Charging Station System for swarming 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

November 30th 2024

Advisor: Fermín Castro Aragón

Contents

Chapter 1

Introduction

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

State of the Art

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 [23], "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 [23], "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" [22]. 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.

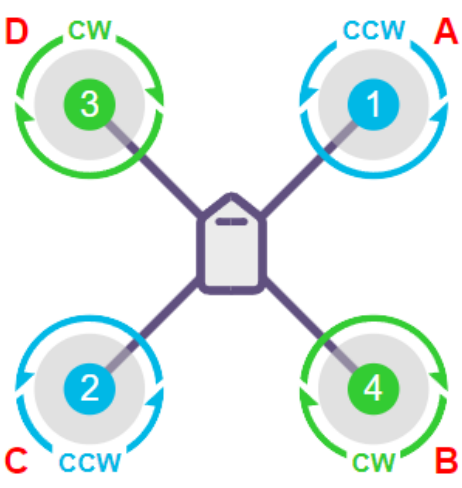
In Figure 3.21c, the Octo Frame configuration is shown, featuring eight motors and propellers. Drones with this configuration are ideal for industrial applications due to their higher payload capacity and redundancy, which improves safety in case of motor or propeller failures.

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.

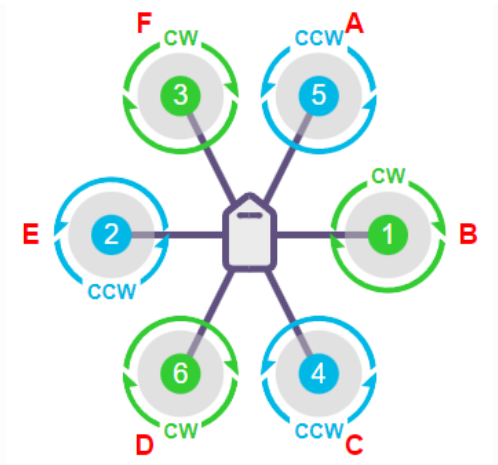
2.2.2 Motors

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:

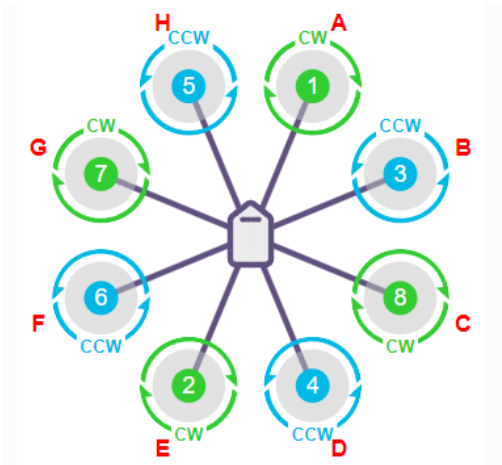
- **Kv Rating:** Indicates the revolutions per minute (RPM) per applied volt without load. A lower Kv provides more torque, making it ideal for larger drones with bigger propellers.
- **Compatibility:** The motor must be compatible with the electronic speed controller (ESC) and the propellers.
- **Mounting:**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



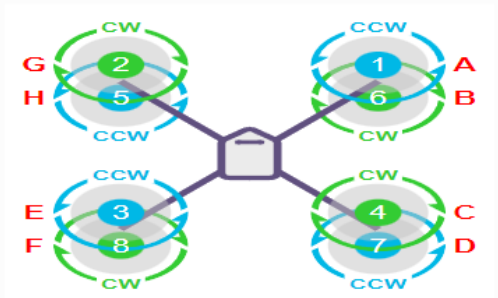
(a) Quad X Frame.



(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. [18]

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. [18]

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

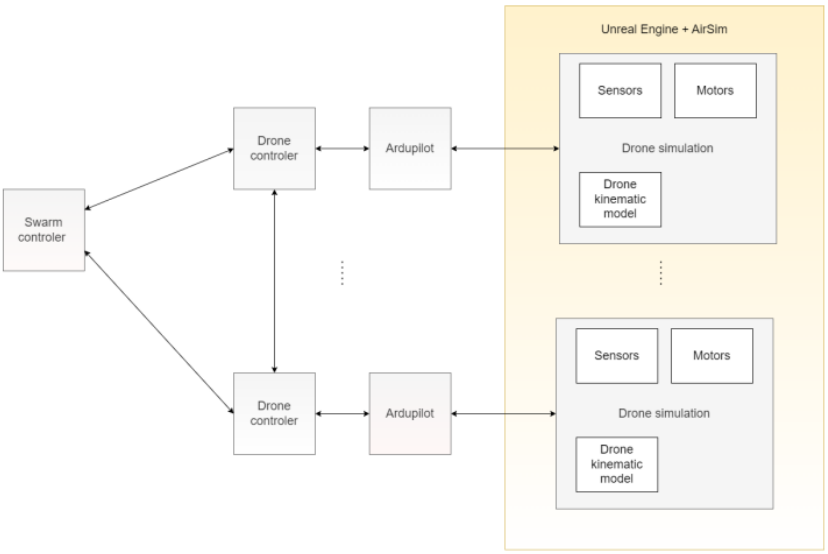


Figure 2.5: Diagram simulating a drone swarm model

Figure 2.6: An example of drone swarm is shown in the image. [20]

to land safely, take off, recharge and/or replace batteries, and transfer data and payload" [17].

In this context, various types of charging stations can be observed, classified according to the following specifications [17]:

- 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)

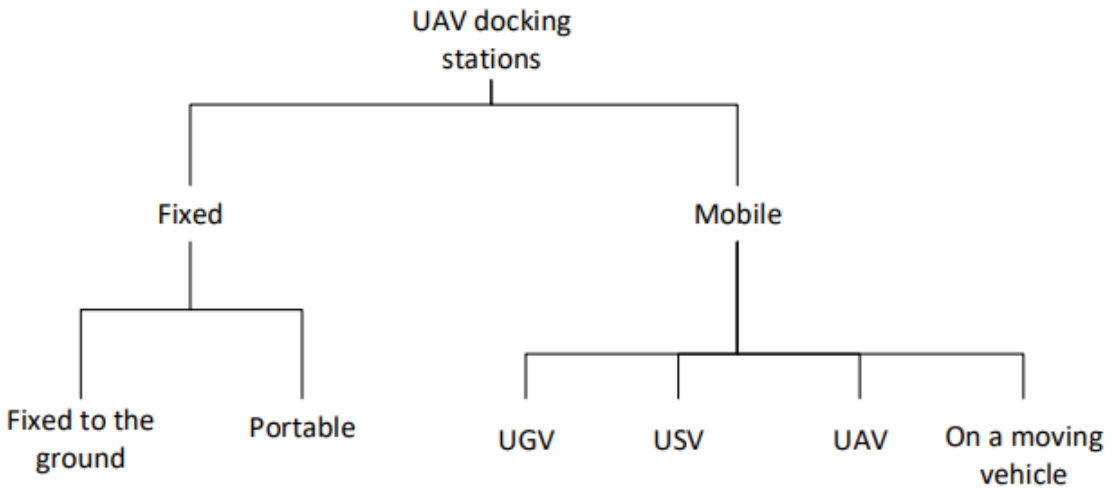


Figure 2.7: Drone Charging Stations Classification [17].

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 [17].

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.

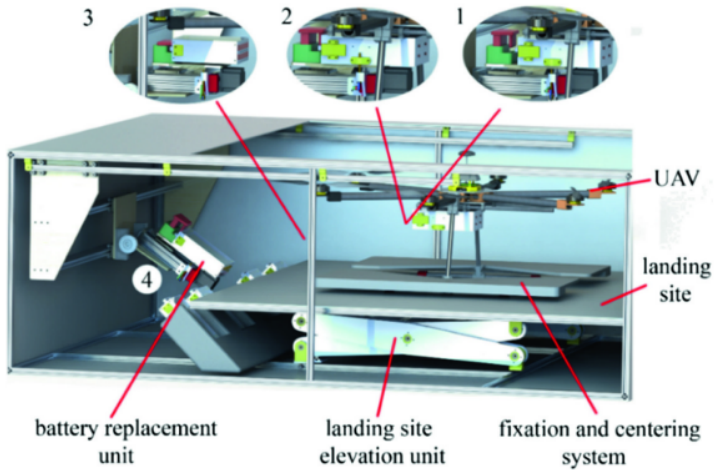


Figure 2.8: Fixed docking station with its subsystems. The labels (1–4) show the battery replacement mechanism in operation [17].

Figure 2.9: Battery replacement docking station [17].

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 [17].

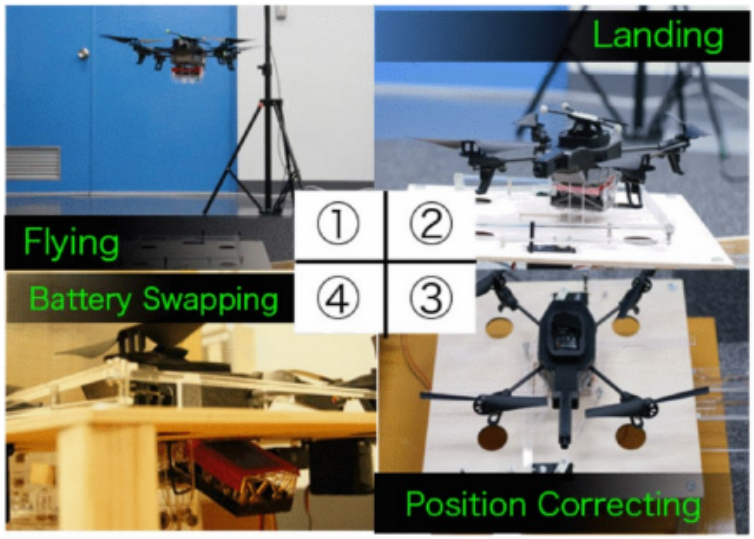


Figure 2.10: Endless Flyer [17], docking station with battery replacement mechanism. The labels in the center show the different landing stages and the order in which they are realized.

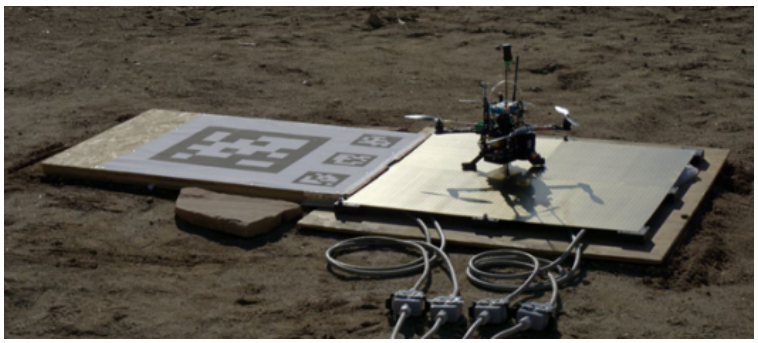


Figure 2.11: Docking station with recharging pad with the custom made UAV from [17]. In the figure, the QR codes used in the approach and landing phase can be seen .

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

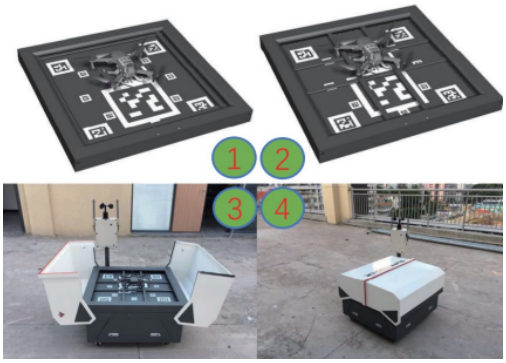


Figure 2.12: Moving docking station with storage system and battery replacement system. Labels in the figure (1–4) show the order in which the active positioning mechanism and storing mechanism take place [17].

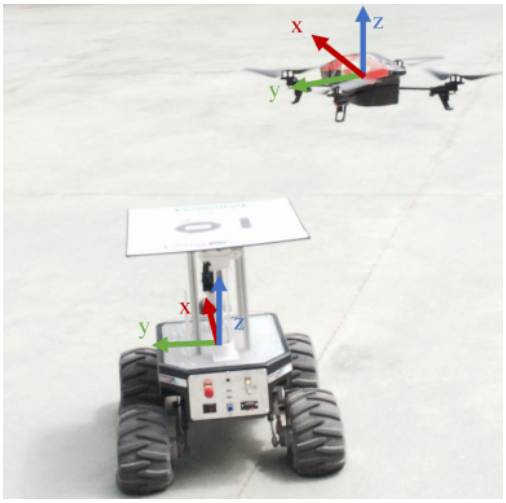


Figure 2.13: UGV–UAV proposed moving docking station solution [17].

Figure 2.14: USV–UAV landing system [17].

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" [33]. 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" [34]. 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 [35].

- **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 [13].
- **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 [14].
- **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 [15].

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 [30].

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. [36]

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. [31]

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. [32]

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. [29]

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 the *ISO/IEEE*, is *"combination of a hardware device and computer instructions and data that reside as read-only software on that device"* [3]. 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)

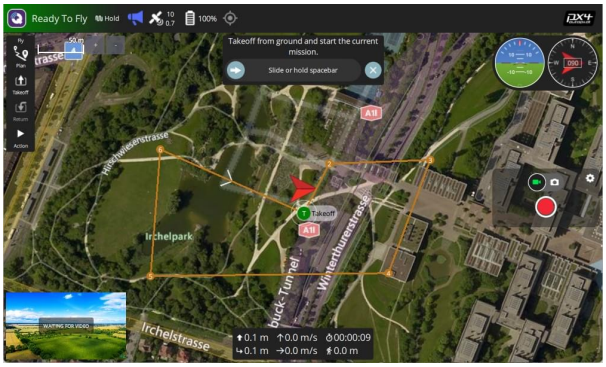
Ground control systems (GCS) are auxiliary tools widely used for the configuration, calibration, and monitoring of drones. Although they are not the main focus of this project, they play an important role in system preparation, facilitating parameter configuration, sensor calibration, and supervision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



(a) QGroundControl Interface.

(b) Mission Planner Interface.

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

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 *"designed for security, reliability, and ease of use"* [5]. 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 [5].

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"* [6]. 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 [7]. 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 [37].

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 [38].

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 [7].

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 [7]:

- **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.

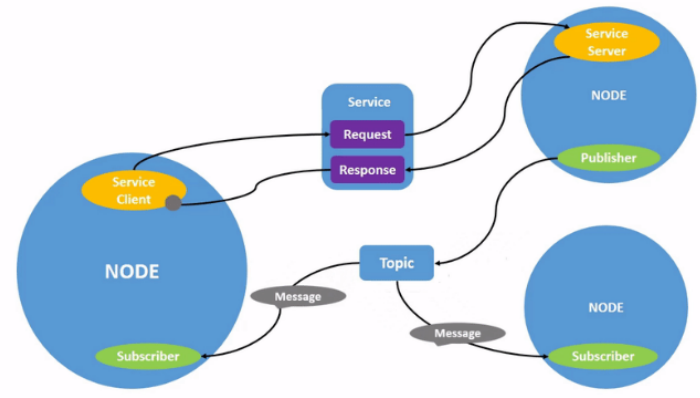


Figure 2.18: ROS 2 Architecture [7].

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 [7]:

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 [8].

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"* [8]. 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 [8].

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"* [39].

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.

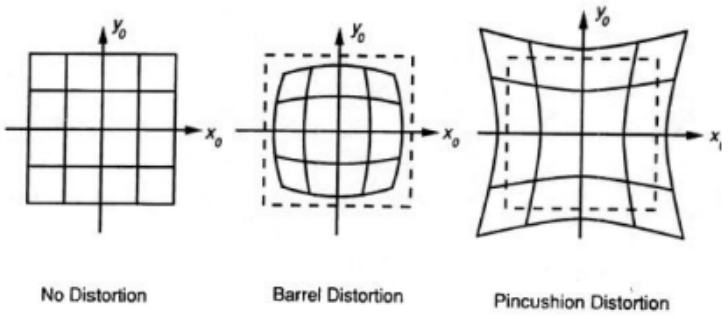


Figure 2.19: Types of camera distortions.

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.

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"* [9]. Detecting these markers is crucial in applications requiring precise estimation of the position and orientation of objects in three-dimensional space.

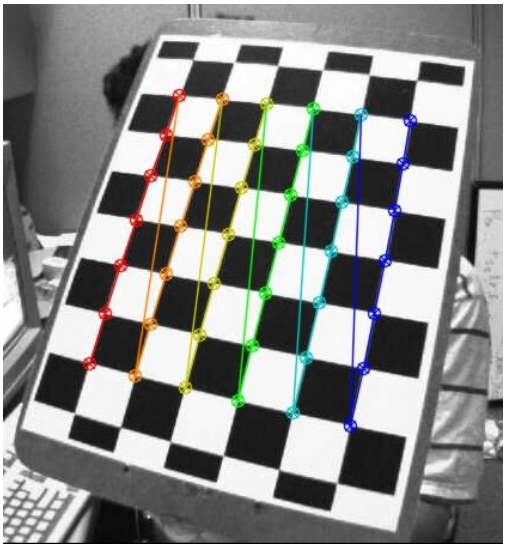


Figure 2.20: Chessboard pattern used in camera calibration with OpenCV.

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 [9].

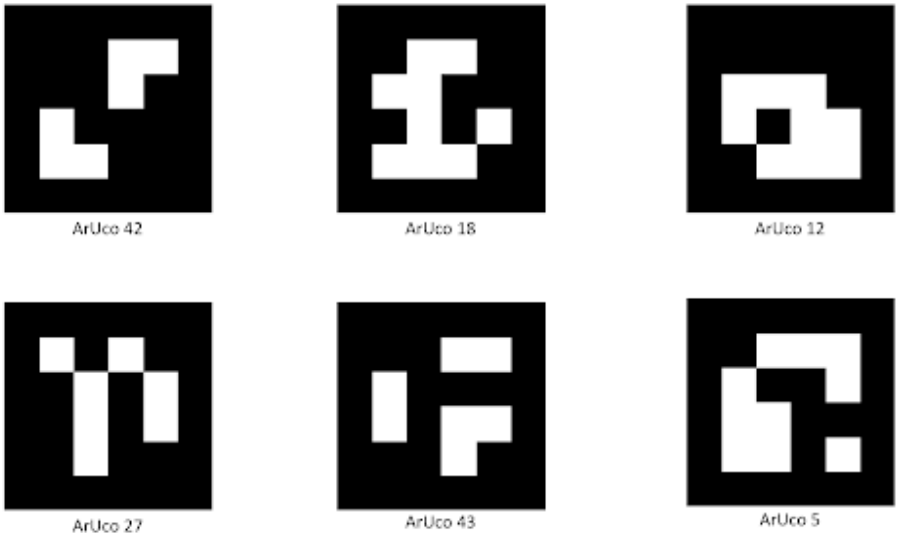


Figure 2.21: Examples of ArUco markers and IDs.

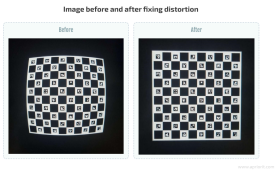


Figure 2.22: Camera Calibration with ArUco

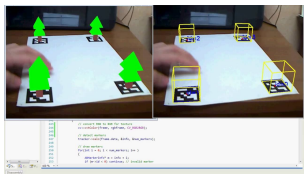


Figure 2.23: Augmented Reality with ArUco

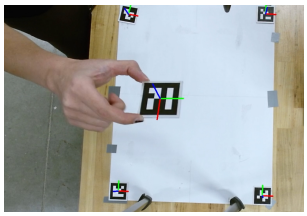


Figure 2.24: Real-Time Tracking with ArUco

Figure 2.25: Common applications of ArUco markers.

Common Applications

ArUco markers are used in various applications, including camera calibration, augmented reality, and robot and drone navigation and control. One of the advantages of using ArUco is their ability to act as reference points in 3D environments, enabling computer vision systems to calculate the camera pose. According to the documentation, *"Markers can be used as 3D landmarks for camera pose estimation"* [40]. 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 [9].

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

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

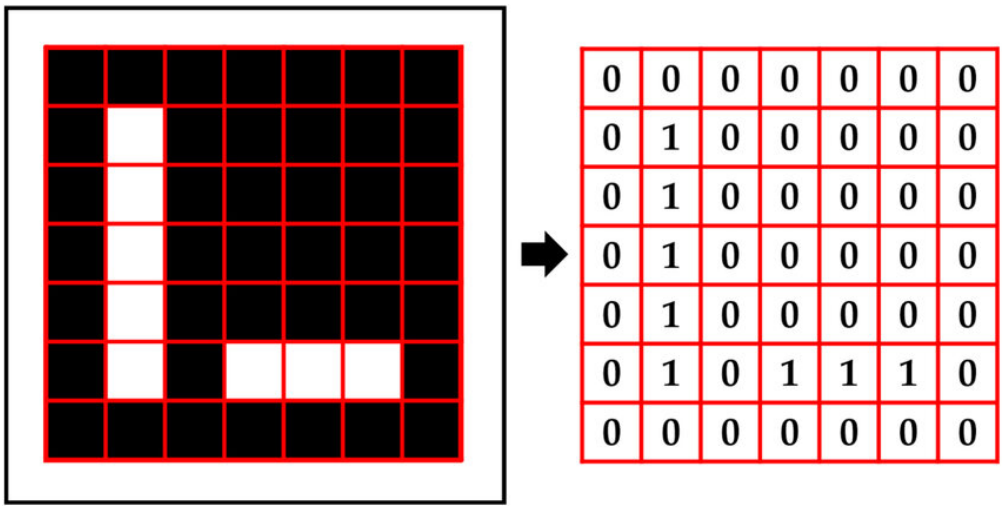


Figure 2.26: Bits of a 7x7 ArUco marker.

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"* [10]. 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 [10].

Figure 2.27: Large/Small ArUco Marker Generation.

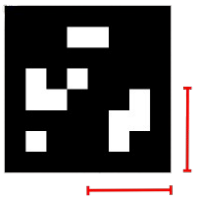


Figure 2.28: Calculating the Large ArUco Marker Center.

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 [10]. 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 [43]. 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"* [44].

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 [43]. 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.

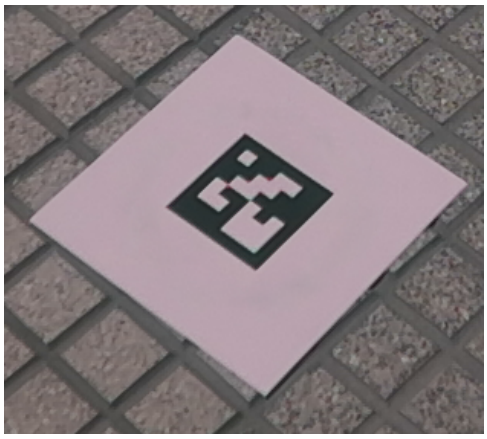


Figure 2.31: ArUco without pose estimation.

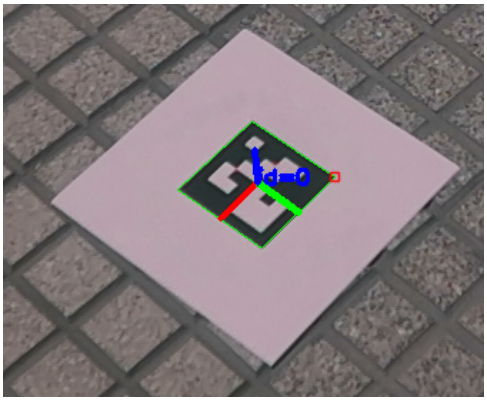


Figure 2.32: ArUco with pose estimation.

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"* [41]. 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 [42].

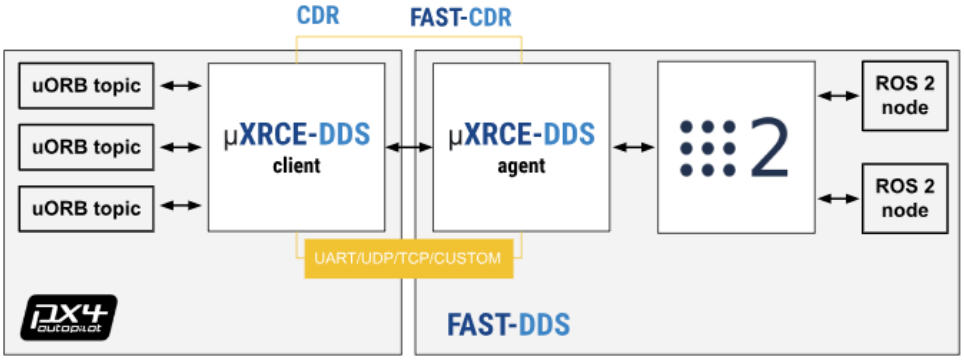


Figure 2.33: ROS 2 - Pixhawk communication using Micro XRCE-DDS.

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 [42].

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 [11].
- **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 [11].

- **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 [11].

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 [12].
- **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 [12].
- **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 [12].
- **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 [12].

Chapter 3

Development

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 Design and Development of the Charging Station

3.1.1 Requirements

The following outlines the technical specifications and requirements for the charging station, ensuring compatibility and efficiency for a drone swarm charging system:

1. The system must feature a modular and stackable design for integrating multiple charging stations.
2. The system must take into account and adapt to the dimensions of the proposed drone for the application.
3. The system must include an automatic mechanism for opening and closing to store the drone.
4. The system must have a charging mechanism for the drone's battery.
5. The system must incorporate a vision-based positioning method for the drone.

3.1.2 Bill of Materials

The following is the list of materials used for constructing the modular charging station. It is worth noting that the design and development of this project heavily relied on previously available resources and materials, enabling cost optimization and efficient use of existing supplies.

- Aluminum Profiles
  - **30 mm:**

        * 4 x Profiles of 100 mm
        * 4 x Profiles of 810 mm
        * 6 x Profiles of 750 mm
    - **40 mm:**
        * 2 x Profiles of 1040 mm
        * 4 x Profiles of 560 mm
        * 4 x Profiles of 830 mm
        * 4 x Profiles of 1120 mm
- 16 x 3D-Printed Aluminum Profile Elbows
- Aluminum Profile Fasteners
- 2 x Telescopic Slide Rails
- 1 x 12V Motor
- 1 x 3D-Printed Motor Bracket
- 1 x V-Belt 1 m
- 2 x Type A Pulleys with 1/2" Inner Diameter
- 2 x Bearings with ID: 15 mm and OD: 35 mm
- 2 x 3D-Printed Bearing Housings
- 2 x 9 mm MDF Boards of 75x75 cm
- 8 x 3D-Printed MDF Spacers
- 1 x 12V Power Supply
- 1 x Arduino Mega
- 1 x Raspberry Pi 4
- 1 x BTS7960 H-Bridge
- 2 x Inductive Proximity Sensors
- 1 x Joystick
- 1 x WiFi Router
- 1 x BT3 Pro Compact Charger
- 1 x Cable Carrier 1 m
- 1 x Shaft (dimensions pending)

3.1.3 CAD Design of the Charging Base

To meet the modular/stackable design requirements and ensure compatibility with the drone's dimensions, the following CAD design was developed:

As shown in Fig. 1, the base known as the "internal drawer" was designed to serve as a platform for the drone's takeoff and landing. This design was created considering the drone's dimensions and ensuring the inclusion of adequate tolerances to prevent potential collisions. A 30 mm aluminum profile was used in this design, taking advantage of previously available materials. Additional profiles were also integrated horizontally in the middle of the drawer to support the rails.

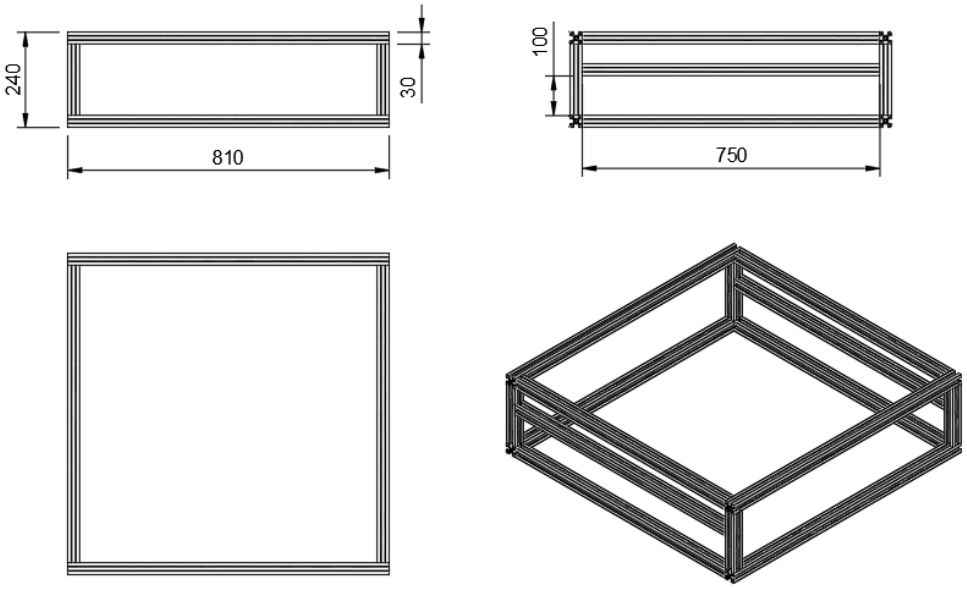


Figure 3.1: Blueprint of the internal drawer for the Charging Station.

Consequently, based on the dimensions of the previous blueprint, the "external drawer" was designed. This drawer serves to house the drone and the electronics of the charging station, as shown in Fig. 2.

Next, the dimensions of the drone's landing gear were considered to 3D print mechanical guides (Fig. 3) that help align the landing gear for proper magnetic connection with the charging system.

Using the designs from Fig. 1 and Fig. 3, the complete design of the internal drawer was developed (Fig. 4). It includes two 9 mm MDF cuts measuring 810 mm x 810 mm and internal spacers with a height of 20 mm to protect the system's cables.

The necessary CAD designs, as shown in Fig. 5 and Fig. 6, were developed to ensure proper integration and adaptation of the various components of the charging station. These designs helped establish the essential dimensions, tolerances, and characteristics to ensure that each element fit precisely, guaranteeing that the charging station functioned efficiently and reliably.

Lastly, as part of the design process, a CAD model was developed to integrate the main com-

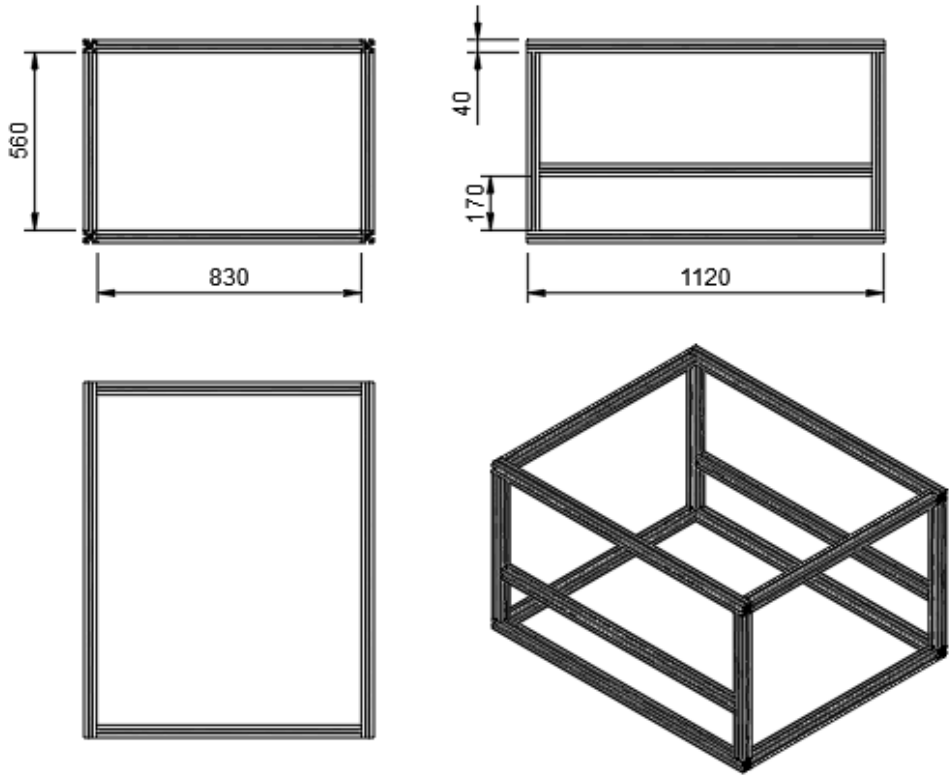


Figure 3.2: Blueprint of the external drawer for the Charging Station.

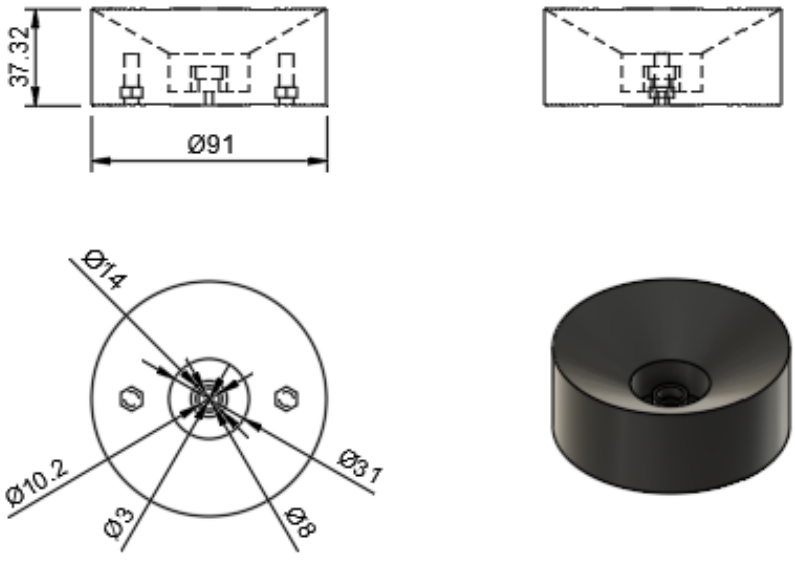


Figure 3.3: Blueprint of the mechanical guide for the Charging Station.

Figure 3.4: View of the internal drawer for the Charging Station.

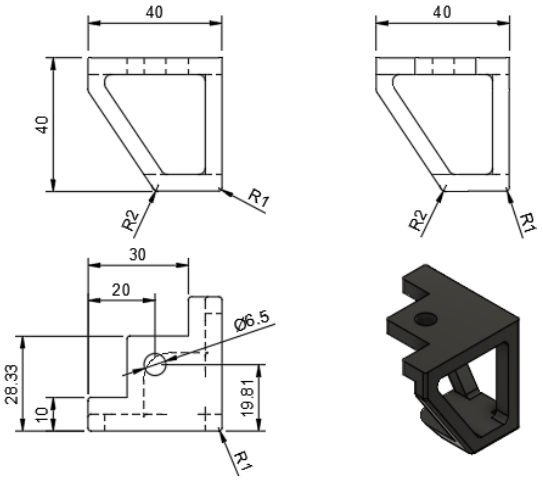


Figure 3.5: Floor supports for the stability of the Charging Station.

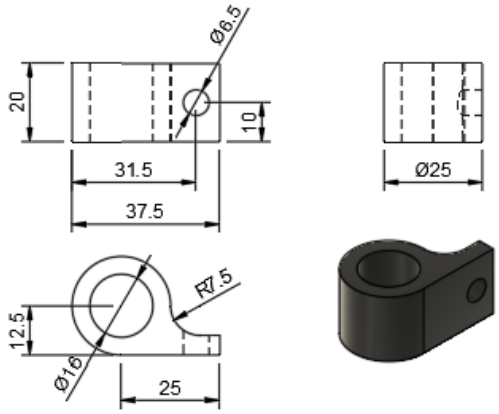


Figure 3.6: 3D-printed holder for the inductive proximity sensors of the Charging Station.

ponents necessary for the proper functioning of the charging station. This final conceptual design, shown in Figure 7, represents the consolidation of initial ideas, the iterations performed, and the decisions made throughout the development. The model ensures the functionality, compatibility, and efficiency of the station, serving as a basis for manufacturing and implementing the system.

3.1.4 Selection and Implementation of the Drawer Movement Mechanism

The design of the automatic drawer mechanism for the charging station went through several conceptual stages, considering different options to ensure efficient and cost-effective movement. Initially, a toothed belt mechanism, similar to those used in 3D printers, was proposed to allow precise forward and backward movement. However, this solution proved costly and complex due to the required dimensions. As an alternative, pistons were considered, but they further increased the project's costs. Finally, a V-belt and pulley system (Type A) was chosen due to its quick availability and significant cost reduction, achieving a balance between functionality, simplicity, and economy.

Components and CAD Designs

1. 12V Motor
2. Pulleys
3. V-Belt Type A
4. Bearings

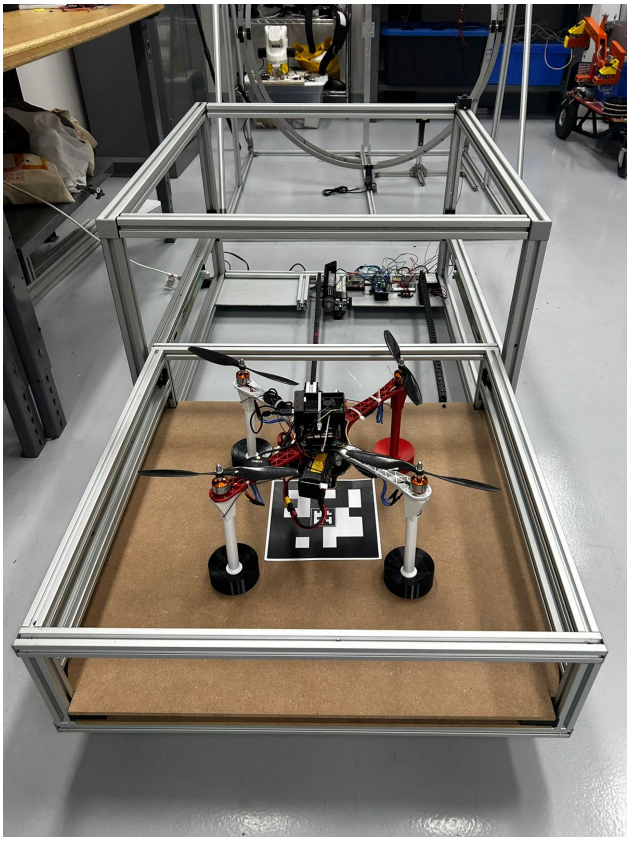


Figure 3.7: View of the Charging Station.

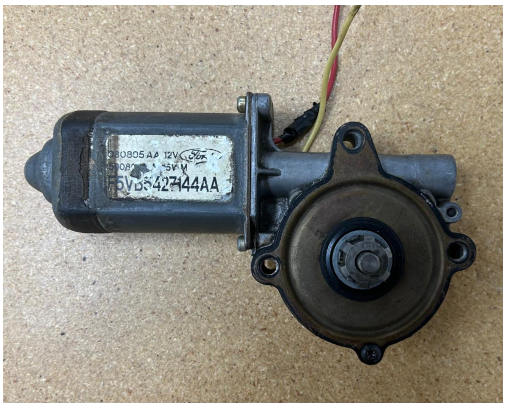


Figure 3.8: 12V Motor used for the Charging Station.

Figure 3.9: Aluminum pulley with 1/2” inner diameter.

Figure 3.10: V-Belt Type A.

Figure 3.11: Bearings for...

5. Bearing Housings

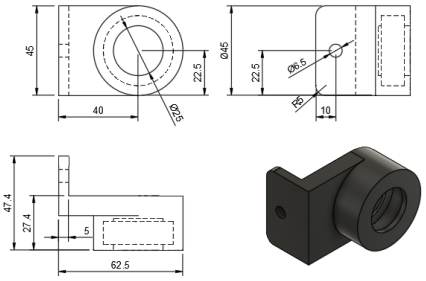


Figure 3.12: Blueprint of bearing housings for the internal drawer movement mechanism.

6. 12V Motor Mounts

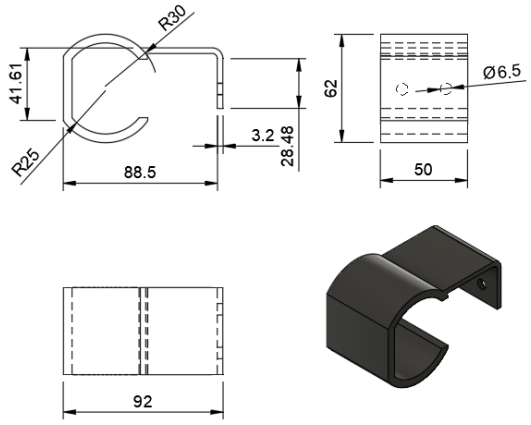


Figure 3.13

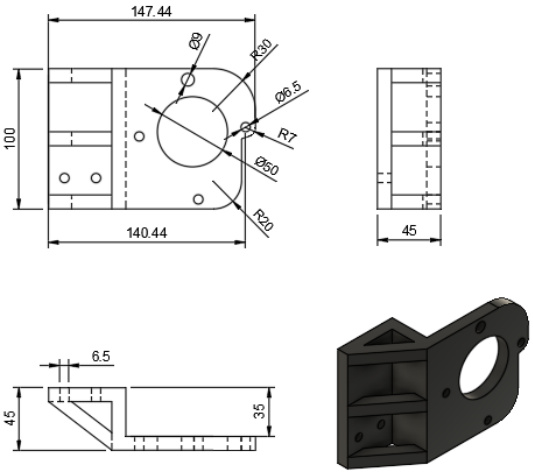


Figure 3.14

7. V-Belt Holder for the internal drawer

V-Belt and Pulley Mechanism

3.1.5 Manufacturing Process of the Charging Base Structure

1. **Material Cutting:** To begin the manufacturing process, precise cuts were made on the aluminum profiles and MDF plates according to the dimensions specified in the bill of materials. This step is crucial to ensure that all pieces assemble correctly in the modular design. This process can be observed in Figure ? a).

2. **Structure Assembly:** Once the pieces were cut, the main structure of the charging station was assembled using screws and nuts to secure the aluminum profiles and MDF plates with 3D-printed 20 mm spacers. It was verified that all components were aligned and leveled to ensure the stability and strength of the charging base. This is shown in Figures ? b) and c).

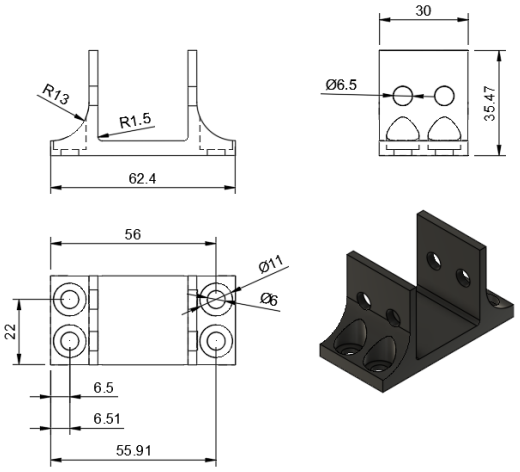


Figure 3.15

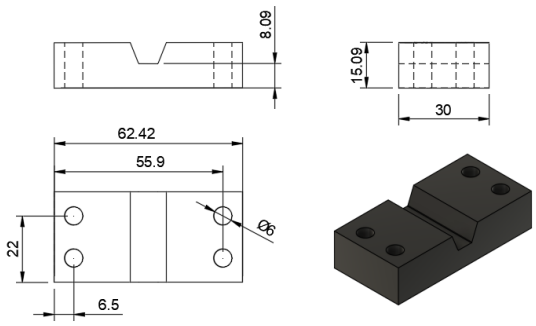


Figure 3.16

3. **Installation of the Movement Mechanism:** After assembling the main structure, the V-belt and pulley mechanism was installed to enable the horizontal movement of the drawer. The pulleys, belt, and motor were placed in their predefined locations, ensuring that the movement system operated correctly and without obstructions. Figure ? 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 Drone Instrumentation

3.2.1 Specifications and Requirements

To ensure compatibility and functionality within the charging system, the drone must meet the following specifications:

- The dimensions of the components must adapt to the frame of the open-architecture quadcopter purchased.
- The drone must have a charging circuit compatible with the charging station.
- The drone must include a camera and vision system for detecting Aruco markers.
- Localization sensors must be integrated for outdoor navigation.
- A microprocessor must be integrated as an auxiliary computer for data processing and communication with the charging station.

3.2.2 Bill of Materials for Drone Instrumentation

- 1 x Open-architecture quadcopter frame (specify model)
- 1 x Flight controller (specify model)
- 1 x Camera (compatible with Aruco detection)
- 1 x Raspberry Pi 4 (Companion Computer)
- 1 x Proximity sensor (preferred model)
- 1 x GPS module (compatible with the flight controller)
- Wiring and connectors
- 3D-printed mounting material (specific mounts for each component)

3.2.3 Drone CAD Designs

Below are the CAD designs of the components developed for the drone, adapted to its original frame to integrate the necessary electronic components and meet the requirements for charging and positioning.

- New mounts were designed for the microprocessor and camera, ensuring stable and precise integration into the frame.
- A mounting space was implemented for the localization sensors and the charging circuit.

Image of the drone CAD designs with the modified components.

3.2.4 Power Distribution Circuit

The power distribution circuit was designed to provide safe and stable power to the drone's critical components, including the motors, flight controller, Raspberry Pi, and camera. Below is a description of each section of the circuit:

1. **LiPo 3S Battery:** A 3-cell lithium-polymer (LiPo) battery with a capacity of 5200 mAh, a nominal voltage of 11.1 V, and a discharge rate of 50C was selected. This battery is ideal for providing the necessary current to the motors and electronic components without compromising flight duration.
2. **Power distribution to the motors:** Power from the battery is distributed through a power distribution board that connects the battery to the four electronic speed controllers (ESC). Each ESC regulates the power sent to its respective A2212 10T motor, allowing precise control of the motor speeds.

3. **XL4005 DC-DC Voltage Regulator:** This regulator converts the battery's 11.1 V output to 5 V, providing stable and safe power for the Raspberry Pi and the connected camera.

4. **Pixhawk 6X RT Flight Controller:** The flight controller is connected directly to the power distribution board to receive the necessary power. It also manages communication with the ESCs and other sensors on the drone, such as the GPS module (M9N) connected to the GPS1 port.

5. **Raspberry Pi 5:** The Raspberry Pi is powered through the voltage regulator and connects to the flight controller via a serial port to receive data and send commands to the system. It also manages the camera connected via USB, which captures images for real-time processing.

6. **M9N GPS Module:** This module connects to the Pixhawk controller via the GPS1 port to provide real-time positioning data outdoors, essential for drone navigation.

3.3 Software Configuration

3.3.1 Configuration of the Ground Control Computer

The ground control computer was configured to monitor and process information from the drone's companion computer. Below are the steps performed for this configuration:

- **Installation of ROS 2 Humble:** Since the ground control computer uses Ubuntu 22.04, ROS 2 Humble was installed following the official instructions. The commands used were:

  ```
  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
  ```

  This included the installation of the necessary tools to develop and run applications in ROS 2 Humble.

- **Environment Setup:** To simplify the use of ROS 2, the `/.bashrc` file was configured. The following line was added to the end of the file:

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

  Then, the following command was executed:

```
source ~/.bashrc
```

- **Cloning the GitHub Repository:** A ROS 2 workspace was created, and the corresponding repository was cloned. The steps performed were:

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

- **Configuration of `ROS_DOMAIN_ID`:** To ensure proper communication between the ground control computer and the drone's companion computer, the `ROS_DOMAIN_ID` was set to 10. This was done by adding the following line to the `/.bashrc` file:

```
export ROS_DOMAIN_ID=10
```

  Then, the following command was executed:

```
source ~/.bashrc
```

- **Verification of the ROS 2 Environment:** Finally, the environment was verified to be properly configured using the following command:

```
ros2 doctor
```

  This confirmed that all necessary dependencies for ROS 2 Humble were installed and functioning correctly.

Image of the environment setup on the ground control computer.

3.3.2 Configuration of the Drone's Companion Computer

The Raspberry Pi 5 was configured as a companion computer for real-time data processing and communication with the charging station. The steps performed for its configuration are detailed below:

- **Installation of Ubuntu 24.04:** The Raspberry Pi Imager tool was used to install Ubuntu Server 24.04 LTS (64-Bit) on the SD card. During the initial setup, SSH was enabled, a user with a password was defined, and the Raspberry Pi was connected to the WiFi network.
- **Initial Connection and SSH:** After inserting the SD card into the Raspberry Pi and connecting it to a monitor and keyboard, the device was powered on, and the configured credentials were entered. The IP address was obtained using the command:

  ```
  hostname -I
  ```

  Using this information, an SSH connection was established from an external computer with:

  ```
  ssh <user>@<IP>
  ```

  This allowed the Raspberry Pi to be configured remotely.
- **System Update:** A full system update was performed using the following commands:

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

  Additionally, `raspi-config` was installed to enable the serial port. This option was configured in `Interface Options > Serial Port` by selecting `No` and then `Yes`.
- **Modification of `bashrc` and ID Configuration:** To ensure communication between all devices, the `ROS_DOMAIN_ID` was set to 10. The `/.bashrc` file was edited with:

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

  At the end of the file, the following lines were added:

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

  Changes were saved, and the following command was executed:

  ```
  source ~/.bashrc
  ```

- **Installation of ROS 2 Jazzy:** The official instructions were followed to install ROS 2 Jazzy on Ubuntu 24.04. The commands included:

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

- **Cloning the GitHub Repository:** A ROS 2 workspace was created to integrate custom nodes. The steps performed were:

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

- **Installation of MAVROS and MAVProxy:** MAVROS and MAVProxy were installed for communication with the Pixhawk. The commands used were:

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

Additionally, geographic plugins were configured:

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

- **Communication Verification:** To ensure MAVROS and MAVProxy worked correctly, MAVProxy was started first with:

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

Then, MAVROS was launched using:

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

Finally, the available topics were verified with:

```
ros2 topic list
```

and communication was confirmed by observing the relevant topic messages.

Image of the environment configuration on the Raspberry Pi 5.

3.3.3 Configuration of the Ground Control Station

The configuration performed for the ground control station using QGroundControl and Mission Planner tools is detailed below. Both applications offer similar functionality, allowing for flight parameter visualization and management. However, since ArduPilot was used as the firmware, Mission Planner was prioritized due to its optimization for this system.

- **Installation of Mission Planner and QGroundControl:** Both tools were installed on the central computer for managing and monitoring the Pixhawk. Mission Planner was primarily used for its direct compatibility with ArduPilot, while QGroundControl was helpful for some initial configurations and redundant calibrations.
- **Sensor Calibration:** Sensor calibration for the Pixhawk was performed using Mission Planner, following these steps:
  - Access the calibration section in the *Initial Setup* tab.
  - Select the calibration option for each sensor:
    - **Accelerometer:** The Pixhawk was placed in different orientations as instructed on-screen to ensure proper calibration.
    - **Gyroscope:** The Pixhawk was kept stationary during the calibration process.
    - **GPS:** Satellite reception was verified, and necessary parameters were adjusted for accurate positioning.
- **Communication Parameters:** To establish communication between the Raspberry Pi and the Pixhawk via MAVROS and MAVLink, the following parameter adjustments were made using Mission Planner:
  - `SERIAL2_PROTOCOL`: Set to 2 to enable MAVLink.
  - `SERIAL2_BAUD`: Set to 921600 to match the Raspberry Pi configuration.
  - `SYS_ID`: Set to the corresponding ID to ensure proper system communication.

3.4 Vision System

3.4.1 Specifications and Requirements

The vision system must meet the following requirements to ensure accurate detection of Aruco markers:

- Obtain the intrinsic and extrinsic calibration matrices of the camera.
- Generate and detect Aruco markers of various sizes in real time.

3.4.2 Camera Calibration

The camera calibration process was carried out using a Python script with the OpenCV library. This procedure is detailed below, combining code snippets and the images obtained at each stage. The full code can be found in the appendices of the document.

1. **Image Capture:** Multiple images of the chessboard were collected from different angles and distances to cover the entire field of view of the camera. These images exhibited inherent lens distortions, as shown in Figure 3.24.

2. **Corner Detection:** The script detected the internal corners of the chessboard using the `cv2.findChessboardCorners` function. The detected corners were then refined with `cv2.cornerSubPix`, and the calibration images were overlaid with the detected lines, as shown in Figure 3.25.

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

3. **Camera Calibration:** Using `cv2.calibrateCamera`, the camera's intrinsic parameters and distortion coefficients were calculated. These parameters allow for the correction of distortions in the captured images. Figure 3.26 shows the result after applying these parameters.

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

Where:

- `mtx`: Camera intrinsic matrix.
- `dist`: Distortion coefficients.
- `rvecs`, `tvecs`: Rotations and translations.

4. **Calibration Error Calculation:** The accuracy of the model was verified by calculating the average error using `cv2.projectPoints`. This calculation compares the detected and projected corners.

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

5. **Parameter Storage:** The obtained calibration parameters were saved in a JSON file for reuse in future image correction processes.

```
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 Generation of ArUco Markers

Custom ArUco markers were generated for the system, including specific sizes and patterns. In particular, embedded markers were created, where an inner marker is embedded within an outer marker to maximize detection accuracy. The step-by-step process is described below, highlighting important parts of the code used. The full code can be found in the appendices of the document.

1. **Loading the ArUco Dictionary:** The predefined `DICT_6X6_250` dictionary from OpenCV was used, suitable for generating markers with different patterns and levels of detail.

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

2. **Generation of the Outer Marker:** A size of 220 pixels (`outer_marker_size`) was defined for the outer marker, and it was generated with a specific ID (`outer_marker_id`). This allowed for creating a large marker to serve as a container.

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

3. **Generation of the Inner Marker:** The inner marker was generated with a size of 50 pixels (`inner_marker_size`) and a specific ID (`inner_marker_id`). This marker was designed to be embedded within the outer marker.

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

4. **Adding a White Border Around the Inner Marker:** A one-pixel white border was added around the inner marker using `cv2.copyMakeBorder`, which facilitates embedding and improves visibility under varying lighting conditions.

```
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  # White
)
```

5. **Embedding the Inner Marker into the Outer Marker:** The central position was calculated to embed the bordered inner marker into the outer marker. The outer marker was modified to include a black background in the center before inserting the bordered inner marker.

```
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. **Saving the Embedded Marker:** Finally, the generated marker was saved as a PNG image for later use.

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

3.4.4 Real-Time Detection of e-ArUco Markers

The system was designed to detect ArUco markers in real-time using images captured by a camera attached to the drone. This process was carried out via a ROS 2 node that processes images and estimates the pose of the markers relative to the camera. The full code is included in the appendices of the document. Below is the step-by-step description of the system:

1. **Image Publishing from the Drone:** The companion computer on the drone publishes compressed images captured by the camera to a ROS 2 topic (`/camera_image/compressed`). These images are sent to the ground computer for processing.
2. **Image Reception on the Ground Computer:** A node on the ground computer subscribes to the images published by the drone and performs the following tasks:
   - Converts the compressed images into matrices for processing with OpenCV.
   - Resizes the images and converts them to grayscale to optimize detection.
3. **Aruco Marker Detection:** The OpenCV library is used to detect the ArUco markers present in the image. The steps include:
   - Using the `DICT_6X6_250` dictionary to identify specific markers.
   - Calculating the corners of the detected markers using `aruco.detectMarkers`.
   - Drawing the detected markers for visualization.
4. **Pose Estimation:** For each detected marker, its position and orientation in space relative to the camera were calculated using the `aruco.estimatePoseSingleMarkers` function, which returns:

- `tvec`: Translation vector (x, y, z) in meters.
- `rvec`: Rotation vector, converted to Euler angles (*roll*, *pitch*, *yaw*) using rotation matrices.

5. **Visualization and Publication of Results:** The obtained results, including the pose of the markers, were visualized in real-time and published to a new ROS 2 topic (`/aruco_detection/compressed`). The processed images included:
   - Detected markers with highlighted corners.
   - Projected X, Y, and Z axes showing each marker's orientation.

6. **Position Corrections:** Based on the `tvec` values, the necessary corrections were calculated to adjust the drone's position relative to the marker, including lateral (x), vertical (y), and depth (z) movements.
7. **System Interaction:** The results of ArUco marker detection were used to visualize the drone's position and orientation in real-time, enabling the control system to adjust these parameters for landing on the charging station.

3.5 Communication System

3.5.1 Specifications and Requirements

The communication system between the drone and the charging station was designed to ensure efficient, real-time transfer of critical data. The main requirements are:

- **WiFi-Based Local Network:** Provide effective communication between devices without requiring Internet access.
- **ROS 2 DDS Middleware:** Use ROS 2's node and topic architecture to manage data communication between connected devices.

3.5.2 Communication Structure

WiFi Local Network

The local network was created using a WiFi router that connects all devices in the system, including the drone, the charging station, and the ground computer. This configuration eliminated the need for Internet access, providing a secure, isolated environment for data transmission. The drone, equipped with a companion computer, transmitted data via ROS 2 topics, such as compressed images and status messages, to the charging station and ground computer.

Communication via ROS 2

The system used ROS 2's node and topic architecture to manage communication between devices. Each device in the network performed specific functions related to publishing and subscribing to relevant data:

- **Drone:** Published camera images (`/camera_image/compressed`) and status data related to battery, position, and other critical parameters.
- **Ground Computer:** Contains an interface that subscribed to topics published by the drone to:
  - Process images and calculate the position of ArUco markers.
  - Display drone status data, such as battery level and flight mode, in real-time.
  - Publish commands to the charging station, such as opening or retracting the drawer, via specific topics.
- **Charging Station:** Subscribed to commands sent from the ground computer's interface to execute necessary actions, such as opening and closing the drawer.

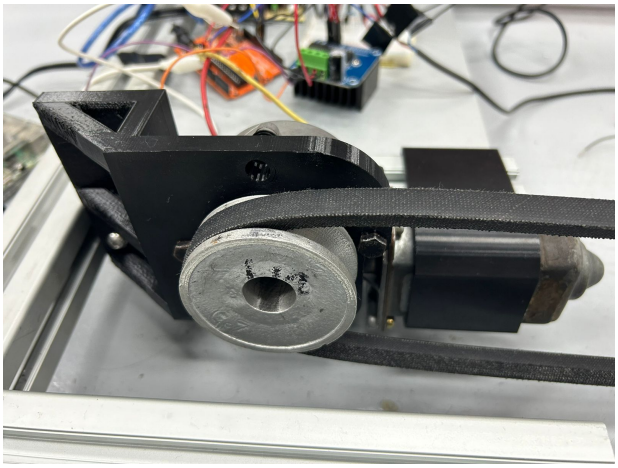


Figure 3.17: View of the internal drawer movement mechanism, showing the 3D-printed motor mounts and the motor with the pulley and belt integrated into the system.

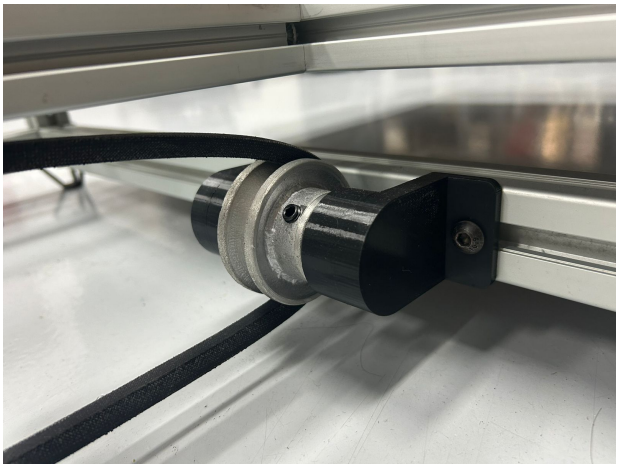


Figure 3.18: View of the internal drawer movement mechanism, showing the 3D-printed bearing housings with...

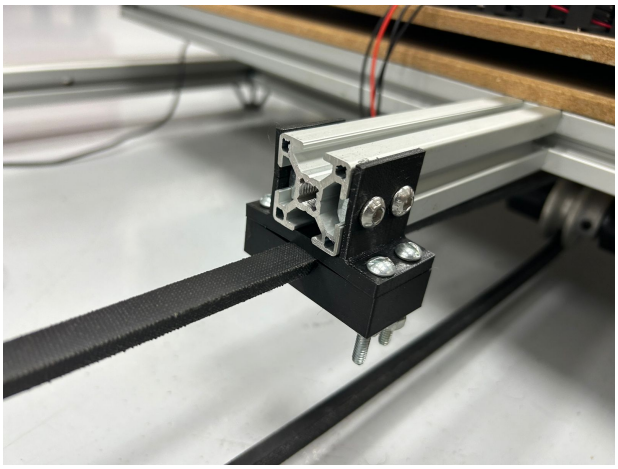


Figure 3.19: View of the internal drawer movement mechanism, showing the 3D-printed holder designed to secure the belt to the internal drawer.

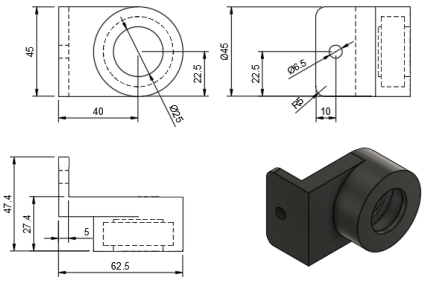
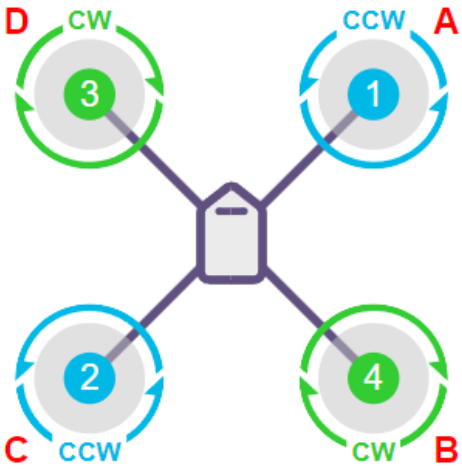
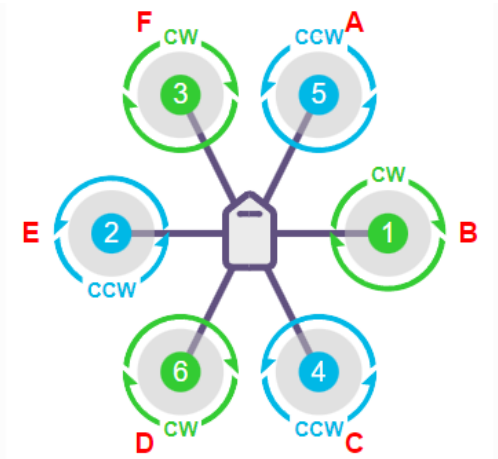


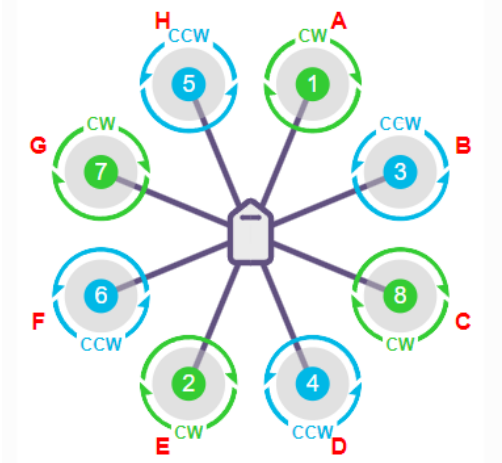
Figure 3.20: Blueprint of bearing housings for the internal drawer movement mechanism.



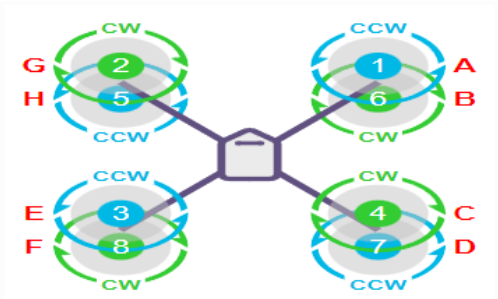
(a) Quad X Frame.



(b) Hexa X Frame.



(c) Octo X Frame.



(d) Octo Quad X Frame.

Figure 3.21: Drone configurations.

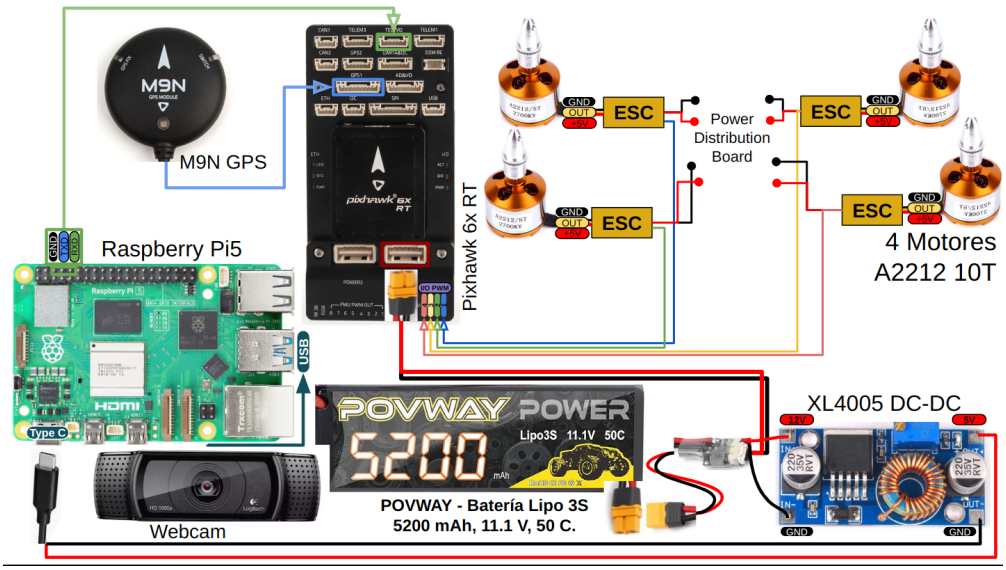


Figure 3.22: Diagram of the drone's power distribution circuit.

Figure 3.23: Image of sensor configuration in Mission Planner.

Figure 3.24: Example of an uncalibrated image captured during the process.

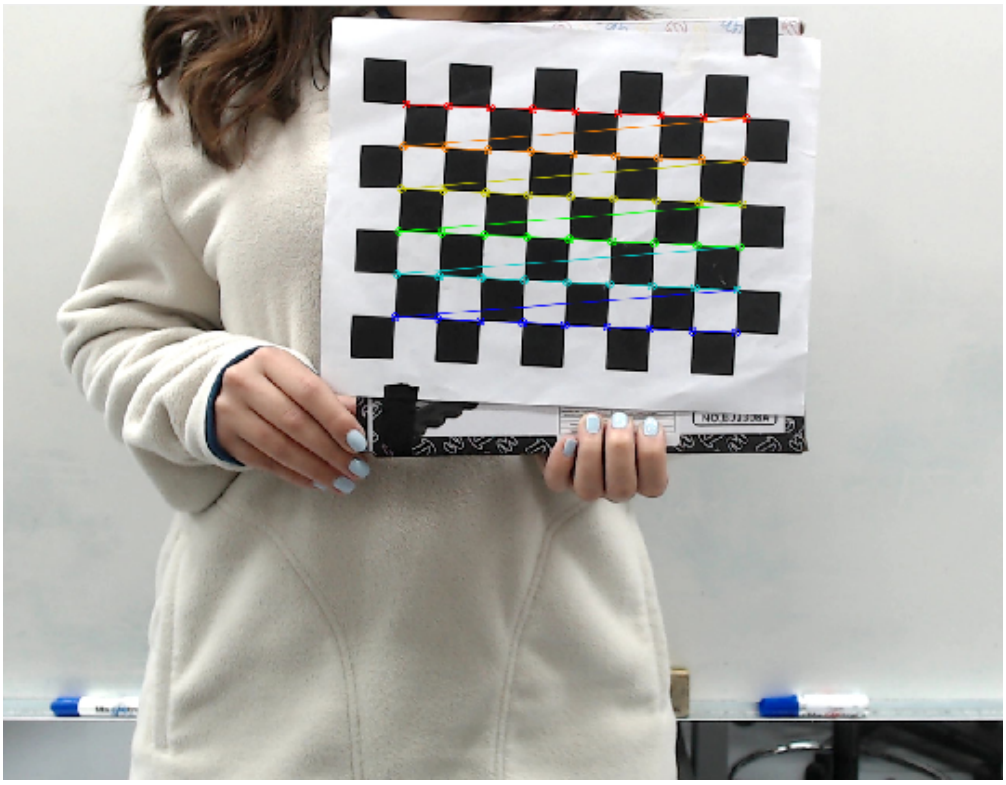


Figure 3.25: Detection and refinement of chessboard corners.

Figure 3.26: Calibrated image after applying the obtained parameters.

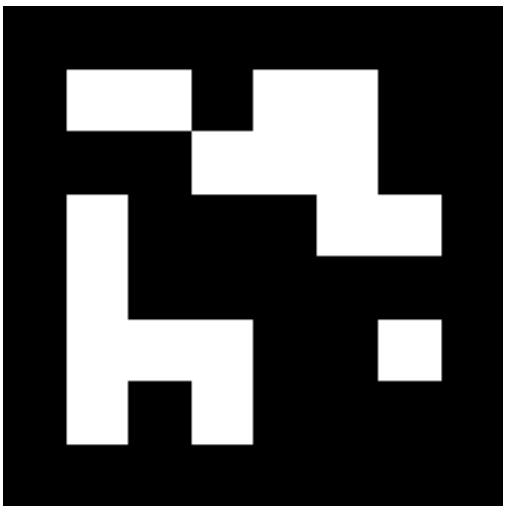


Figure 3.27: Generated outer ArUco marker.

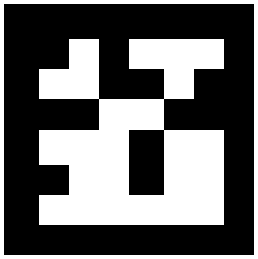


Figure 3.28: Inner marker with added white border.

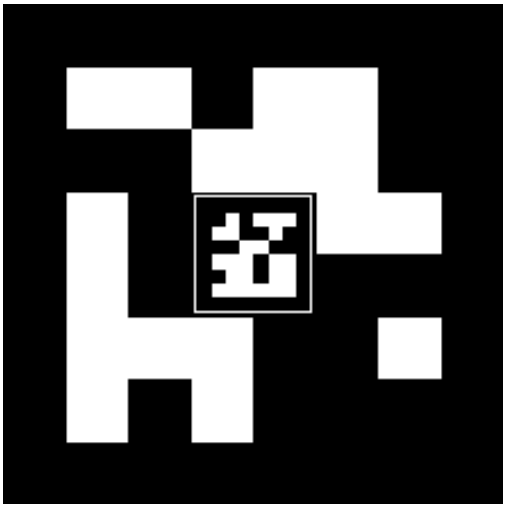


Figure 3.29: Generated embedded ArUco marker.

Figure 3.30: Detection of ArUco markers in the processed image.

Figure 3.31: Graphical representation of the calculated position and orientation of an ArUco marker.

Figure 3.32: Image with detected markers and projected axes.

Figure 3.33: Diagram of the WiFi local network used in the system.

Figure 3.34: Communication diagram between devices using ROS 2.

Chapter 4

Results

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 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

Conclusions

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] ISO/IEC/IEEE 24765:2017, "Systems and Software Engineering Vocabulary," International Organization for Standardization, 2017.

[4] 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*].

[5] Ubuntu Documentation, "Designed for security, reliability, and ease of use." Disponible en: `https://ubuntu.com`. [*Último acceso: mes, año*].

[6] 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*].

[7] 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*].

[8] 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*].

[9] ArUco Documentation, "ArUco is an OpenSource library for detecting squared fiducial markers in images." Disponible en: . [*Último acceso: mes, año*].

[10] 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].

[11] 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).

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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*].

[20] 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.

[21] 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*].

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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*].

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

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

[36] 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.

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

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

[39] 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*].

[40] 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.

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

[42] 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*].

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

[44] 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*].