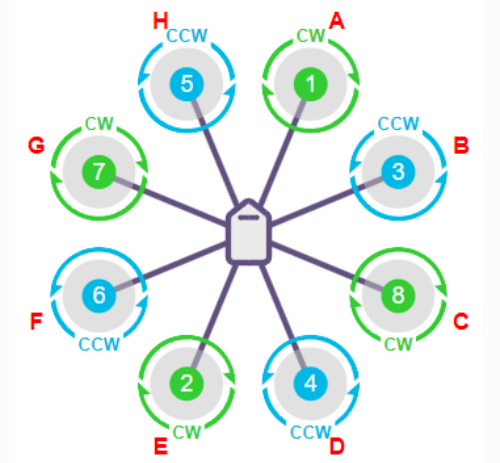
Figure 3.20: Plano de chumaceras para el mecanismo de movimiento del cajón interno.



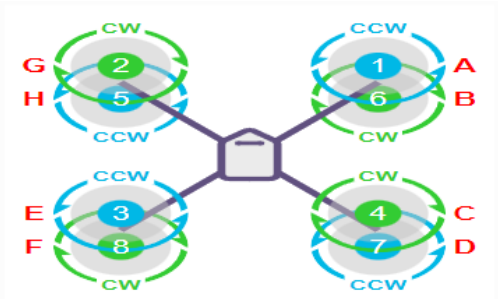
(a) Quad X Frame.



(b) Hexa X Frame.



(c) Octo X Frame.



(d) Octo Quad X Frame.

Figure 3.21: Drone configurations.

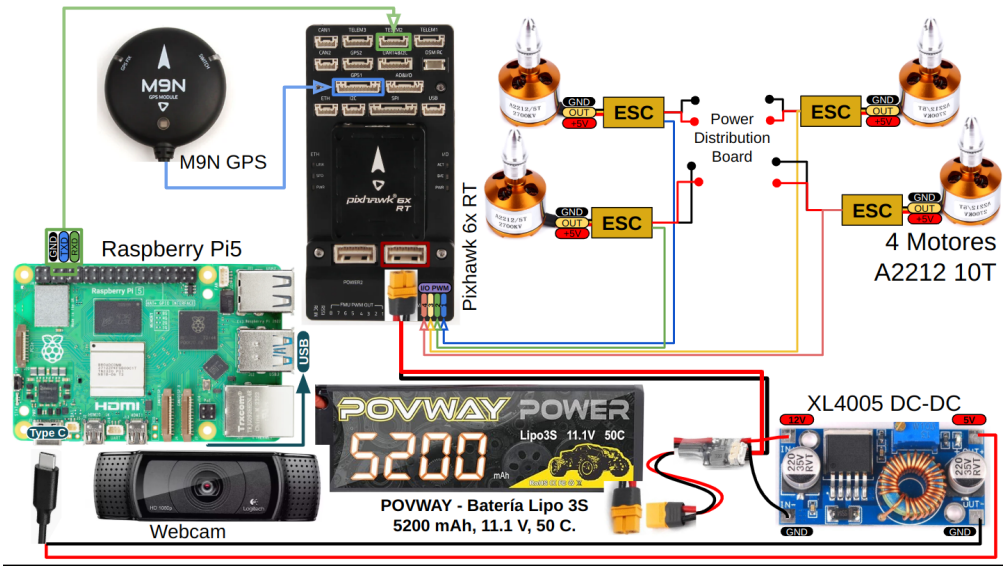


Figure 3.22: Diagrama del circuito de distribución de energía del dron.

Figure 3.23: Imagen de la configuración de sensores en Mission Planner.

Figure 3.24: Ejemplo de imagen descalibrada tomada durante el proceso de captura.

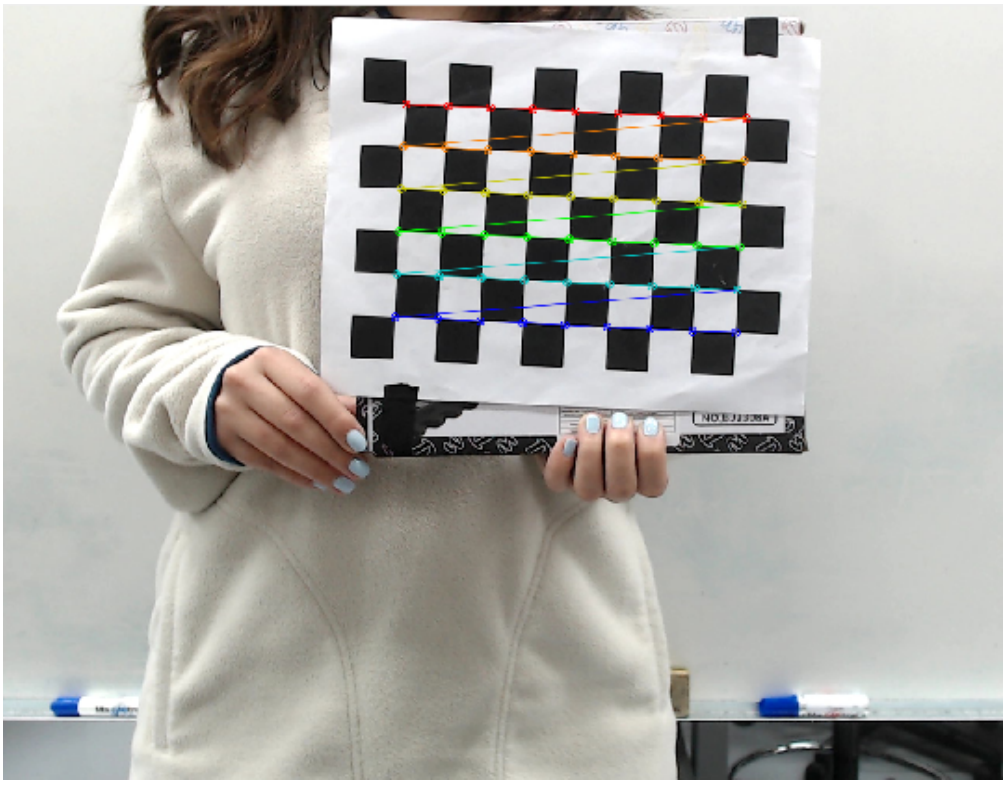


Figure 3.25: Detección y refinamiento de esquinas en el tablero de ajedrez.

Figure 3.26: Imagen calibrada después de aplicar los parámetros obtenidos.

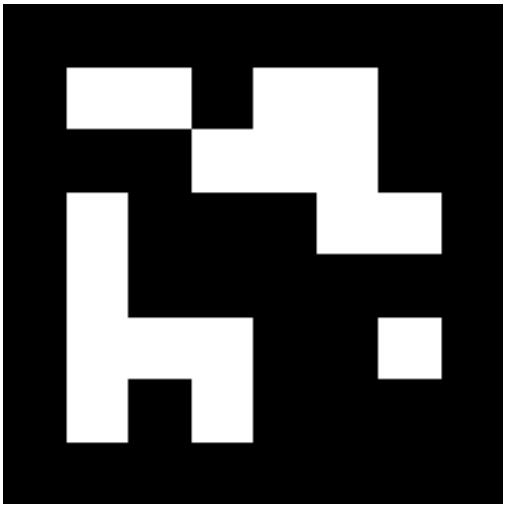


Figure 3.27: Marcador ArUco externo generado.

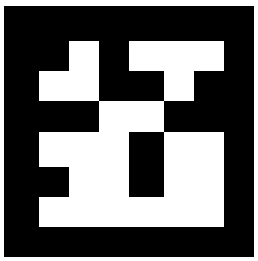


Figure 3.28: Marcador interno con borde blanco añadido.

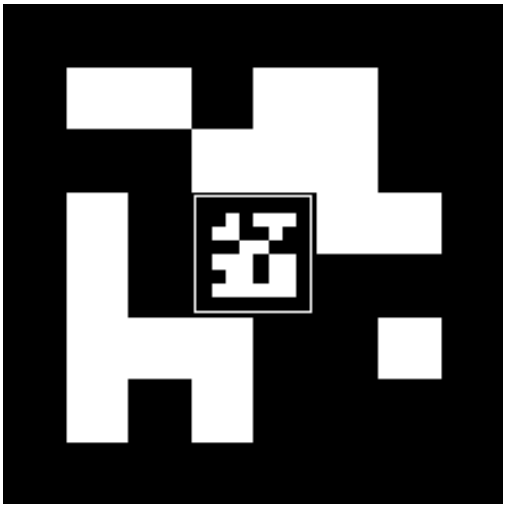


Figure 3.29: Marcador ArUco embebido generado.

Figure 3.30: Detección de marcadores Aruco en la imagen procesada.

Figure 3.31: Representación gráfica de la posición y orientación calculada de un marcador Aruco.

Figure 3.32: Imagen con marcadores detectados y ejes proyectados.

Figure 3.33: Diagrama de la red local WiFi utilizada en el sistema.

Figure 3.34: Diagrama de comunicación entre los dispositivos utilizando ROS 2.

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

En esta sección se presentan los resultados obtenidos en el diseño y manufactura de la estación de carga y del drone. Se incluyen los resultados de las pruebas de funcionamiento de la estacíon de carga y de las pruebas de vuelo, detección de ArUcos y comunicación entre los diferentes dispositivos.

4.1.1 Diseño CAD Final del 1er protopipo 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

Chapter 6

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].

[10] GPS y RTK-GPS en sistemas de navegación para drones. Disponible en cualquier documentación técnica sobre GPS para drones, como los recursos de fabricantes (ej. Pixhawk o ArduPilot).

[11] *Non-GPS Navigation for ArduPilot.* Disponible en: `https://ardupilot.org/copter/docs/common-non-gps-navigation-landing-page.html`

[12] Gupta, A., "UART Communication," *The IoT Hacker's Handbook: A Practical Guide to Hacking the Internet of Things*, Apress, Berkeley, CA, 2019, pp. 59–80. DOI: https://doi.org/10.1007/978-1-4842-4300-8_4.

[13] Wootton, C., "Serial Peripheral Interface (SPI)," *Samsung ARTIK Reference: The Definitive Developers Guide*, Apress, Berkeley, CA, 2016, pp. 335–349. DOI: https://doi.org/10.1007/978-1-4842-2322-2_21.

[14] Gazi, O. and Arlı, A. Ç., "Inter Integrated Circuit (I2C) Serial Communication in VHDL," *State Machines using VHDL: FPGA Implementation of Serial Communication and Display Protocols*, Springer International Publishing, Cham, 2021, pp. 193–235. DOI: https://doi.org/10.1007/978-3-030-61698-4_5.

[15] Subero, A., "USART, SPI, I2C, and Communication Protocols," *Programming PIC Microcontrollers with XC8: Mastering Classical Embedded Design*, Apress, Berkeley, CA, 2024, pp. 297–366. DOI: https://doi.org/10.1007/979-8-8688-0467-0_8.

[16] Grlj, C. G., Krznar, N. "A decade of UAV docking stations: A brief overview of mobile and fixed landing platforms," *Drones*, vol. 6, no. 1, 2022, p. 17. DOI: https://doi.org/10.3390/drones6010017.

[17] DJI, "Aplicaciones en la industria agrícola," Disponible en: https://ag.dji.com/es. [*Último acceso: noviembre, 2024*].

[18] Amazon Prime Air, "Preparativos para entregas de paquetes con drones," Disponible en: https://www.aboutamazon.com/news/transportation/amazon-prime-air-prepares-for-drone-deliveries. [*Último acceso: noviembre, 2024*].

[19] Marek, D., Paszkuta, M., Szyguła, J., Biernacki, P., Domanski, A., Szczygieł, M., Krol, M., Wojciechowski, K., "General concepts in swarm of drones control: Analysis and implementation," *2023 IEEE International Conference on Big Data (BigData)*, 2023, pp. 5070–5077. DOI: https://doi.org/10.1109/BigData52589.2023.10386672.

[20] Military Africa, "Swarm drones: A new phase in the Sudan conflict," Disponible en: https://www.military.africa/2024/10/swarm-drones-a-new-phase-in-the-sudan-conflict/. [*Último acceso: noviembre, 2024*].

[21] Piotr, W., et al., "Drones or Unmanned Aerial Systems (UAV - Unmanned Aerial Vehicle or UAS - Unmanned Aerial Systems)," *Introduction to UAV Systems*, Wiley, 2016, pp. 25–50. Disponible en: https://doi.org/10.1002/9781118899893.

[22] ArduPilot Documentation, "Multicopter Overview," Disponible en: https://ardupilot.org/copter/docs/common-multicopter-overview.html. [*Último acceso: noviembre, 2024*].

[23] Bacco, M., et al., "Drone Swarms in Industry 4.0: Potential and Challenges," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 12, 2019, pp. 6116–6125. DOI: https://doi.org/10.1109/TII.2019.2904121.

[24] Zhao, Z., et al., "Survey on Drone Charging Stations: Methods and Designs," *Drones*, vol. 5, no. 2, 2021, pp. 50–62. DOI: https://doi.org/10.3390/drones5020050.

[25] Anaya, C., "Understanding Brushless Motors for Drones," *Drone Electronics: A Guide*, Springer, 2020, pp. 35–55. DOI: https://doi.org/10.1007/978-3-030-41326-2.

[26] Vetelino, J., "Design and Selection of Propellers for UAVs," *Introduction to Drone Engineering*, CRC Press, 2018, pp. 145–170. Disponible en: https://www.crcpress.com.

[27] Chen, L., "Advances in Lithium Polymer Batteries for UAVs," *Journal of Power Sources*, vol. 360, 2021, pp. 123–140. DOI: https://doi.org/10.1016/j.jpowsour.2021.230917.

[28] Lu, X., et al., "Wireless Charging Technologies: Fundamentals, Standards, and Network Applications," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, 2016, pp. 1413–1452. DOI: https://doi.org/10.1109/COMST.2015.2499783.

[29] Chen, J. and Huang, S., "Analysis and Comparison of UART, SPI and I2C," *2023 IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)*, 2023, pp. 272–276. DOI: https://doi.org/10.1109/EEBDA56825.2023.10090677.

[30] Sadeghian, O., et al., "A comprehensive review on electric vehicles smart charging: Solutions, strategies, technologies, and challenges," *Journal of Energy Storage*, vol. 54, 2022, pp. 105241. DOI: https://doi.org/10.1016/j.est.2022.105241.

[31] Mohammed, S. A. Q., and Jung, J. W., "A Comprehensive State-of-the-Art Review of Wired/Wireless Charging Technologies for Battery Electric Vehicles: Classification/Common Topologies/Future Research Issues," *IEEE Access*, vol. 9, 2021, pp. 19572–19585. DOI: https://doi.org/10.1109/ACCESS.2021.3055027.

[32] Rohde , Schwarz, "Optimizing ESC Design: Insights into Efficient Electronic Speed Controllers for Drones," Technical Whitepaper, 2020. Disponible en: https://www.rohde-schwarz.com. [*Último acceso: noviembre, 2024*].

[33] Pixhawk Development Team, "Pixhawk Autopilot Platform," 2023. Disponible en: https://docs.px4.io/. [*Último acceso: noviembre, 2024*].

[34] Raspberry Pi Foundation, "Raspberry Pi Documentation," Disponible en: https://www.raspberrypi.com/documentation/. [*Último acceso: noviembre, 2024*].

[35] Neaimeh, M., Salisbury, S. D., Hill, G. A., Blythe, P. T., Scoffield, D. R., Francfort, J. E., "Analysing the usage and evidencing the importance of fast chargers for the adoption of battery

electric vehicles," *Energy Policy*, vol. 108, pp. 474-486, 2017. DOI: https://doi.org/10.1016/j.enpol.2017.06.033.

[36] ROS 2 Documentation, "Humble Hawksbill (LTS) Overview," Disponible en: https://docs.ros.org/en/humble/. [*Último acceso: noviembre, 2024*].

[37] ROS 2 Documentation, "Jazzy Jalisco Overview," Disponible en: https://docs.ros.org/en/jazzy/. [*Último acceso: noviembre, 2024*].

[38] OpenCV Documentation, "Camera Calibration and 3D Reconstruction (calib3d module)," Disponible en: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html. [*Último acceso: noviembre, 2024*].

[39] Garrido-Jurado, S., et al., "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, 2014. DOI: https://doi.org/10.1016/j.patcog.2014.01.005.

[40] MAVLink Documentation, "Protocol Overview," Disponible en: https://mavlink.io/en/. [*Último acceso: noviembre, 2024*].

[41] PX4 and ROS Integration, "Using MAVROS and Micro XRCE-DDS with ROS 2," Disponible en: https://docs.px4.io/en/ros2/. [*Último acceso: noviembre, 2024*].

[42] OpenCV Documentation, "Detection of ArUco Markers," Disponible en: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. [*Último acceso: noviembre, 2024*].

[43] OpenCV Documentation, "ArUco Marker Detection: Example Tutorial," Disponible en: https://docs.opencv.org/4.x/dc/dc3/tutorial_aruco_detection.html. [*Último acceso: noviembre, 2024*].