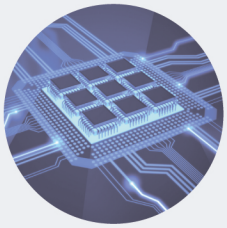
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA DE SISTEMAS COMPUTACIONAIS CRÍTICOS



Authority Handover Procedure and Safety Decision Strategy in Unmanned Aerial Vehicles

GLEIZIELLY ALVES FERREIRA
outubro de 2023

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Authority Handover Procedure and Safety Decision Strategy in Unmanned Aerial Vehicles

Gleizielly Ferreira

Dissertation submitted in partial fulfilment of the requirements for the Master's degree in Critical Computing Systems Engineering

Supervisor: Dr. Eduardo Tovar
Co-Supervisor: Dr. Sérgio Penna

Evaluation Committee:
President:
Luis Miguel Pinho, ISEP

Members:
David Pereira, ISEP
Eduardo Tovar, ISEP

Porto, October 3, 2023

Statement of Integrity

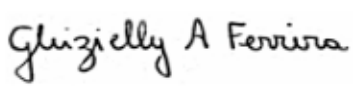
I hereby declare having conducted this academic work with integrity.

The work presented in this document is original and authored by me, and performed in the scope of the Master's degree in Critical Computing Systems Engineering.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration, all references have been acknowledged and fully cited, and all text was originally produced by me (except when duly noted).

I further declare that I have fully followed the Code of Good Practices and Conduct of the Polytechnic Institute of Porto.

Porto, Porto, October 3, 2023



Abstract

Over recent years, Unmanned Aerial Vehicles (UAVs) applications have become popular in different areas, such as aerial image acquisition, agriculture, inspection and maintenance, mapping, and delivery services. Some of these services require the ability to fly UAVs Beyond Visual Line of Sight (BVLOS) to cover greater distances.

Data provided by onboard instruments control the BVLOS operation. The flight controller is responsible for directing the drone flight by controlling the motor's speed and gathering sensor data. The relevant information about the aircraft, such as position, altitude, speed, and direction of flight, are transmitted via a radio link that informs an operator or a Ground Control Station (GCS).

In some drone architectures, there is also an extra computer known as a companion computer or mission computer. They are responsible for providing more intelligence to the flight controller by changing flight parameters. The tasks running on a companion computer can add the capacity to make intelligent decisions during autonomous flight or emergencies, for instance, when the drone loses the radio link with GCS. In addition, for complex drone operations in larger coverage areas, it is necessary to transfer wireless communication links from one access point to another without experiencing connectivity loss. This procedure is known as Handover, and there is much research in this area.

Therefore, studies in this field are still needed to find better solutions to avoid failures and increase public and regulatory acceptance of BVLOS operations with UAVs. In this context, the thesis intends to address solutions to the security authorization handover procedure and addresses security strategies in case of a loss of connection.

Keywords: Unmanned Aerial Vehicles, Beyond Visual Line of Sight, Authority Handover

Resumo

Durante os últimos anos, a utilização de Veículos Aéreos Não Tripulados (VANTs) vem se tornando popular em diferentes áreas como agricultura, inspeção e manutenção de estruturas, mapeamento e serviços de entregas. Alguns desses serviços exigem a capacidade de voar VANTs além do campo de visão (BVLOS) para cobrir distâncias maiores.

Os dados fornecidos pelos instrumentos de bordo controlam a operação BVLOS. O controlador de voo é responsável por direcionar a aeronave controlando a velocidade dos motores e coletando dados dos sensores. Alguns dados do veículo, como posição, altitude, velocidade e direção do voo, são transmitidos por meio de um link de rádio que informam operadores ou uma estação de controle em solo.

Em algumas arquiteturas de VANTs, há também um dispositivo extra conhecido como computador complementar ou computador de missão. Eles são responsáveis por fornecer inteligência ao controlador de voo devido a maior capacidade de processamento. As tarefas executadas em um computador de missão podem agregar a capacidade de tomar decisões inteligentes durante o voo autônomo ou em situações de emergências, por exemplo, quando o veículo perde a conexão com a estação de controle em solo.

Para operações de VANTs em áreas de maior cobertura, é necessário transferir a conexão de comunicação sem fio de um ponto de acesso para outro sem que o veículo experiencie perda significativa de conectividade. Esse procedimento é conhecido como handover, e há muita pesquisa nessa área. Entretanto, ainda existem preocupações em torno das atuais tecnologias em termos de confiabilidade de comunicação, cibersegurança e controle autónomo.

Portanto, estudos nesta área ainda são necessários para encontrar melhores soluções para evitar falhas e aumentar a aceitação pública e regulatória das operações BVLOS com VANTs. Neste contexto, a tese aborda soluções para o procedimento de transferência de autorização do controle do veículo e aborda estratégias de segurança em caso de perda de conexão com a estação de controle.

Acknowledgement

This thesis would not have been possible without the support of some people. I can't let away this opportunity without naming some more impactful ones.

Firstly, I express my gratitude to Professor Dr. Ênio Filho for inviting me to join CISTER and for inspiring my academic journey since graduation. "Primeiro acenda um LED, depois domine o mundo." his fun phrases still resonate and motivate my pursuit of knowledge.

I also want to thank my advisor, Dr. Sérgio Penna, for his support and guidance throughout this research journey. His expertise, patience, and insightful feedback are essential to completing this thesis. I would also like to thank my colleague, RúbeN Cruz, for his support and valuable insights during this project.

I want to thank everyone at CISTER, particularly my supervisor, Dr. Eduardo Tovar, for their support and for providing the necessary resources and facilities for conducting this research.

A special sign of gratitude to Márcia Rocha, a friend from graduation, master, and for life, who has been a source of support throughout this time. I couldn't go through all these challenging moments without her by my side.

Lastly, I want to acknowledge the most significant support from my family, particularly my parents and my sister. Their love, constant encouragement, and support have motivated me despite the distance. Without their incredible efforts throughout their lives, none of what I am and aspire to become would have been possible.

Thank you all for being part of this journey!

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDP/UIDB/04234/2020) by FCT and the EU ECSEL JU under the H2020 Framework Programme, within project ECSEL/0010/2019, JU grant nr. 876019 (ADACORSA); The JU receives support from the European Union's Horizon 2020 research and innovation program and Germany, Netherlands, Austria, France, Sweden, Cyprus, Greece, Lithuania, Portugal, Italy, Finland, and Turkey. The ECSEL JU and the European Commission are not responsible for the content of this paper or any use that may be made of the information it contains.

Contents

List of Figures **xiii**

List of Source Code **xv**

List of Acronyms **xvii**

1 Introduction **1**
1.1 Overview . . . 1
1.2 Research Motivation and Context . . . 1
1.3 Research Objectives . . . 2
1.4 Contributions . . . 2
1.5 Thesis Organization . . . 3

2 Unmanned Aerial Systems **5**
2.1 Unmanned Aerial Vehicles . . . 5
2.2 Beyond Visual Line of Sight Operations . . . 6
2.3 Authority Handover Procedure . . . 8

3 Simulation technologies **9**
3.1 Dronecode Foundation . . . 9
3.1.1 PX4 Autopilot . . . 9
3.1.2 MAVLINK . . . 10
3.1.3 MAVSDK . . . 10
3.1.4 QGroundControl . . . 10
3.2 Gazebo . . . 11
3.3 Ardupilot . . . 11
3.4 MavProxy . . . 12
3.5 PixHawk . . . 13
3.6 Cube Orange . . . 13
3.7 Jetson Nvidia . . . 14

4 Authority Handover Procedure **17**
4.1 Application Scenario . . . 17
4.2 System Architecture . . . 18
4.3 Mission Computer . . . 18
4.3.1 On Load . . . 19
4.3.2 On Mission . . . 20
4.3.3 On Handover . . . 21
4.4 Ground Control Stations . . . 21
4.5 Messages Sequence . . . 25
4.6 Handover Region . . . 28

4.6.1 Circular Geofence 28
4.6.2 Mission Waypoint Detection 30
4.6.3 Polygonal Geofence 31
4.7 Safety and Security 33

5 Software and Hardware Architecture **35**
5.1 Testbed Architecture 35
5.2 Flight Controller Integration 38
5.3 Integration with Adacorsa Partners 40

6 Evaluation and Results **43**
6.1 Simulation Results 43
6.1.1 Authority Handover 44
6.1.2 Authority Check 47
6.1.3 Emergency Procedure 49
6.2 Flight Controller Integration Results 50
6.3 Integration Results with Adacorsa Partners 54

7 Conclusion and Future Work **57**

Bibliography **59**

List of Figures

2.1 Operation Categories 7

3.1 QGroundControl: Vehicle parameters configuration 11
3.2 Cube Orange 14
3.3 Jetson NVIDIA 15

4.1 Authority Handover Procedure 17
4.2 System Architecture 18
4.3 Mission Computer: Main Function 19
4.4 Mission Computer: Server Messages 20
4.5 Mission Computer: Handover 21
4.6 Ground Control Stations main function 22
4.7 Ground Control Stations Commands 23
4.8 Ground Control Stations Messages 24
4.9 Message Sequence diagram 27
4.10 Circular Geofence 28
4.11 Circular Geofence Limitation 30
4.12 Mission Waypoint Detection 31
4.13 Polygonal Geofence 32

5.1 Simulation Setup 35
5.2 Simulation Softwares 36
5.3 GAIA 160MP with a Cube orange 38
5.4 Hardware Setup 39
5.5 Hardware Integration with Adacorsa Partner 41

6.1 Mission Plan 43
6.2 Start Mission Command 44
6.3 Authority Handover Messages 45
6.4 Authority Handover Messages Sequence 46
6.5 Authority Handover Messages Sequence 47
6.6 Authority Check GCSA 48
6.7 Authority Check GCSB 49
6.8 Emergency Procedure 50
6.9 GCSA and Mission Computer: Delivery flight Handover 51
6.10 GCSA and Mission Computer: Delivery flight Handover 52
6.11 GCSA and Mission Computer: Return Flight Handover 53
6.12 Integration Adacorsa Partner Results:Drone 54
6.13 Integration Adacorsa Partner Results 55
6.14 Integration Adacorsa Partner: Log Mission Computer 56

List of Source Code

4.1 Handover Procedure in the Mission Computer 25
4.2 Haversine Function. 29
4.3 Point in Polygn Function. 32
5.1 Terminal Commands to start simulation 36
5.2 Flight Data Acquisition using MAVSDK 37
5.3 Multlink Configuration Script 39
5.4 Service File on Linux 40
5.5 Enable Service 40
5.6 Serial Communication Setup between Mission Computer and Flight Controller 40
5.7 Data Struct from TCP Provider 41
5.8 Flight Data Acquisition using TCP/IP 41

List of Acronyms

BVLOS	Beyond Visual Line of Sight.
C2	Command and Control.
EASA	European Union Aviation Safety Agency.
EVLOS	Extended Visual Line of Sight.
FC	Flight Controller.
GCS	Ground Control Station.
GPS	Global Positioning System.
IMU	Inertial Measurement Unit.
MAVLink	Micro Air Vehicle Link.
MC	Mission Computer.
QGC	QGroundControl.
UAS	Unmanned Aircraft System.
UAV	Unmanned Aerial Vehicle.
VLOS	Visual Line of Sight.

Chapter 1

Introduction

1.1 Overview

Unmanned Aerial Vehicles (UAVs) technologies introduce several advantages in both the industrial and academic fields. The vehicles are characterized by high maneuverability, a wide variety of usage, and low cost. As a result, UAVs have become widespread in various fields, such as the agricultural sector (Rejeb et al. 2022), delivery logistics (Benarbia and Kyamakya 2022), military and medical (Ayamga, Akaba, and Nyaaba 2021).

Given the number of applications for UAVs, it is necessary to have an integrated embedded system to deal with all required tasks. It includes a set of sensors, such as the Inertial Measurement Units (IMUs), to specify the orientation and stability, and the Global Positioning System (GPS) for positioning. Other systems can also be integrated to perform specific tasks, such as a radar system and cameras to collect data or to detect and avoid obstacles (Bigazzi et al. 2022).

Elmeseiry, Alshaer, and Ismail 2021 investigated and classified the Unmanned Aircraft System (UAS) accornding to applications. They also reviewed future directions in industry/research interest and the challenges and limitations of UAVs, such as battery charging, collision avoidance, and security. Politi et al. 2021, in their work, reviewed the technologies to enable Beyond Visual Line of Sight (BVLOS) applications. They concluded that autonomous flight capability and communication are paramount to BVLOS operations in UAS.

Nonetheless, studies in this field are still needed to find better solutions to avoid failures and increase public and regulatory acceptance of BVLOS operations with UAVs. The necessity for constant communication links and the vehicle's ability to change control authority between ground stations while moving is a challenge that must be investigated. Given what has been said, this work aims to increase the autonomous capacity of an unmanned aerial vehicle by adding an extra computer to handle the challenges of the BVLOS operations.

1.2 Research Motivation and Context

The use of UAV in Beyond visual line of sight (BVLOS) operations is attractive to the industry because it can increase the efficiency of drone flights by reducing the need for a visual observer and enabling the vehicle to fly further distances. These can save time and resources in the agriculture, energy, and delivery industries. Also, BVLOS flights allow more comprehensive data collection, especially in areas that are difficult to access by ground vehicles. As a result, it can be helpful in industries such as natural resource management, disaster response, and infrastructure inspection. In addition, BVLOS flights can help reduce

the risk of accidents and injuries by allowing drones to fly over hazardous or difficult-to-reach areas without putting human operators in harm's way (Hartley, Henderson, and Jackson 2022).

However, as the authors explained in (Matalonga et al. 2022), the BVLOS operation lacks the technical capabilities to ensure that the vehicle does not threaten public safety. In addition, there are restricted regulations for BVLOS drone operations because of concerns about safety, privacy, and security. Thus, this thesis contributed by improving safety operations in large coverage areas through the support of multiple communication ground links and safe authority handover procedures. Additionally, the thesis increased the autonomous capacity by adding a mission computer on board the vehicle to handle drone behavior.

This thesis has been funded by Research Centre in Real-Time and Embedded Computing Systems (Cister 2023), a participant in the Adacorsa (Airborne Data Collection on Resilient System Architectures) project. The Adacorsa project aims to strengthen the European drone industry and increase the acceptance of BVLOS operations by developing and demonstrating technologies for safe, reliable, and secure drone operations (Adacorsa 2023).

1.3 Research Objectives

The main objective presented in this thesis is to review the most recent literature, practices, and regulations and implement a simulation of the complete process of the UAV authority handover between two ground control stations. Besides that, the thesis pretends to increase the vehicle's autonomous capability by adding an extra processor unit to handle unexpected situations. In particular, the thesis delivers:

- A systematic review of the literature;
- Algorithm for an authorization handover procedure;
- Discussion for an emergency procedure;
- Configuration of the simulators;
- Configuration of the Mission Computer;
- Execution hardware simulation;
- Documentation of all the results obtained.

1.4 Contributions

In this thesis, the main contributions are:

- Overview of Unmanned Aerial Vehicles architectures;
- Overview of different drone operations, focusing on the challenges of Beyond Visual Line of Sight (BVLOS) operations;
- Overview of handover techniques for BVLOS operations;
- Overview of simulation technologies for Unmanned Aircraft Systems (UAS);
- Implementation of an authority handover procedure;
- Integration with Adacorsa partner projects;

- Validation of authority handover procedure in simulations and real-world scenarios;
- Demonstration of requirements from the Adacorsa project.

1.5 Thesis Organization

The present text is composed of seven parts. Chapter 1 presents the thesis' theme by describing and contextualizing the problem and enumerating the objectives this work aims to achieve. In chapter 2, there is a brief review of recent works that have motivated this thesis. In chapter 3, the chapter presents the simulation tools and other technologies that supported the thesis development. Chapter 4 describes the proposed solution and strategies to implement the authority handover procedure. Also, it highlights the progress made in this research regarding the Handover Region selection and provides an overview of message safety and security and insights into emergency procedures. Chapter 5 describes the technical approach to implement the authority handover procedure. Whereas Chapter 6 presents the results obtained. Finally, Chapter 7 discusses the results and thoughts about future work and developments related to these projects.

Chapter 2

Unmanned Aerial Systems

This chapter scans recent works that have motivated this thesis. Section 2.1 clarifies some concepts, defines some components in the UAV architecture, and discusses some communications and drone networks. Section 2.2 analyses the challenges and limitations and discusses, from a legal point of view, the regularization of drones in the BVLOS operations. Finaly, Section 2.3 shows handover techniques already studied in the last years.

2.1 Unmanned Aerial Vehicles

Unmanned Aerial Vehicles (UAVs) are aircraft designed to be piloted remotely or preprogrammed to fly specific routes. In both cases, the interoperability of multiple systems is expected to ensure safe flight operations. According to European Union Aviation Safety Agency (EASA), an UAS refers to UAV and all apparatus to control it remotely. It includes the vehicle system, the person or team on the ground controlling the flight, and the devices necessary to connect both, such as ground control systems, transmission systems, and software.

The several applications justify the different compositions of a UAS depending on their purpose. Typically it consists of several key components that together perform the designed task. For example, the propulsion system is composed of motors, propellers, and batteries that provide the power and thrust necessary for flight. The sensors, such as GPS, accelerometers, gyroscopes, and magnetometers, provide information on the drone's position, orientation, and movement. The Flight Controller (FC) is the central processing unit that collects the data from all these devices and uses them to manage and controls the drone's flight and navigation, guaranteeing part of its autonomy.

The Ground Control Station (GCS) comprises ground-based hardware components and software that can maintain the Command and Control (C2) link to the onboard system. The ground control can also enable operators to plan missions onto a UAV and view incoming sensor data. Since real-world operations are unpredictable and prone to non-deterministic behaviors, maintaining a reliable Command and Control link is fundamental for the safe operation of the UAV.

The Communication System comprises radio transceivers, antennas, and other components that allow the vehicle to transmit data to ground systems or be controlled remotely. The drone also sends flight data and telemetry back to the ground controller over the Wi-Fi link, Mobile communication such as 4G/5G/LTE, and Zigbee. These are some of the most commonly used communication protocols for drones, and the choice of protocol depends on

the specific requirements of the UAV system, including the range, data transfer rate, and reliability needed for the mission (Noor et al. 2020).

In their work, Neji, Mostfa, and Sebbane 2019 analysed the most suitable radio communication system between a UAV and a GCS in two different application scenarios. First, they identify some relevant criteria that cover most use cases, such as communication range between the vehicle and the pilot, energy consumption, infrastructure cost, security, throughput, and latency. In addition, they investigated emergent technologies such as 5g and LP-WAN. Also well-consolided tecnologies such LOra, Wi-fi. They conclude that the 5G will provide good performance, like low power consumption and high-security levels. Although LP-WAN technologies scores are likely to increase soon, considering they are still under development.

Some drone architectures have different devices designed to operate a specific task. The payloads are extra weight that can be transported in the UAV. It can be equipment such as cameras, additional sensors, tools, armaments that the drone carries for specific missions, or other determined task-specialized devices. For example, the mission computer, also called a companion computer, can increase the drone's processing capacity. It can include processing images, pre-processing information from advanced sensors, Detect and Avoid Systems, or actuating auxiliary motors. Besides, the mission computer can unlock the potential of autonomous flights by increasing the drone's capacity to make intelligent decisions during flight. All these components work together to allow UAVs to perform various missions, from aerial photography and surveying to delivery and military operations or unlock the drone's capacity to autonomous flight.

2.2 Beyond Visual Line of Sight Operations

The European Union Aviation Safety Agency (EASA 2022) defines three operational categories based on UAVs' range: Visual Line of Sight (VLOS), Extended Visual Line of Sight (EVLOS), and BVLOS. Figure 2.1 illustrates all three types of UAV operation. In practice, VLOS operations allow the remote pilot to control the aircraft's flight path in his visual range, typically up to a few kilometers. The remote pilot must maintain continuous unaided visual contact with the aircraft to avoid collisions with other aircraft, people, and obstacles.

In EVLOS operations, the UAVs are operated beyond the operator's visual range but aided by technological means or with one or more visual observers who keep the UAV in sight and communicate with the remote pilot. This category typically covers UAVs that are operated up to a few tens of kilometers from the operator. The remote pilot has direct control of the UAS at all times and must maintain uninterrupted situational awareness of the airspace in which the operation is being conducted.

In contrast, in the BVLOS operations, the UAVs are operated beyond the operator's and observer's visual range. Consequently, BVLOS operations are complex and risky and require advanced technologies, such as autopilot software, communication systems, and real-time data links, to ensure safe and accurate flight.

The BVLOS operation has many application scenarios where the operator needs to fly the UAV over long distances or out of sight, such as delivering packages, agriculture, surveying, and inspection. However, strict regulations for BVLOS operations have been established to ensure safety and prevent accidents.

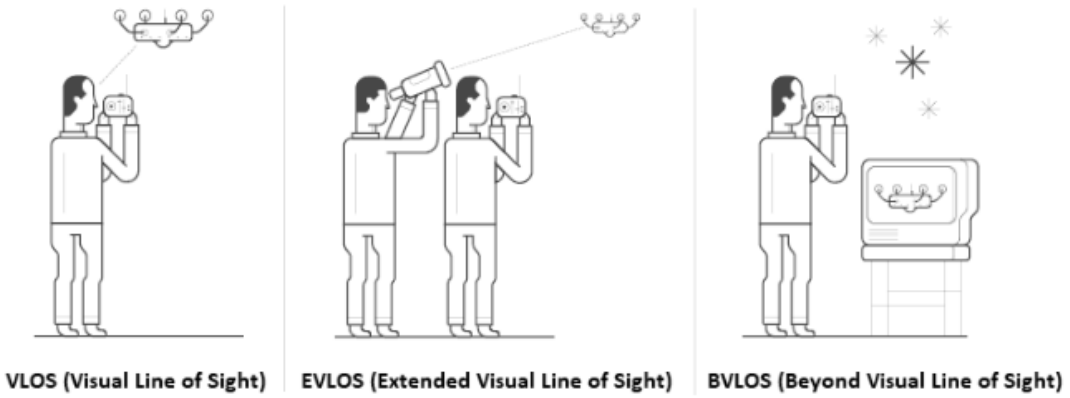


Figure 2.1: Operation categories regulated by EASA (from www.revistapesquisa.fapesp.br(Suzel Tunes 2021))

The Portuguese Civil Aviation Authority (ANAC 2022) considers the BVLOS as a specific category and only allowed with a specific clearance. In addition, it must comply with several requirements, such as geofenced areas and operational risk analysis and mitigation.

In (Matalonga et al. 2022), the authors reviewed the regulatory and technological aspects of unlocking autonomous BVLOS operations of UAVs. Also, they summarised the state-of-the-art quality assurance of critical software-intensive systems. In addition, they presented the RAPID Project, an example use case for Autonomous BVLOS Operations. The project demonstrated the capacity of a swarm of autonomous UAS to survey a bridge in a busy environment. Finally, the authors identify requirements for robust autonomous BVLOS operations. One of these requirements is to establish reliable communication between the UAS and the ground system.

In (Andersen et al. 2021), they tested the MultiRX module, a multiplexer designed to accept multiple connections to the same Flight Controller (FC). The work provided a reliable functionality for BVLOS operations by allowing the control of the vehicle by multiple pilots in different locations. Their project was developed in compliance with the communication requirements for BVLOS operation described in Specific Operations Risk Assessment (SORA), an EASA framework that imposes requirements on UAV operations. However, this work does not explore the vehicle autonomy feature once the tests only were conducted by manual control.

In (Politi et al. 2021), the authors surveyed the main BVLOS challenges for drone operation and the emerging technology requirements. First, they listed the main challenges for BVLOS: Security and Safety, Route Planning and Navigation, Communication, Object Detection, and Collision Avoidance. Further, they identify several key areas to develop. It includes, among other things, ensuring a reliable Command and Control (C2) link and advances in autopilot systems focusing on more intelligent approaches of adaptive control technologies that can sustain a higher degree of autonomy during flight.

Since the BVLOS operation requires sophisticated software, communication systems, and the ability to monitor the UAV's parameters, this work pretends to develop an authority handover procedure to enable a BVLOS operation, considering the safety and reliable communication between the vehicle and the ground system.

2.3 Authority Handover Procedure

In the context of unmanned aerial vehicles (UAVs), a handover (HO) refers to transferring communication links from one network connection to another with minimal loss of quality or connectivity. There are two HO scenarios of a UAV. The horizontal handover occurs when the mobile device switches from one network to another of the same type (e.g., from one Wi-Fi network to another). On the other hand, a vertical handover happens when the mobile device switches from one network technology to another of a different type. For example, switching between different generations of wireless technology (4G, 5G) or different types of networks (e.g., Wi-Fi to cellular). In both cases, the goal of the handover is to provide a seamless connection and reliable communication services during the UAV flight (Shayea et al. 2022, pp. 11–13).

In (Angjo et al. 2021), the authors list several works that deal with the challenges of handover in mobile drone networks. They analysed several works and grouped them based on the approaches. In addition, they also introduced some issues related to drone operations and HO management, such as the development of communications protocols designed considering the drones' mobility characteristics. Finally, they conclude that the integration of drones into existing networks and their mobility issues are the current focus of the literature.

According to (Andersen et al. 2021), handover techniques are essential for Beyond Visual Line of Sight (BVLOS) operations because they allow for maintaining uninterrupted communication and transfer of control of the unmanned aerial vehicle (UAV) from one operator or control station to another during the flight. In the literature, there are several works about the handover procedure. In their survey (Shayea et al. 2022, pp. 16–27), the authors examined and classified related works about handover management in future mobile networks. Besides that, the authors highlighted various research challenges, such as security and mobility management. They affirm that security and safety issues will increase due to drones' massive growth and capability to cover larger areas.

With that in mind, this work implemented an authority handover procedure, a set of steps and procedures followed to transfer a UAV's control and responsibility from one ground entity to another. The authority handover procedure involves verifying the readiness of the receiving entity, communicating the current state of the UAV, establishing a secure and reliable communication link, and confirming the transfer of control before the UAV is handed over.

Chapter 3

Simulation technologies

This chapter presents an overview of some open-source projects that provide a range of tools and resources to support the development of this work. Autopilot simulators are the simulation tools relevant to the scope of this thesis since the main goal was to develop software that mediates the communication between a flight controller and a ground control station.

The two most relevant autopilot simulator candidates for this project are the PX4 from Dronecode Foundation and Ardupilot. These two are arguably the most used and open-source community-supported in the area, with various works related to Unmanned Aerial Systems.

After this, Gazebo will be overviewed since both autopilots designed their modules, allowing integration with this software. Also, two Ground Control Station software will be analyzed. Later sections present the hardware utilized in this project.

3.1 Dronecode Foundation

Dronecode Foundation is an open-source, collaborative platform that provides tools and technologies for developing unmanned aerial vehicle (UAV) applications and solutions. The Dronecode platform is maintained by the Linux Foundation and includes a variety of projects. Its open-source characteristic allows developers to collaborate, contribute, and build on existing projects to create new and innovative solutions (Dronecode Foundation 2023).

Dronecode includes multiple projects, such as flight stacks, ground control stations, and hardware abstraction layers. In addition, Dronecode projects consist of software tools, libraries, protocols, and APIs, that enable developers to create new applications and services UAS both in industrial and academic applications. The following subsections show an overview of the projects supported by the Dronecode foundation that helped the development of the current project.

3.1.1 PX4 Autopilot

PX4 Autopilot project is a flight control software for unmanned aerial vehicles (UAVs). It provides a platform for autonomous flight control, navigation, and mission planning, enabling developers to create custom UAV applications and solutions. Additionally, it is designed to be safe, secure, and reliable, making it a trusted choice for UAV control. PX4 has advanced features, including flight control, sensor management, communication with Ground Control Station (GCS), and navigation and control algorithms (Dronecode 2023c).

PX4 Autopilot supports a wide range of UAV hardware platforms, making it suitable for various applications. In addition, the software is highly modular and configurable, allowing developers to customize it to meet their specific needs.

3.1.2 MAVLINK

Micro Air Vehicle Link (MAVLink) is a lightweight communication protocol widely used in UAS and UAV components. It provides a compact and efficient way of transmitting data between UAVs, ground control stations, onboard sensors, or companions computers. In addition, MAVLink is used to transmit telemetry data, including information about the state of the UAV and its sensors, as well as commands from the ground station to the UAV (Dronecode 2023a).

The protocol is designed to be easy to implement and can be used over various communication links, including serial links, Wi-Fi, and cellular networks. As a result, it is a protocol widely used for GCS-based drone control, running in numerous Autopilot-based systems such as ArdupilotMega, pxIMU Autopilot, SLUGS Autopilot, and of course, PX4 Autopilot (Kwon et al. 2018).

Once MAVLink is designed as a lightweight and efficient communication protocol, messages are not encrypted by default. This is because the process of encryption can add overhead and latency. However, it is possible to transmit MAVLink messages over an encrypted VPN or to encrypt the payload of MAVLink messages using an encryption library.

In (Allouch et al. 2019), the authors discussed the security threats of the MAVLink protocol. They proposed the MAVSec, a security-integrated mechanism for MAVLink which uses encryption algorithms to improve the security of MAVLink messages exchanged in UAS.

3.1.3 MAVSDK

MAVSDK project is a set of libraries in different programming languages that provide an interface for communicating with unmanned aerial vehicles (UAVs) using the MAVLink protocol. The libraries are designed to be platform-agnostic and can be used on various operating systems, including Linux, macOS, and Windows (Dronecode 2023b).

The MAVSDK supports multiple programming languages, including C++ and Python. In addition, it provides a high-level API for accessing UAV information and sending commands, making it ideal for developers who want to create a wide range of applications that interact with UAVs.

In this work, the MAVSDK C++ library was used to obtain real-time data from the flight controller, allowing the implementation of the software of the Authority Handover.

3.1.4 QGroundControl

QGroundControl (QGC) provides a graphical user interface for controlling and monitoring UAVs and managing their missions. QGroundControl (QGC) communicates with UAVs using the MAVLink protocol and supports various hardware devices and autopilots, including the PX4 Autopilot (Dronecode 2023d).

The QGC project is compatible with multiple operating systems like Linux, macOS, Windows, iOS, and Android. It connects to the UAV using MAVLink and reports essential status from the vehicle and parameter information such as battery level, flight mode, and others.

The software allows users to input simple commands or complex, detailed autonomous missions with multiple waypoints in a user-friendly interface. In this project, QGC was used as the graphic interface to create and test the missions in a simulation since it provides advanced features such as flight data analysis, vehicle parameters configuration (Figure 3.1), and offline map support. It was used to set up and execute autonomous missions that defined waypoints, flight paths, and actions for the drone to follow.

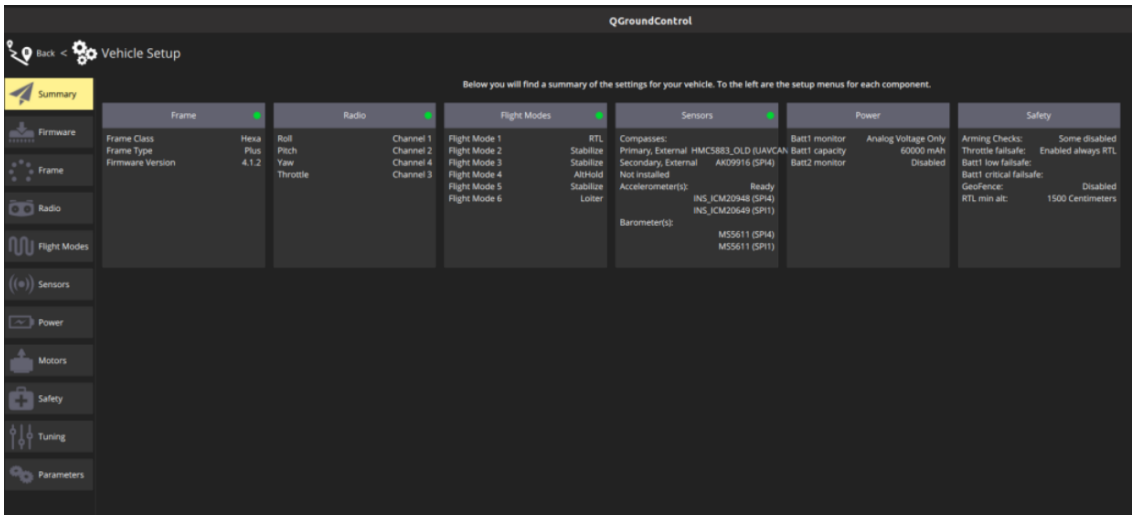


Figure 3.1: QGroundControl: Vehicle parameters configuration

3.2 Gazebo

Gazebo is a set of open-source software frameworks to make developing high-performance applications easier. It provides a flexible and realistic simulation environment that allows developers to test and evaluate the behavior of complex robotic systems, including unmanned aerial vehicles (UAVs) (Open Robotics 2023).

Gazebo provides a comprehensive API for creating custom plugins and simulations, making it a highly flexible platform for robotic simulation to suit many different use cases. For example, the PX4 community contributes to a set of Gazebo plugins necessary to assemble UAV models with different sensor configurations. It includes various sensors and actuators commonly used in UAVs, such as cameras, GPS, IMUs, and additional supplied plugins to communicate with simulated hardware over MAVLink (PX4 2023a).

Gazebo is free, open-source, and supported on all major operating systems, such as Linux, macOS, and Windows. Gazebo is conceived to simulate realistic conditions of the robotic system and the surrounding environment. Its features allow more detailed observation and analysis of the robotic's system's behavior. In addition, it supports robotics industry developers and academic researchers in testing and validating its projects.

3.3 Ardupilot

ArduPilot is an open-source autopilot software suite compatible with various hardware platforms, including Pixhawk[3.5] and Cube Orange[3.6]. It serves as the brains behind autonomous drone operations, providing the essential intelligence and control algorithms.

ArduPilot offers many functionalities, encompassing waypoint navigation, mission load, telemetry, and fail-safe modes (Ardupilot 2023a).

Like PX4, ArduPilot is an open-source autopilot software, but it extends its versatility to support a broad spectrum of unmanned vehicles, including drones, autonomous cars, boats, ground rovers, and submarines. This project focuses on ArduCopter, which specializes in stabilizing and controlling multi-copter drones, such as quadcopters, hexacopters, and octocopters. ArduCopter equips these vehicles with the necessary intelligence and control algorithms, enabling autonomous or semi-autonomous operations.

ArduPilot offers an array of flight modes, including stabilizing, loiter, auto, and more. In this context, we utilize the 'auto' mode for executing autonomous missions. ArduPilot allows vehicles to follow predefined waypoints along their mission paths. Additionally, the software incorporates fail-safe mechanisms to ensure safe operation in signal loss or emergencies, triggering actions like Return-to-Launch (RTL) procedures.

Furthermore, ArduPilot performs pre-configuration and comprehensive hardware checks to confirm the vehicle's readiness for flight. Procedures such as calibration and pre-arm checks permit verifying the correct functionality of motors, servos, and sensors.

ArduPilot has an extensive range of built-in vehicles and is compatible with various external simulators. This versatility allows ArduPilot to be tested in diverse UAS scenarios.

Lastly, ArduPilot benefits from an active open-source community continuously working on software development and enhancements. This community support provide to users access to documentation, tutorials, and third-party tools.

3.4 MavProxy

MAVProxy (Micro Air Vehicle Proxy) is an open-source command-line ground station software designed primarily for controlling and monitoring unmanned aerial vehicles (UAVs) using the MAVLink protocol. It is written in Python, providing flexibility and customization when interacting with UAVs Ardupilot 2023b.

Like other ground control stations, MAVProxy is an intermediary between the operator and the UAV, facilitating communication by relaying MAVLink messages. Its extensibility allows users to create custom Python scripts and modules tailored to their needs. These scripts can interact with the drone, process telemetry data, and implement custom control logic. Users can send MAVLink commands and receive telemetry data through the MAVProxy command-line interface, including commands like takeoff, land, change flight mode, set waypoints, and more. It also offers map viewing and the capacity to send emergency commands, such as pausing a mission or initiating a return to the mission's origin (RTL).

MAVProxy's ability to handle communication with multiple devices simultaneously is important for managing multiple vehicles or ground control. It can also act as a router to send drone data to other ground control stations such as QGroundControl or Mission Planner and mission computers.

MAVProxy is commonly used during UAV software and hardware development and testing phases, providing a means to interact with and monitor the drone's behavior in a controlled environment. To establish a connection with MAVProxy, it is necessary to specify the communication link (e.g., serial port or network connection) and the corresponding serial or IP address.

This project used MAVProxy only to enable real-time data acquisition during simulations. However, due to its architecture based on the command line and the customization capacity, it can also be a useful tool for sending commands from the Mission computer to the flight controller without direct manual intervention.

This work employed MAVProxy for simulation purposes and routing telemetry data from the flight controller to simulation tools. Meanwhile, QGroundControl served as a graphical interface for map visualization and mission planning.

3.5 PixHawk

The Pixhawk system is an open-source hardware platform for UAV systems development. It integrates a range of hardware components, including microcontrollers, sensors, communication interfaces, and power management systems, in a compact and modular design. Pixhawk open standards provide guidelines and mechanical and electrical specifications for UAV control and navigation (Lorenz Meier 2023).

Pixhawk began as a project in research of autonomous flight idealized by Lorenz Meier in 2008. Since 2014, The platform has been maintained and developed by the Dronecode community, and new features and improvements are regularly added to its repository. Its main goal is to integrate and maintain hardware components, helping UAV application developers reduce costs and time of designing and consequently focusing more on the application's functionality (Auterion 2023).

Pixhawk offers many features and capabilities for controlling and navigating UAVs. It supports various communication protocols, including MAVLink, and can be easily integrated with other UAV components and systems, such as ground control stations and mission computers. It is designed to provide a reliable and flexible platform for various UAV applications, including hobby and commercial drone projects and academic research.

3.6 Cube Orange

The Cube Orange autopilot is an integral component within a comprehensive ecosystem of autopilot modules and carrier boards, forming part of the Cube series of flight controllers. Designed for unmanned aerial vehicles (UAVs), this versatile device boasts a reputation for exceptional reliability. Cube Orange is equipped with redundant sensors and power supplies to ensure uninterrupted operation in the face of potential sensor or power source failures (Ardupilot 2023c).

This autopilot is the core hardware responsible for stabilizing and controlling UAV flight. Its compatibility extends to a broad spectrum of UAV types, ranging from multicopters such as hexacopters to fixed-wing aircraft, making it suitable for various applications. Cube Orange can operate on the foundation of open-source flight control software such as PX4 or Ardupilot, offering users the flexibility to customize and tailor the flight controller to their specific requirements. This open-source nature provides full access to the software code, empowering users to enhance, modify, and actively contribute to its ongoing development (PX4 2023b).

Furthermore, Cube Orange integrates with high-precision GPS receivers, ensuring precise positioning and navigation—a vital feature for tasks like autonomous flights. Its adaptability extends to various payloads, including cameras, sensors, and communication systems. With

connectivity options encompassing UART, CAN bus, and I2C, it can interface effortlessly with various peripherals. Cube Orange was linked to an onboard computer through a serial connection in this work, facilitating real-time vehicle monitoring.

Figure 3.2: Cube Orange (from https://ardupilot.org/copter/docs/common-thecubeorange-overview.html(Ardupilot 2023c))

3.7 Jetson Nvidia

In this work, Jetson da Nvidia was chosen as a mission computer. Its processing capacity was also leveraged as a Flight controller simulator. This computer stays on board in the vehicle and operates using the data from the flight controller. The software running on this device processed this data to achieve specific objectives.

This project utilized the mission computer to execute Handover procedures and establish connections with two other ground-based computers through a multilink communication channel encompassing wifi and LTE. However, it holds the potential for diverse applications, such as logging flight data, automating commands during missions, implementing a detect and avoid system, and various other functionalities.

Figure 3.3: Jetson NVIDIA (from https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/(Nvidia 2023))

Chapter 4

Authority Handover Procedure

This chapter presents the system design to solve the problem enunciated in 1. First, section4.1 describes the application scenario of the UAV in a Beyond Visual Line of Sight operation. Then, section 4.2 provides an overview of the proposed design solution. Sections 4.3 and 4.4 detail the Mission Computer and Ground Control Stations software implementation. Section4.5 explains the sequence of the messages exchanged between the software components. Section 4.6 highlights the progress made in this research regarding the Handover Region selection. Lastly, Section 4.7 provides an overview of message safety and security and insights into emergency procedures.

4.1 Application Scenario

The system is compounded by two GCSs and the UAV. The vehicle must be able to deliver a packet crossing the authority handover region and turn back to the launch point. Figure 4.1 illustrates the proposed scenario to simulate a drone authority handover procedure.

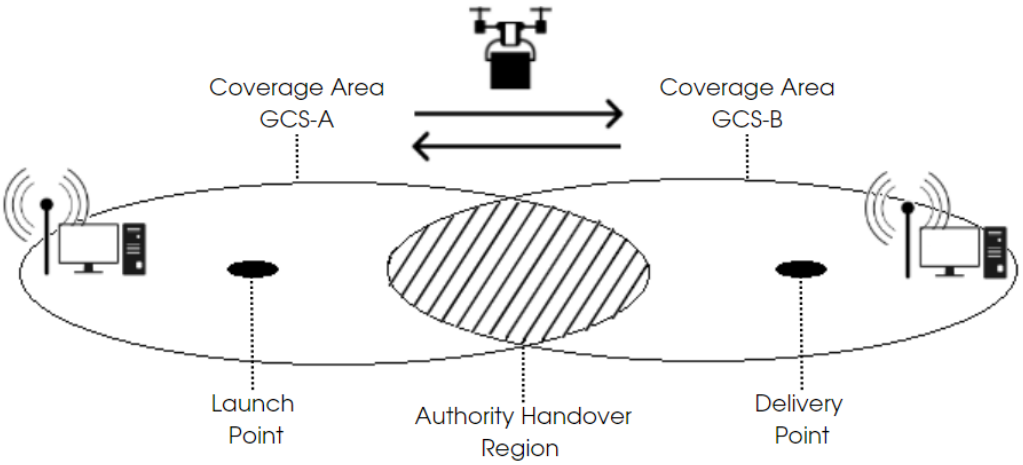


Figure 4.1: Authority Handover Procedure

At the start, the vehicle receives a mission plan with important parameters to conclude the task, such as destination, waypoints, flight speed, and rally points. When the vehicle is inside the coverage area, each GCS shall be able to receive a vehicle heartbeat and send commands during the flight, if necessary. A wireless access node onboard UAV will connect to already-known ground control stations.

Upon entering the handover region, the HO module must notify the GCS-A that it will transfer the authority over the mission to the GCS-B. If communication with the GCS-A is successful, the HO module will respond only to the GCS-B. The reverse happens on the trip back to the launch point.

4.2 System Architecture

The application scenario is based on package delivery. This work pretends implementing an application layer software using the C++ programming language and API Mavsdk. The implemented software (also referred to as "HO module" or "authority handover system" in this work) runs at Mission Computer and intermediate the communication between Ground Control Station and Flight Controller.

The Authority Handover System are able to collect information about the status of the vehicle, such as coordinates, estimated battery remaining, altitude, and speed. When the Mission Computer (MC) receives a mission or command from GCSs, the HO module can accept or reject the commands depending on the information gathered in the vehicle, and the authority region. The module also sends the status of the vehicle to the intended GCS. Figure 4.2 depicts the proposed solution.

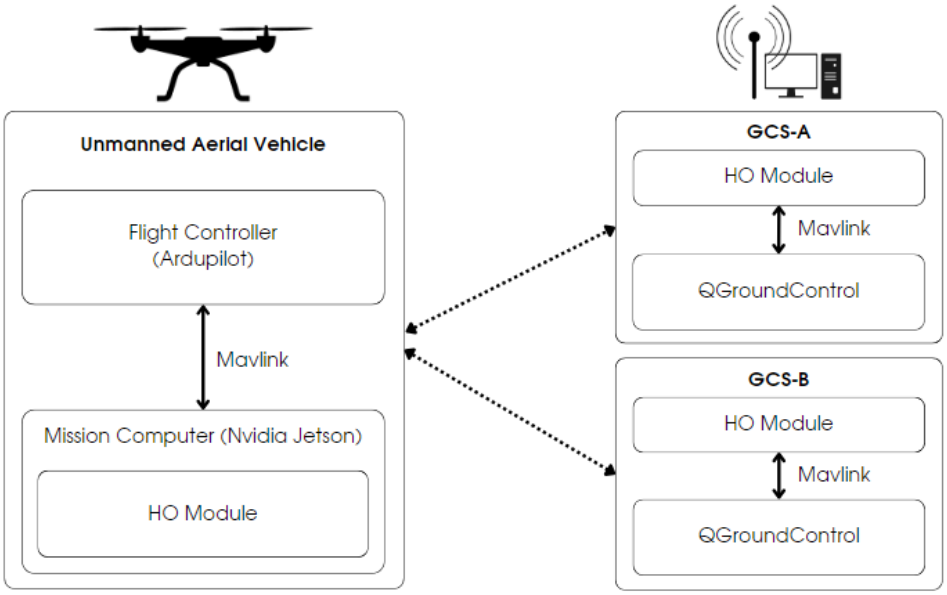


Figure 4.2: System Architecture

4.3 Mission Computer

The mission computer is responsible for controlling the drone behavior according to the data gathered by the flight controller. It has three central states: on_load, on_mission, on_handover. In each state, it has different behaviors or tasks. The main responsibility of each state is listed below.

4.3.1 On Load

The initial task of the mission computer involves establishing a connection with the flight controller. This connection is crucial for obtaining real-time data from the flight controller, including GPS coordinates. A separate task is dedicated to managing this connection and must run concurrently with another task.

Subsequently, the Mission Computer is required to establish a connection with the parent ground control station. Parameters such as the ID and IP address of the parent ground control station play a vital role in ensuring the security of communications. Finally, a third task is receiving and handling messages from ground control stations. Figure 4.3 shows the flowchart of these main tasks.

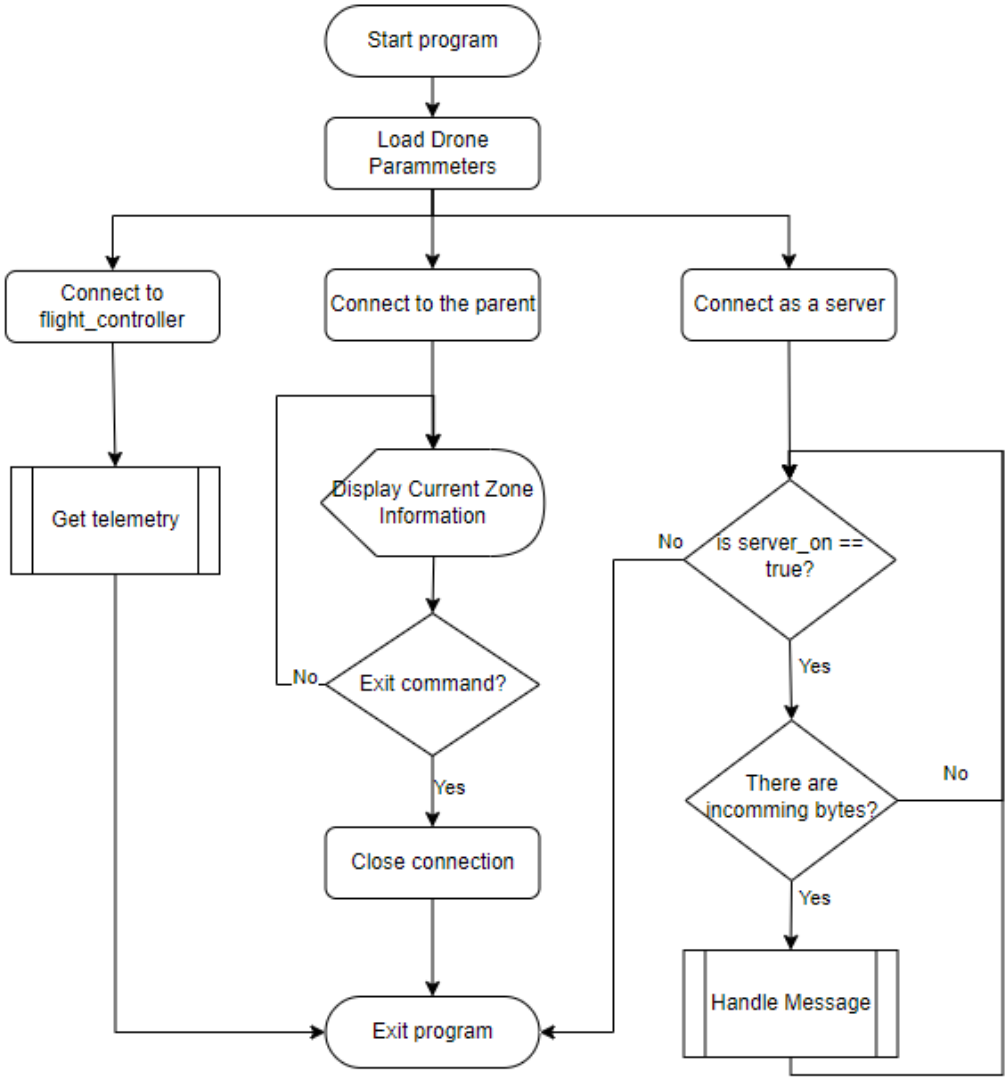


Figure 4.3: Mission Computer: Main Tasks

After that, the mission computer must wait and handle commands from the parent ground control station. In this implementation, the MC can handle only two initial commands. The first one is the command to receive a mission, and after having a mission loaded, the mission computer can handle the command to start the mission. In this work, the mission is a file

containing the information listed below. However, this work did not implement the task load mission using a file. It was written directly on code. Future work may consider implementing the task of reading mission parameters from a file.

The mission file includes:

- a list of waypoints;
- a list of rally points;
- GCS Parameters;
- GCS Geofences;
- Amount of authority handover procedures that the vehicle needs to do during the mission.

4.3.2 On Mission

When the vehicle receives the start command from the parent ground control station, the mission computer changes its state from on_load to on_mission. There are three main tasks that the drone needs to do when it is on_mission state. The first task is to send telemetry to the ground control station with authority over the drone (also called commander in this work). The frequency of this task is one message per second.

The second task is to start calculating the distance from each waypoint in the mission. This routine checks if the drone arrived at the delivery and land points. The third task runs each 50 milliseconds, and it verified in which region the vehicle is flying. These three tasks run parallel with both tasks that obtain data from the flight controller and listen for incoming commands from the current GCS (Figure 4.4).

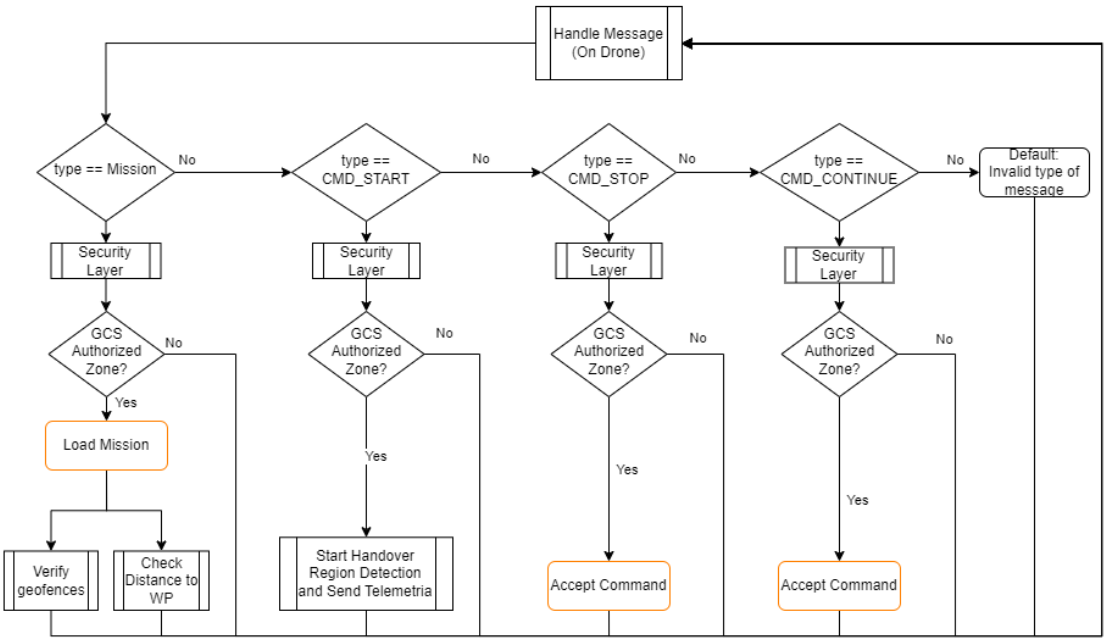


Figure 4.4: Mission Computer: Server Messages

4.3.3 On Handover

When the mission computer detects the vehicle flying over the handover region (as referenced in Chapter 4.6), it initiates the authority handover procedure. This procedure commences with the drone transmitting a signal to establish a connection with the ground control station to which authority will be transferred, referred to as "the partner." Following this, the vehicle awaits authorization from the current GCS, known as "the commander."

Subsequently, the vehicle sends a CONNECT_HANDOVER message to the partner and awaits the commander to send the DISCONNECT message. Finally, the vehicle disconnects from the current ground control station and switches to the new commander. Figure 4.5 illustrates this process.

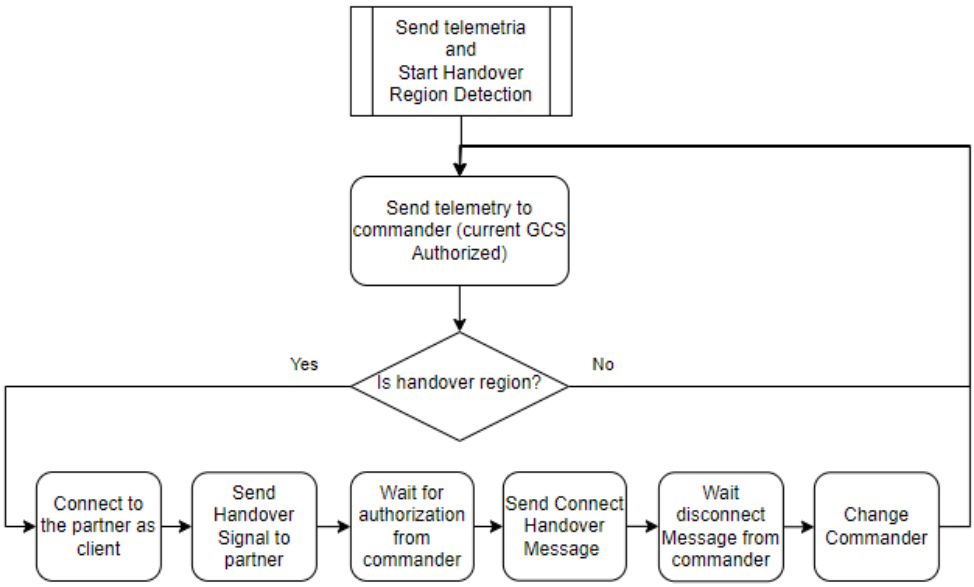


Figure 4.5: Mission Computer: Handover

4.4 Ground Control Stations

The ground control station sends and receives information to the mission computer, such as commands and telemetry. It also has an implementation of the Authority Handover module. The main task starts by reading the GCS parameter and connecting it as a server.

There are two tasks running parallel. The first task is responsible for reading commands from the terminal. The second task is receiving and handling messages from the drone or other ground control stations. Figure 4.6 provides a visual representation of these tasks.

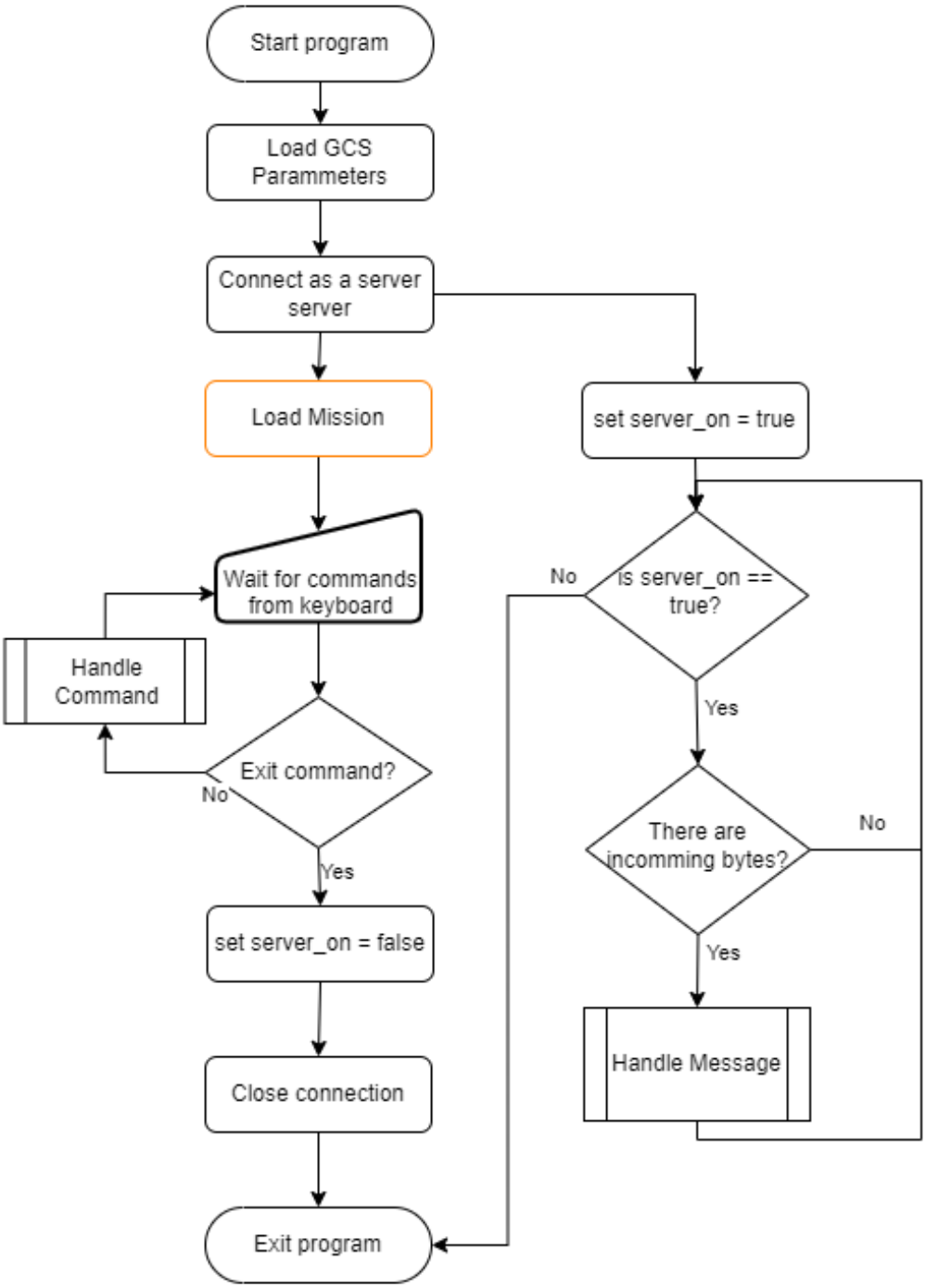


Figure 4.6: Ground Control Stations main function

Figure 4.7 illustrates the flowchart depicting the task responsible for managing commands received from the terminal. This task identifies and processes the commands accepted through the command line. When the variable "command" is set to 0, the loop terminates, concluding the main function of the GCS program. A value of "1" for the variable triggers the procedure for mission sharing, involving the transmission of the loaded mission to both

the drone and the partner. For variables equal to 2, 3, or 4, corresponding commands start, stop, and continue are transmitted to the mission computer.

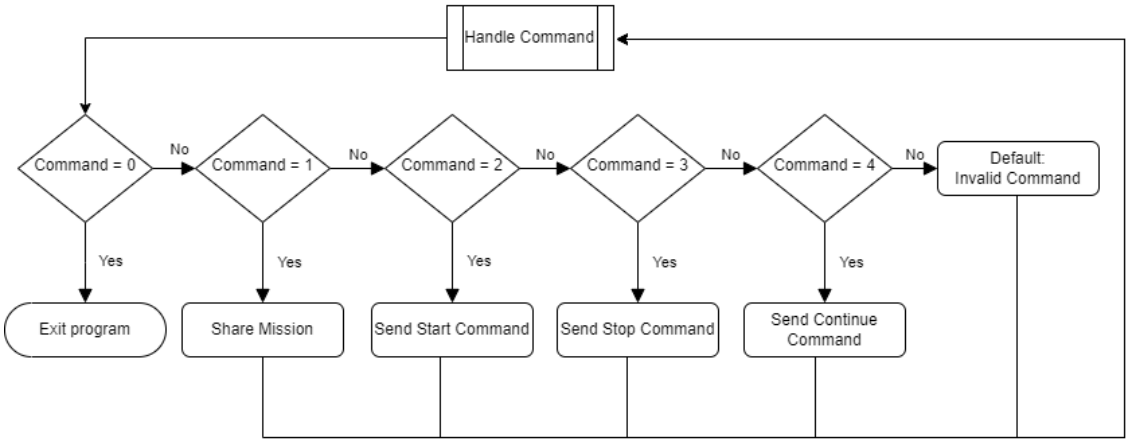


Figure 4.7: Ground Control Stations Commands

Figure 4.8 provides a visual representation of the message handling. The messages exchanged by the mission computers and the ground control stations have a code identifying the message type. If the device does not recognize the message type, it discards it. Consequently, the mission computer does not waste time processing an invalid message. Subsequently, all the messages pass through a security layer to ensure authenticity before executing their respective functions.

When the GCS receives a message of the Telemetry type, it processes the structure depicted in Listening 4.4. Not all variables in the data structure are utilized, as this depends on the data provided by the flight controller. The structure was designed to incorporate parameters considered necessary for the flight controller and relay this information to the ground control station, thereby monitoring the vehicle's status.

```
struct telemetry_ho{
    double time;
    float battery_v;
    double lat_deg;
    double long_deg;
    float altitude_m;
    float pitch_rad_s;
    float roll_rad_s;
    float yaw_rad_s;
    float vel_north_m_s;
    float vel_east_m_s;
    float vel_down_m_s;
    double heading_deg;
    float GroundSpeed;
    int wp;
    int zone;
};
```

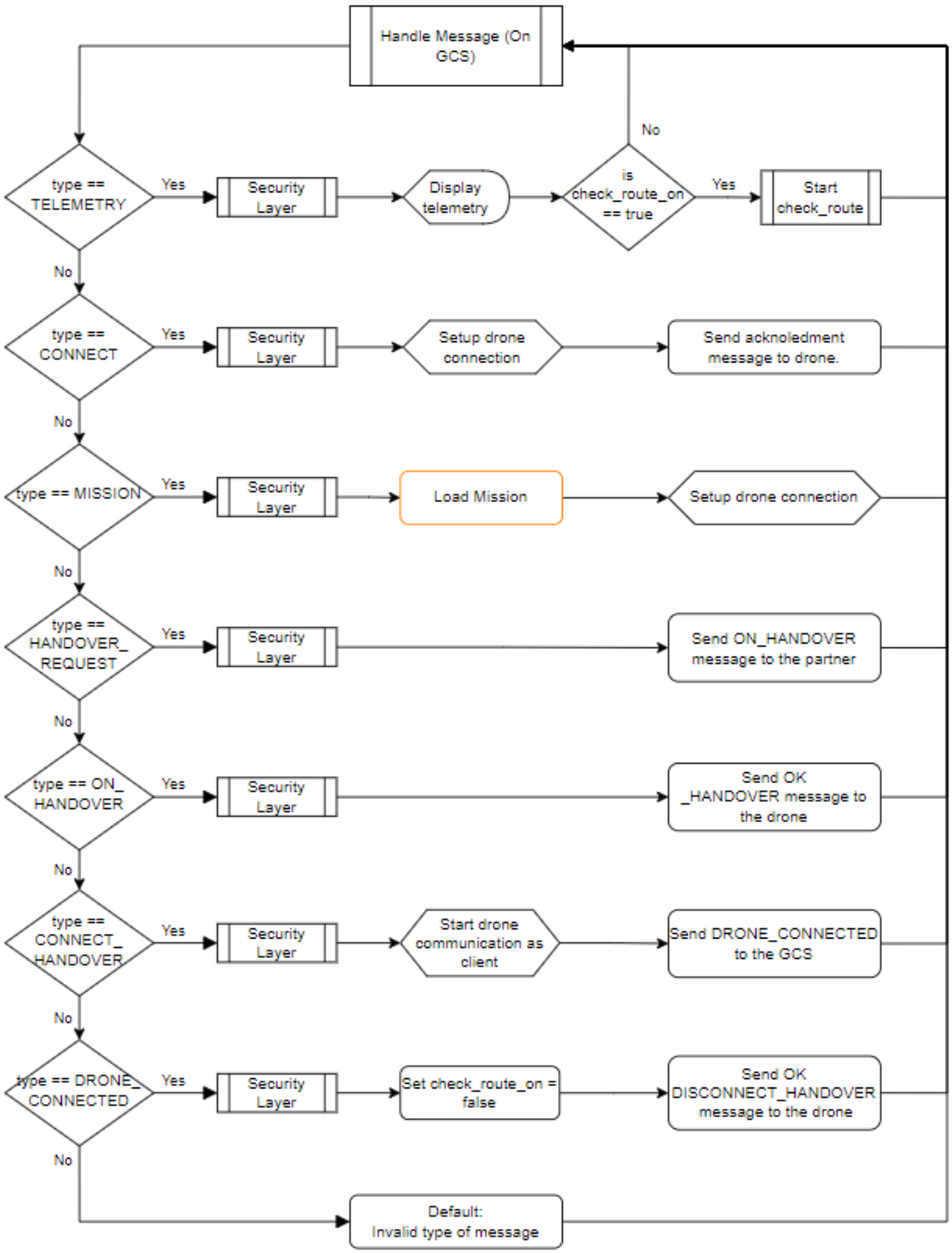


Figure 4.8: Ground Control Messages

After displaying the data, the subsequent procedure activates the routine "check_route," if it is inactive. This routine compares the heading received from the telemetry message with the expected heading for the mission. It enables the Ground Control Station (GCS) to monitor whether the drone adheres to the mission path and detect any unexpected trajectory changes before losing communication.

The "CONNECT" message type configures a new connection with the drone server by storing the drone's parameters and initializing communication. Upon completing this process, the GCS sends a confirmation message to the drone. When the GCS receives a "Mission"

message type, it loads the mission parameters received from a partner. Ideally, this should encompass all previously listed parameters. However, in this study, this procedure solely shares the information necessary to establish a connection with the vehicle. The mission parameters are static in the code. Future work could implement features to transmit all waypoints and mission parameters as previously described.

The remaining message types are related to the handover procedure. During a handover, ground control stations assume a passive role. They await the initial request from the drone and subsequently take appropriate action.

4.5 Messages Sequence

The handover procedure is a message sequence that starts when the drone enters the handover region. However, the message exchange between the devices starts when the drone powers up. After powering up and realizing internal configuration, the mission computer sends a "CONNECT" message to the ground control station registered as the parent. The parent, in turn, sends a confirmation to the mission computer, which remains on standby, waiting for commands from the parent.

To share the mission with both the drone and the partner, the operator of the ground control station needs to type a command via the command line. Once the mission is shared, the GCS partner waits for a signal from the drone while the mission computer waits for the parent ground control station to send the start command.

After the mission starts, the Mission Computer will send telemetry to the ground control station and start the routine to check the GCS geofences. When the mission detects that it has arrived in the GCSB coverage area, It will start the handover procedure by calling the handover function in Listing 4.1.

```
1  void Drone::handover(GCS *from, GCS *to)
2  {
3      unsigned char buffer[BUFFER];
4
5      std::cout << "[" << now() << "|" << this->get_id() << "][On handover
       ]: Start Handover from " << from->info.id << " to " << to->info.id <<
        std::endl;
6
7      to->connect(CLIENT_MODE);
8
9      std::cout << "[" << now() << "|" << this->get_id() << "][On handover
       ]: Sending HO message to " << to->info.id << "." << std::endl;
10
11     this->send_to(HANDOVER_REQUEST, this->get_id(), strlen((char *)this
       ->get_id()), to);
12
13     std::cout << "[" << now() << "|" << this->get_id() << "][On handover
       ]: Waiting for response from " << from->info.id << "." << std::endl;
14
15     this->wait_response(OKHANDOVER, 3, buffer, from);
16
17     std::cout << "[" << now() << "|" << this->get_id() << "][On handover
       ]: OK handover received from " << from->info.id << "." << std::endl;
18
19     this->send_to(CONNECT_HANDOVER, this->get_id(), strlen((char *)this
       ->get_id()), to);
```

```
20
21      std::cout << "[" << now() << "|" << this->get_id() << "][On handover
        ]: Waiting disconnect message from " << from->info.id << "." << std::
        endl;
22
23      this->wait_response(DISCONNECT_HANDOVER, 3, buffer, from);
24
25      std::cout << "[" << now() << "|" << this->get_id() << "][On handover
        ]: " << from->info.id << " Disconnected." << std::endl;
26
27      this->commander = to;
28      this->handover_done = true;
29  }
```

Listing 4.1: Handover Procedure in the Mission Computer

First, the drone sends a HANDOVER_REQUEST to the partner. When the partner receives this request, it will verify the drone's authenticity and then notify the ground control station that currently has authority over the drone. The current ground control station will receive this request identified by the ON_HANDOVER message, and then it will authorize the drone to go the handover by sending the message OK_HANDOVER.

After receiving the authorization, the drone will send the message CONNECT_HANDOVER to the partner. It will set up the communication parameters with the drone server and confirm that the drone is already connected to the ground control station.

Finally, after the ground control station receives the DRONE_CONNECTED message, it will send the drone a DISCONNECT_HANDOVER that will conclude the handover procedure by changing the ground control station that has authority over the drone and disconnecting from the ground control station.

The payload of these messages is information necessary to guarantee the authenticity of the devices. The diagram presented in Figure 4.9 illustrates the sequence of these messages. The same procedure repeats on the trip back to the launch point. However, the authority handover is done from GCSB to GCSA.

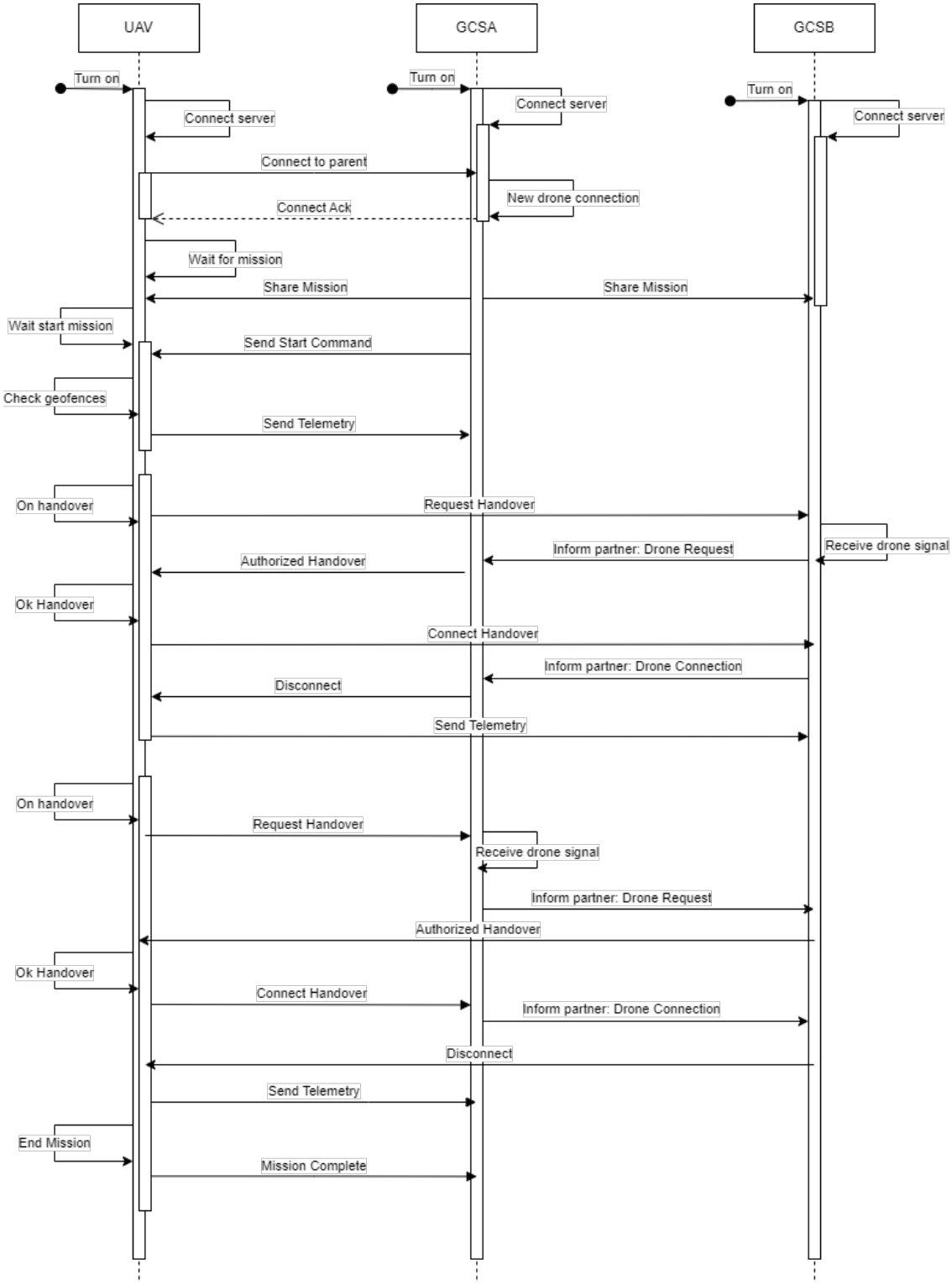


Figure 4.9: Message Sequence diagram

4.6 Handover Region

The authority handover process involves a series of message exchanges between an Unmanned Aerial Vehicle (UAV) and two distinct Ground Control Stations (GCSs). The initiation of this message sequence depends on the UAV reaching a predefined area referred to as the handover region. The handover region is where two or more GCS signal ranges intersect. Consequently, when the mission computer detects that the drone has entered this region, it triggers the authority handover process.

During the development of this research, three distinct approaches were proposed to delineate this handover region. These methodologies will be explained in the subsequent sections of this work.

4.6.1 Circular Geofence

The initial approach to defining the handover region involves the intersection of two circles, with their centers located at coordinates representing a virtual position of the Ground Control Station and their radius determined by the range, as shown in Figure 4.10.

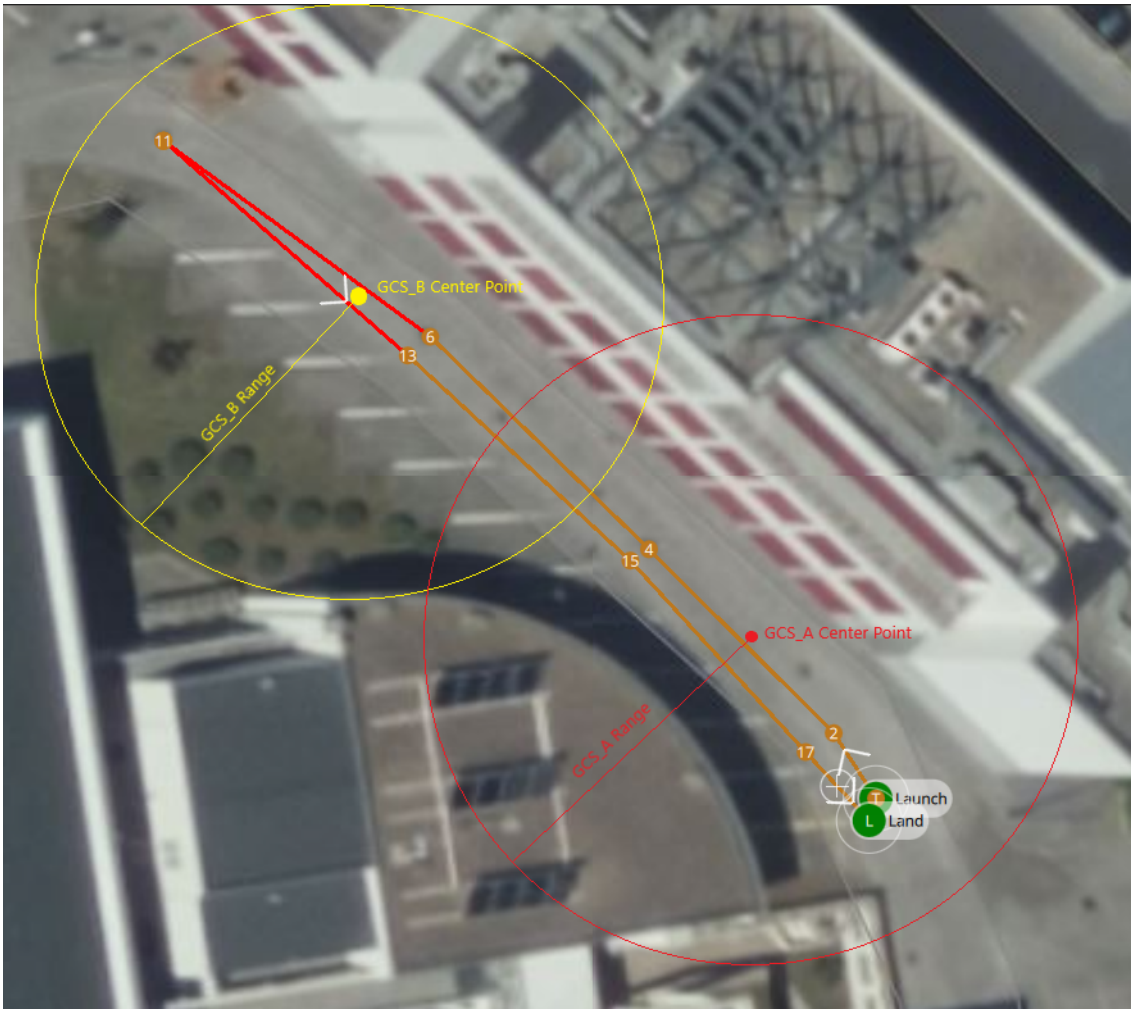


Figure 4.10: Circular Geofence

During the flight, the Mission Computer (MC) calculates the distance between the drone's current position and the coordinates of the circle's center. If this distance is less than the radius, the drone is inside the GCSs coverage area. In this implementation, handover messages are triggered when condition 4.1 is satisfied for both GCSs.

$$Distance - GCS_{radius} < 0 \tag{4.1}$$

The distance was computed using the Haversine formula, which calculates the distance between two coordinates on a sphere. This formula can be adapted for geographic coordinates, latitude, and longitude on the Earth's surface. However, the Earth's surface is not a perfect sphere, and there are errors associated with the result of this operation (Andreou et al. 2023).

```
double distanceCoordinates(double* lat1, double* lon1, double lat2,
    double lon2) {

    //Earth's radius in kilometers
    const double raioTerra = 6371;

    //Conversion to Radians
    double rad_lat1 = *lat1 * M_PI / 180;
    double rad_lon1 = *lon1 * M_PI / 180;
    double rad_lat2 = lat2 * M_PI / 180;
    double rad_lon2 = lon2 * M_PI / 180;

    //Calculate differences between the two coordinates in radians.
    double dLat = rad_lat2 - rad_lat1;
    double dLon = rad_lon2 - rad_lon1;

    //Haversine Formula
    double a = pow(sin(dLat/2),2) + cos(rad_lat1) * cos(rad_lat2) * pow(
    sin(dLon/2),2);
    double c = 2 * atan2(sqrt(a), sqrt(1-a));
    double distance = raioTerra * c;

    //Converting to Meters
    distance = distance * pow(10,3);

    //Return distance
    return distance;
}
```

Listing 4.2: Haversine Function.

The code above is an implementation of the Haversine function. In this function, the parameters "lat1" and "lon1" represent the coordinates of the drone, while "lat2" and "lon2" correspond to the virtual coordinates of the GCS's center.

While this approach works well for missions that follow a linear trajectory, real-world drone scenarios often involve more complex flight paths. For example, in the mission presented in the 4.10, the handover region of the delivery and the return paths are the same. In a scenario where this does not happen, delimiting a circle that covers both routes can become complicated, as it is possible to see in Figure 4.11. Consequently, a second strategy was implemented to define the handover region more effectively.

Figure 4.11: Circular Geofence Limitation

4.6.2 Mission Waypoint Detection

The second solution proposed was to associate the handover trigger with a specific waypoint in the mission. The Mission Computer continuously calculates the distance to the next waypoint in this implementation. It also employs the Haversine formula to determine the distance between two coordinates on the Earth's surface. When the distance becomes less than a certain radius defined around the waypoint, the MC recognizes that the drone has reached the waypoint. During the mission creation, it is necessary to specify which waypoint corresponds to each GCS authority region manually. Figure 5.4 illustrates this process. The trigger for the handover messages happens when the MC detects that the achieved waypoint belongs to another GCS.

The mission waypoint detection approach does not depend on delimiting a region around the control station. It was one solution when the mission path was complex. Nevertheless, this approach also brings to the project some limitations. For example, if the mission is not automatic, i.e., the mission is in guided mode, and a pilot needs to control the vehicle, the pilot must pass through the waypoint in the approximated coordinates to the software identifies that the vehicle achieved this region. In scenarios where the UAV needs to fly around obstacles, the MC may not recognize that the vehicle has reached a specific waypoint. Implementing a code to handle these situations can increase the software's complexity.

Additionally, manually determining which waypoints correspond to each GCS can be hard for mission planners. For these reason, a third solution was proposed to define the handover trigger more effectively.

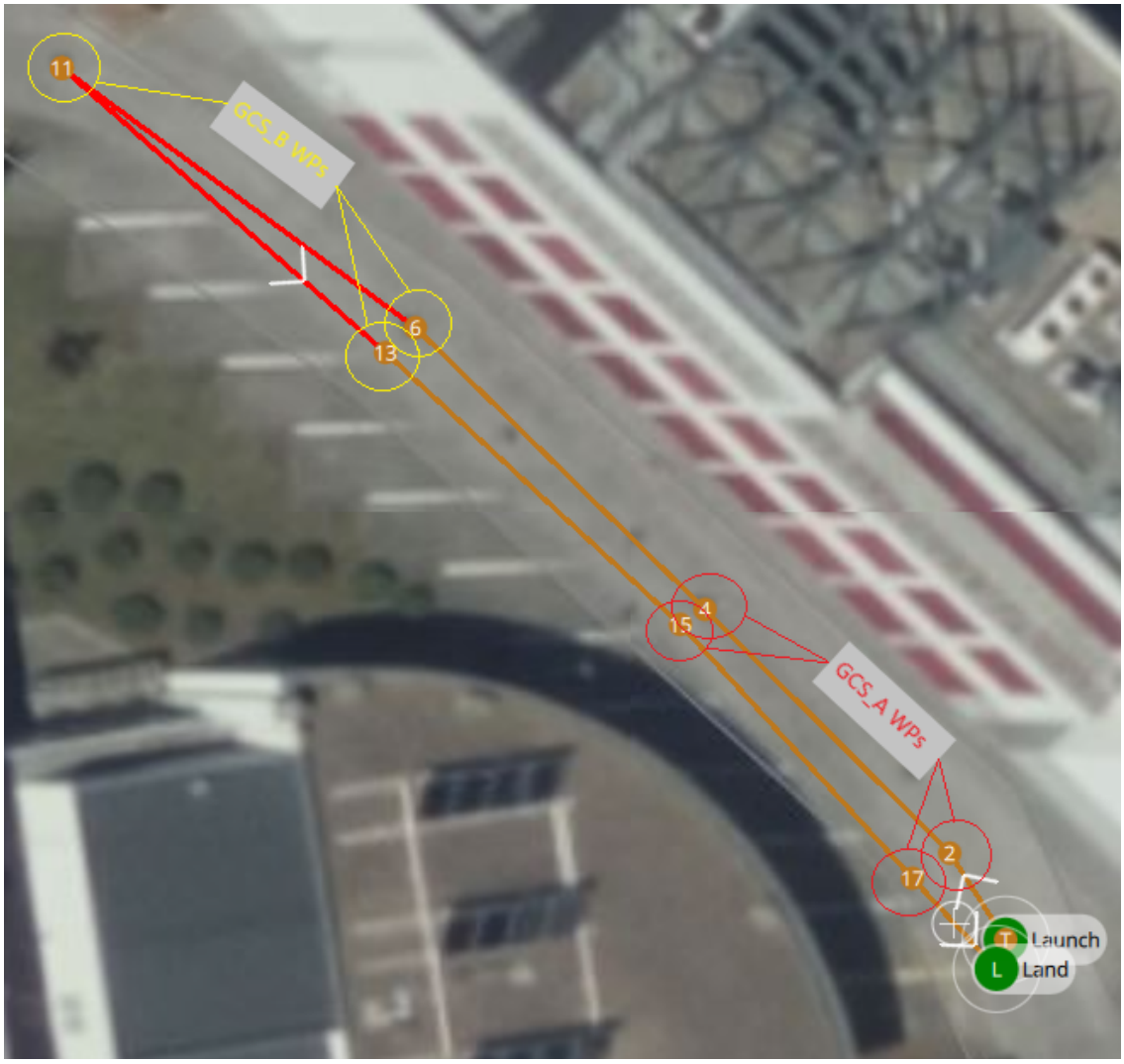


Figure 4.12: Mission Waypoint Detection

4.6.3 Polygonal Geofence

The third solution was to define a polygon for identifying the range of each ground station. It involves defining a polygonal boundary by connecting a series of geographic coordinates to form a closed shape, as shown in Figure 4.13. While determining whether a point belongs to a region may seem straightforward, implementing this task involves computational complexity. Several algorithms have been proposed, each offering different trade-offs between speed and accuracy. The algorithm selection depends on factors such as the geofence's complexity, the desired accuracy level, and the available hardware resources (Stevens, Rastgoftar, and Atkins 2017).

This work implemented the Ray Casting method to determine whether a point lies within the polygon's boundaries. This method involves tracing an imaginary line from the point to be analyzed until it crosses a polygon's edge and then counting these ray intersections. If the number of crossings is odd, the point is inside the polygon. Otherwise, if the number of crossings is even, it is outside.(Fu et al. 2019).

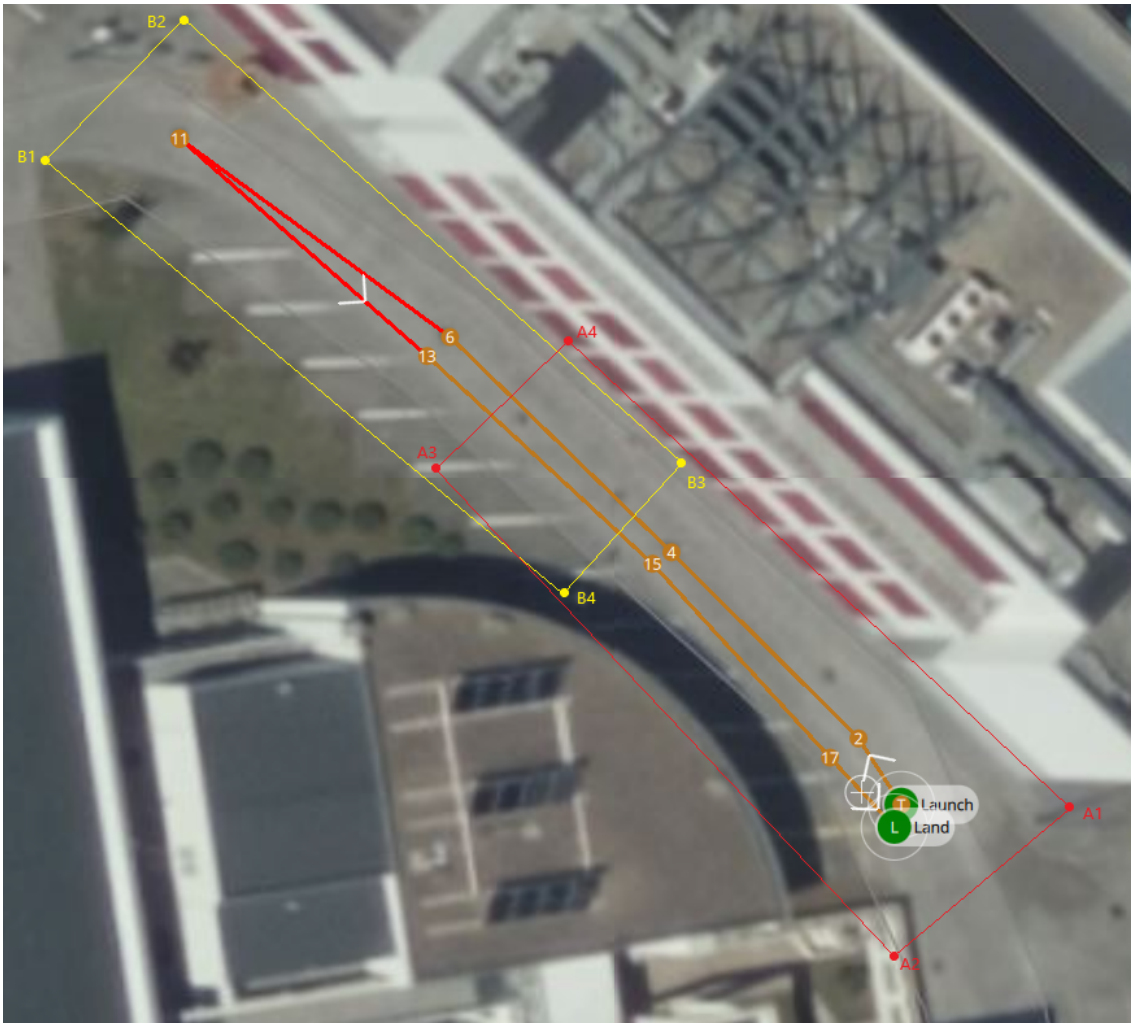


Figure 4.13: Polygonal Geofence

```
bool isInsidePolygon(Point point, std::vector<Point>& polygon) {
    int i, j;
    int num_intersections = 0;
    int num_vertices = polygon.size();
    for (i = 0, j = num_vertices-1; i < num_vertices; j = i++) {
        if (((polygon[i].y>point.y) != (polygon[j].y>point.y)) &&
            (point.x < (polygon[j].x-polygon[i].x) * (point.y-polygon[i
    ].y) / (polygon[j].y-polygon[i].y) + polygon[i].x))
            num_intersections += 1;
    }
    return num_intersections % 2 == 1;
}
```

Listing 4.3: Point in Polygn Function.

The function iterates over the polygon edges and checks the ray intersection from the actual vehicle location. The point variable is the current location of the drone. Each GCS has its polygon structure. The polygon structure was abstracted as a vector of vertices, where each vertex is defined by its latitude and longitude. In this implementation, polygon[i] and polygon[j] represent consecutive vertices of the polygon.

The condition inside the if statement determines if the ray from the point crosses the edge in terms of latitude and longitude. By repeating this process for all vertices of the polygon, it is possible to determine if the current drone location is inside or outside the polygon by verifying if the number of intersections is odd or even.

During the flight, the Mission Computer calculates if the vehicle is inside of GCSs' geofence and the trigger for handover messages occurs when this condition is true for both GCSs.

4.7 Safety and Security

This work implements the Wireless Safety and Security Layer (WSSL) in the communication between the Ground Control Station and the Mission Computer to ensure safety and security in the authority handover scenarios.

The WSSL consists of an external layer implemented in a general wireless communication environment. The WSSL was designed to be used in critical communication. It enables the detection of communication errors and checks the message's authenticity using a message signature that identifies the device (Cunha Rocha et al. 2023).

WSSL can be used to monitor the message delay, and in case of network failure, It returns an error message that can trigger the emergency procedure. WSSL also increases the system's security by detecting and discarding messages from unauthorized devices.

The software modules developed for this research did not include emergency procedures in case of communications failures. However, some features from the autopilot can be helpful, for example, pre-determine rally points and the command Return To Launch (RTL).

The rally points provide alternative landing points when the command return to launch is sent to the flight controller. The RTL is a safety feature that many autopilots use to ensure that the drone returns safely to a known location in case of unexpected events, such as communication loss. The RTL command is essential when the drone operates beyond the visual line of sight.

Once RTL is initiated, the drone will autonomously navigate back to the GPS coordinates where it took off. It typically ascends to a predefined safe altitude to clear potential obstacles and then follows a straight path back home.

When WSSL detects unexpected behavior, the mission computer can send the RTL command to the flight controller. If the mission disposes of a list of rally points, the mission computer can calculate the distance of the nearest rally point considering the amount of battery remaining.

Furthermore, the routine check_route can help safety since it identifies when the drone has a trajectory deviation and can trigger an emergency routine. Another option already implemented in the software is the check_geofence routine. If the drone is flying outside a coverage area, it also can activate the emergency procedure.

Chapter 5

Software and Hardware Architecture

This chapter describes the implementation of the authority handover in three different scenarios. First, section 5.1 presents the simulation environment and the software implementation. Section 5.2 describes the integration between the Mission Computer and Flight Controller. Finally, section 5.3 explains the integration with Adacorsa partners.

5.1 Testbed Architecture

The Authority Handover Module was designed to run in the application layer. Consequently, it will work independently of the hardware used in the communication between the devices on the ground and the air. In this work, the communication between the Ground Control Station was implemented using UDP/IP, and it was assumed that all the devices had a pre-defined static IP address.

Only in development and simulation scenarios is it possible to replace the wireless communication with a wired one. During the simulation tests, two computers were used as GCS, the Jetson Nvidia, and a switch was used to connect all the devices (Figure 5.1).

Figure 5.1: Simulation Setup

In this setup, the two computers are running in the Authority Handover Module of the GCS side, and each one loads a parameters list with specific information such as ID and IP. On Jetson Nvidia, the Authority Handover Module of the vehicle side and the software simulator were running (Figure 5.2).

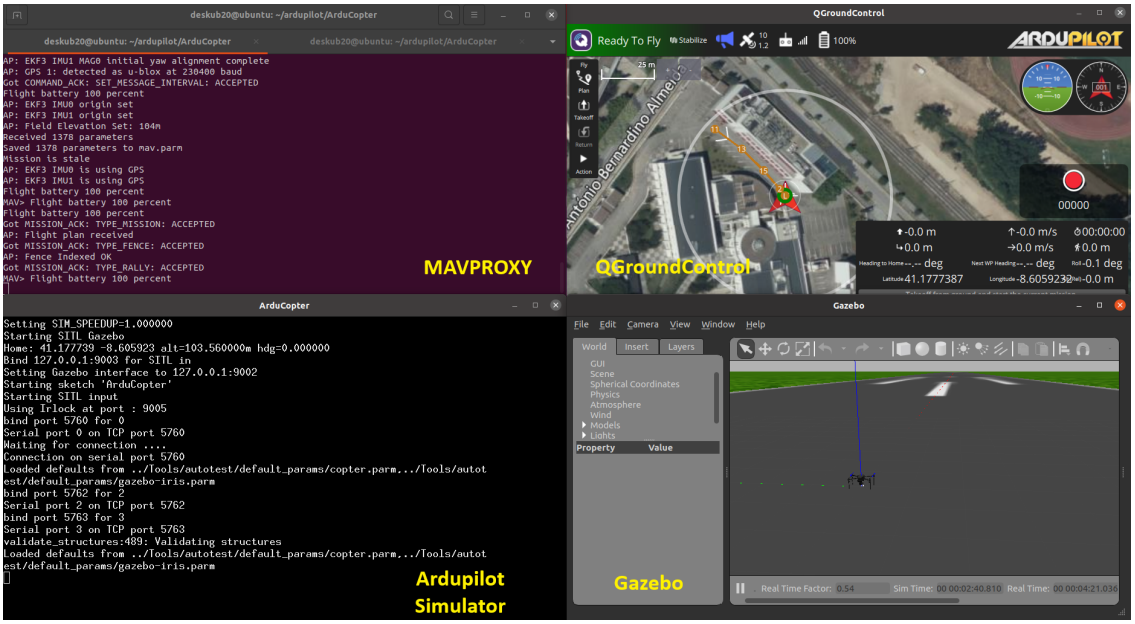


Figure 5.2: Simulation Softwares

Mavproxy was used to route the simulator connection over UDP to the ground control stations and Gazebo. QGroundControl and Gazebo connected with the autopilot allowed the observation of the vehicle movement while the tests ran. The commands below were used to run all simulators.

```
//Ardupilot+Mavproxy
$ cd ~/ardupilot/ArduCopter/$
$ ../Tools/autotest/sim_vehicle.py -L Cister -f gazebo-iris --out
    =127.0.0.1:14530 --out=192.168.10.100:14550 --out
    =192.168.10.128:14550

//Gazebo
$ gazebo --verbose ~/ardupilot_gazebo/worlds/iris_arducopter_runway.
    world
```

Listing 5.1: Terminal Commands to start simulation

The Authority Handover Procedure needs to have access to the GPS information of the vehicle to calculate in which zone it is. This work used the MAVSDK C++ library to access the vehicle data and telemetry. Listing 5.2 shows a simple implementation using subscription methods to continuously receive and process flight data from the drone in the application implemented using C++ programming language.

```
1  int MC::connect_mc(){
2
3      //Connect with simulator
4      this->mavsdk.add_any_connection("udp://:14530");
5
6      std::cout << "Waiting to discover system...\n";
7      this->system = get_system(this->mavsdk);
8      if (!system) {
9          return -1;
10     }
11
12     //Subscribe to the telemetry
13     this->telemetry = new mavsdk::Telemetry{system};
14
15     this->telemetry->subscribe_battery([&](mavsdk::Telemetry::Battery
       battery) {
16         this->vehicle_telemetry.battery_v = battery.voltage_v;
17     });
18
19     this->telemetry->subscribe_position([&](mavsdk::Telemetry::Position
       position) {
20         this->vehicle_telemetry.lat_deg = position.latitude_deg;
21         this->vehicle_telemetry.long_deg = position.longitude_deg;
22         this->vehicle_telemetry.altitude_m= position.relative_altitude_m
       ;
23         this->vehicle_telemetry.time = time(0);
24     });
25
26     this->telemetry->subscribe_velocity_ned([&](mavsdk::Telemetry::
       VelocityNed velocity_ned) {
27         this->vehicle_telemetry.vel_north_m_s = velocity_ned.north_m_s;
28         this->vehicle_telemetry.vel_east_m_s = velocity_ned.east_m_s;
29         this->vehicle_telemetry.vel_down_m_s = velocity_ned.down_m_s;
30     });
31
32     this->telemetry->subscribe_attitude_angular_velocity_body([&](mavsdk
       ::Telemetry::AngularVelocityBody velocity_body) {
33         this->vehicle_telemetry.pitch_rad_s = velocity_body.pitch_rad_s;
34         this->vehicle_telemetry.roll_rad_s = velocity_body.roll_rad_s;
35         this->vehicle_telemetry.yaw_rad_s = velocity_body.yaw_rad_s;
36
37     });
38
39     this->telemetry->subscribe_heading([&](mavsdk::Telemetry::Heading
       heading) {
40         this->vehicle_telemetry.heading_deg = heading.heading_deg;
41     });
42
43     return 0;
44 }
```

Listing 5.2: Flight Data Acquisition using MAVSDK

The code shows the subscription to the battery voltage, position, heading, velocity, and angular velocity data. First, it was necessary to initialize the MAVSDK library and set up the connection to the drone. For the simulation, a network connection was used (line 4).

After initializing the library, an object representing the drone was created and used to set up

the subscription methods. After subscribing to a particular data type, the callback functions are triggered whenever new data is available. This data provides real-time information about the drone's state. The real-time data enabled the programming and control of the mission computer behavior to accomplish the Authority handover specification.

This simulation does not implement commands sent from the mission computer to the flight controller over Mavlink.

Utilizing these open-source simulators, libraries, and tools allowed for establishing a simulated environment for implementing and testing the Authority Handover Module and communication systems between the vehicle and diverse Ground Control Stations (GCSs).

The next step involved a gradual transition to real-world implementations, which included integrating with the flight controller hardware and upgrading communication hardware.

5.2 Flight Controller Integration

The subsequent phase of this project was to implement and test the Authority handover module in real-world scenarios. A drone GAIA 160MP with a Cube Orange Flight Controller was utilized To accomplish this objective (Figure (5.3)). The Cube Orange has a serial port that provides full flight control. It was connected to the Jetson Nvidia to obtain the vehicle flight data

Figure 5.3: GAIA 160MP with Cube Orange Flight Controller

The Multilink Gateway was used to establish the communication between the devices in the air and ground. Adacorsa partners provided it, and it is still in development. The Multilink gateway was designed to provide reliable communication in choosing the most appropriate link technologies, such as LTE and WiFi. One Gateway and the Concentrator compound it. The Gateway was on board the vehicle and connected with Jetson Nvidia through Ethernet. The Concentrator is on the ground, and it is connected with a router and also with the GCS computers. Figure 5.4 identifies each connection of this hardware setup.

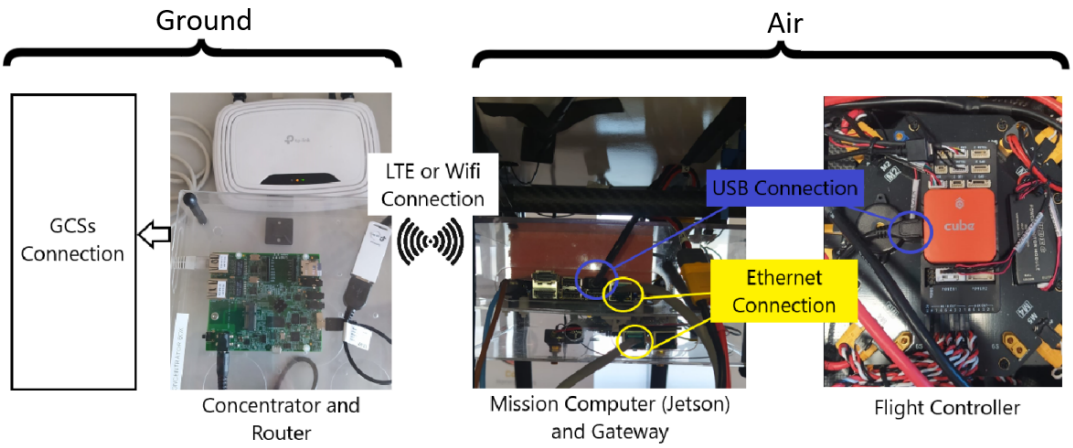


Figure 5.4: Hardware Setup

It is necessary to define a static IP and add a new entry in the IP route of the devices involved to make this system work properly. The listing 5.3 below implements a bash script that was used to set up this configuration.

```
#!/bin/bash

#MC
#IP="192.168.11.100"
#NETMASK="255.255.255.0"
#ROUTE="192.168.10.0/24"
#GATEWAY="192.168.11.1"
#IF_NAME_ETH="eth0"

#GCS
IP="192.168.10.100"
NETMASK="255.255.255.0"
ROUTE="192.168.11.0/24"
GATEWAY="192.168.10.1"
IF_NAME_ETH="eno1"

echo "Setting up IP address..."
sudo ifconfig "$IF_NAME_ETH" "$IP" netmask "$NETMASK"

echo "Adding new entry on ip route"
sudo ip route add "$ROUTE" via "$GATEWAY" dev "$IF_NAME_ETH"
```

Listing 5.3: Multlink Configuration Script

On Jetson Nvidia, this script was implemented as a Linux Systemd Service, so it was possible it ran automatically as soon as the system powered up. The implementation of the Service is shown on the listing 5.4. After creating the service file on location "/etc/systemd/system/," it was necessary to enable the Service by running the command of the Listing 5.5.

```
[Unit]
Description= Multilink Setup

[Install]
WantedBy=multi-user.target

[Service]
Type=simple
ExecStart=/home/adacorsa/multlink_setup.sh
Restart=on-failure
RestartSec=10
KillMode=process
```

Listing 5.4: Service File on Linux

```
sudo systemctl enable multilink_setup.service
```

Listing 5.5: Enable Service

Regarding software, most of the implementation in the integration with the Flight Controller was the same as described in Section 5.1. As the Multilink guarantees the maintenance of the same IP on the devices, the only adjustment was in the MAVSDK library. To acquire flight data, it was necessary to specify the serial connection with the Flight Controller on the Mission Computer. It was possible specify the serial port and the baud rate as shown below and replace it on line 4 in Listing 5.2.

```
this->mavsdk.add_any_connection("serial:///dev/ttyACM0:57600");
```

Listing 5.6: Serial Communication Setup between Mission Computer and Flight Controller

5.3 Integration with Adacorsa Partners

As mentioned before, the Adacorsa project has many partners, each partner developed its module separately, and until the end of the project, all projects will be integrated.

This approach was practical to increase the modularity and decrease the dependency between partners during the development. Figure 5.5 show the hardware integration of all box developed by the partners. In the image 5.5, it is possible to identify the mission computer box that contains the Jetson Nvidia inside and the switch that connects all boxes via Ethernet.

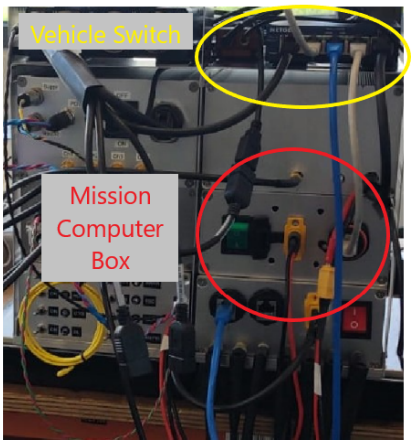


Figure 5.5: Hardware Integration with Adacorsa Partner

According to the Adacorsa specification, the flight controller data would be provided by another box onboard the vehicle over TCP/IP socket using the message format presented in the code below.

```
struct droneData
{
    double Time;          // UTC Time [sec.nanosecs]
    double Latitude;      // Latitude [rad]
    double Longitude;     // Longitude [rad]
    float BaroHeight;     // ISA-barometric height [m]
    float AglHeight;      // AGL height (barometric) [m]
    float TrueTrack;      // True track [rad]
    float TrueHeading;    // True heading [rad]
    float GroundSpeed;    // Ground speed [m/s]
    float EastVelocity;   // West->east velocity [m/s]
    float NorthVelocity;  // South->north velocity [m/s]
    float UpVelocity;     // Down->up velocity [m/s]
};
```

Listing 5.7: Data Struct from TCP Provider

To accomplish this integration, a TCP client was necessary to request the flight data to a server. The listing 5.8 shows a client implementation requesting the server's flight controller information. In the software implementation, it was necessary to call the function at Listing 5.8 instead of the function presented in Listing 5.2.

```
void MC::run_connect_tcp()
{
    int status, valread, client_fd;
    struct sockaddr_in serv_addr;

    socklen_t socklen = sizeof(struct sockaddr_in);

    char buffer[BUF] = { 0 };

    if ((client_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return;
    }

    serv_addr.sin_family = AF_INET;
```

```
16      serv_addr.sin_port = htons(PORT_NRL);
17
18      if (inet_pton(AF_INET, IP_NRL, &serv_addr.sin_addr)<= 0) {
19          printf("\nInvalid address/ Address not supported \n");
20          return;
21      }
22      if ((status = connect(client_fd, (struct sockaddr*)&serv_addr,sizeof
        (serv_addr)))< 0) {
23          printf("\nConnection Failed \n");
24          return;
25      }
26
27      std::cout << "TCP Server initialized." << std::endl;
28
29      while (!status && this->connect_tcp_on.load())
30      {
31          valread = read(client_fd, buffer,BUF);
32          droneData *data = (droneData *)(buffer);
33
34          print_drone_data(data);
35          this->vehicle_telemetry.time  = data->Time;
36          this->vehicle_telemetry.lat_deg = data->Latitude;
37          this->vehicle_telemetry.long_deg = data->Longitude;
38          this->vehicle_telemetry.BaroHeight = data->BaroHeight;
39          this->vehicle_telemetry.altitude_m = data->AglHeight;
40          this->vehicle_telemetry.TrueTrack = data->TrueTrack;
41          this->vehicle_telemetry.TrueHeading = data->TrueHeading;
42          this->vehicle_telemetry.GroundSpeed = data->GroundSpeed;
43          this->vehicle_telemetry.vel_down_m_s = data->UpVelocity;
44          this->vehicle_telemetry.vel_east_m_s= data->EastVelocity;
45          this->vehicle_telemetry.vel_north_m_s= data->NorthVelocity;
46      }
47      close(client_fd);
48  }
```

Listing 5.8: Flight Data Acquisition using TCP/IP

Chapter 6

Evaluation and Results

This chapter describes the results of the authority handover in three different scenarios described in Chapter 5. Section 6.1 presents the results of the simulation environment. Section 6.2 shows the integration results between the Mission Computer and Flight Controller. Finally, Section 6.3 includes the result of the integration with Adacorsa partners.

6.1 Simulation Results

The mission plan presented in the simulation scenario was based on the Adacorsa demonstrator 8.4 flight path. Figure 6.1 identifies the GCSA with a red polygon and the vertices Ax. The yellow rectangle represents the GCSB coverage area. The drone path is in orange. The takeoff and the land location are the same. The waypoint 8 represents the delivery point. Also, it is possible to identify the drone across the handover region four times.

Figure 6.1: Mission Plan

Figure 6.2 shows the procedure of starting a mission. Both Ground control Stations are already running when the drone simulations are powered up. The first drone action was

connecting to the flight controller and activating its server. Then, it connected to the parent, in this case, the GCSA.

After that, the drone waited to receive a mission from the parent. Both ground control stations also started connecting their server. After, they will wait for a connection from the drone or receive the mission from another GCS.

The drone was inside the GCSA area, meaning it only accepted the GCSA commands. The green arrows show both GCS trying to send the command start to the drone. However, only the GCSA command is accepted. After that, the mission computer started to send telemetry to the GCSA.

Figure 6.2: Start Mission Command

6.1.1 Authority Handover

When the vehicle entered the GCSA and GCSB geofence intersection, the mission computer recognized the handover region and initiated the authority handover procedure.

Figure 6.3 illustrates the authority handover messages exchanged between the devices involved. The drone initiated by transmitting a signal to GCSB (Step 1) and awaited a response from GCSA. GCSB received the drone's signal and relayed it to GCSA (Step 2). Upon receiving the request from GCSB, GCSA responded to the drone (Step 3).

Following the confirmation from GCSA for the handover, the drone transmitted a "connect handover" message to GCSB (Step 4). GCSB accepted the connection and notified the partner (Step 5), while GCSA sent a "disconnect" message to the drone (Step 6). After the

handover, the drone disconnected from GCSA (Step 7) and commenced sending telemetry data to GCSB.

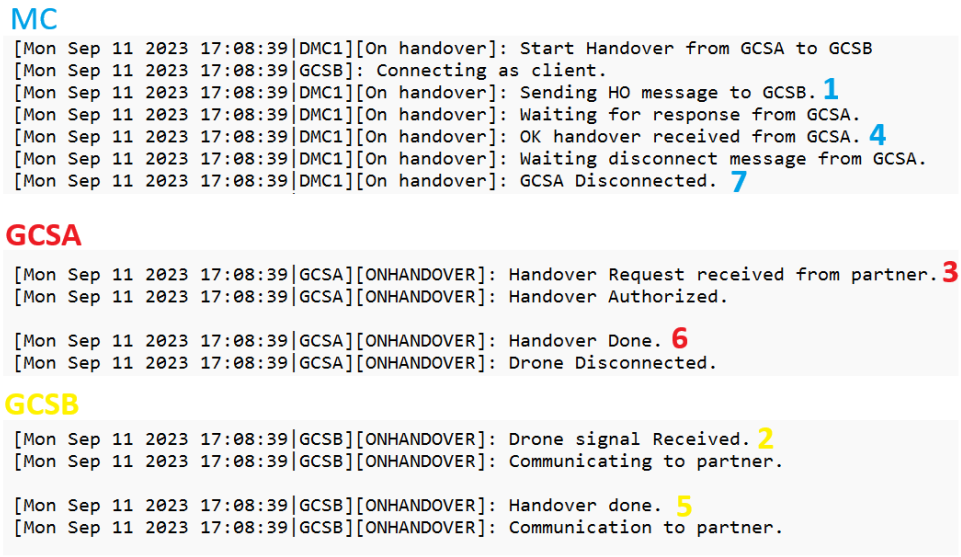


Figure 6.3: Authority Handover Messages

Figure 6.4 shows the vehicle in the QGroundControl crossing the handover region. The green arrows show the sequence of handover messages at that moment. In sequence, it is possible to visualize that GCSA stopped to receive telemetry from the vehicle, and GCSB started to receive the telemetry messages. In the handover from GCSB to GCSA, the same sequence happens, but instead, the drone starts sending the signal to GCSA (Figure 6.5).

Figure 6.4: Authority Handover Procedure

Figure 6.5: Authority Handover Procedure

6.1.2 Authority Check

As the drone flies through the coverage area of GCSB, it is required to transmit telemetry data to GCSB and acknowledge their commands. This scenario is depicted in Figure 6.7, where the drone is situated within the coverage area of GCSB. When it received the GCSB command, it accepted it. However, when the command was sent from GCSA, the mission computer rejected the command.

In Figure 6.6, the green arrows similarly illustrate that the GCSs sent the commands to the vehicle while within GCSA's coverage area. Notably, the mission computer rejected the command from GCSB but accepted the command from GCSA, aligning with the expected behavior.

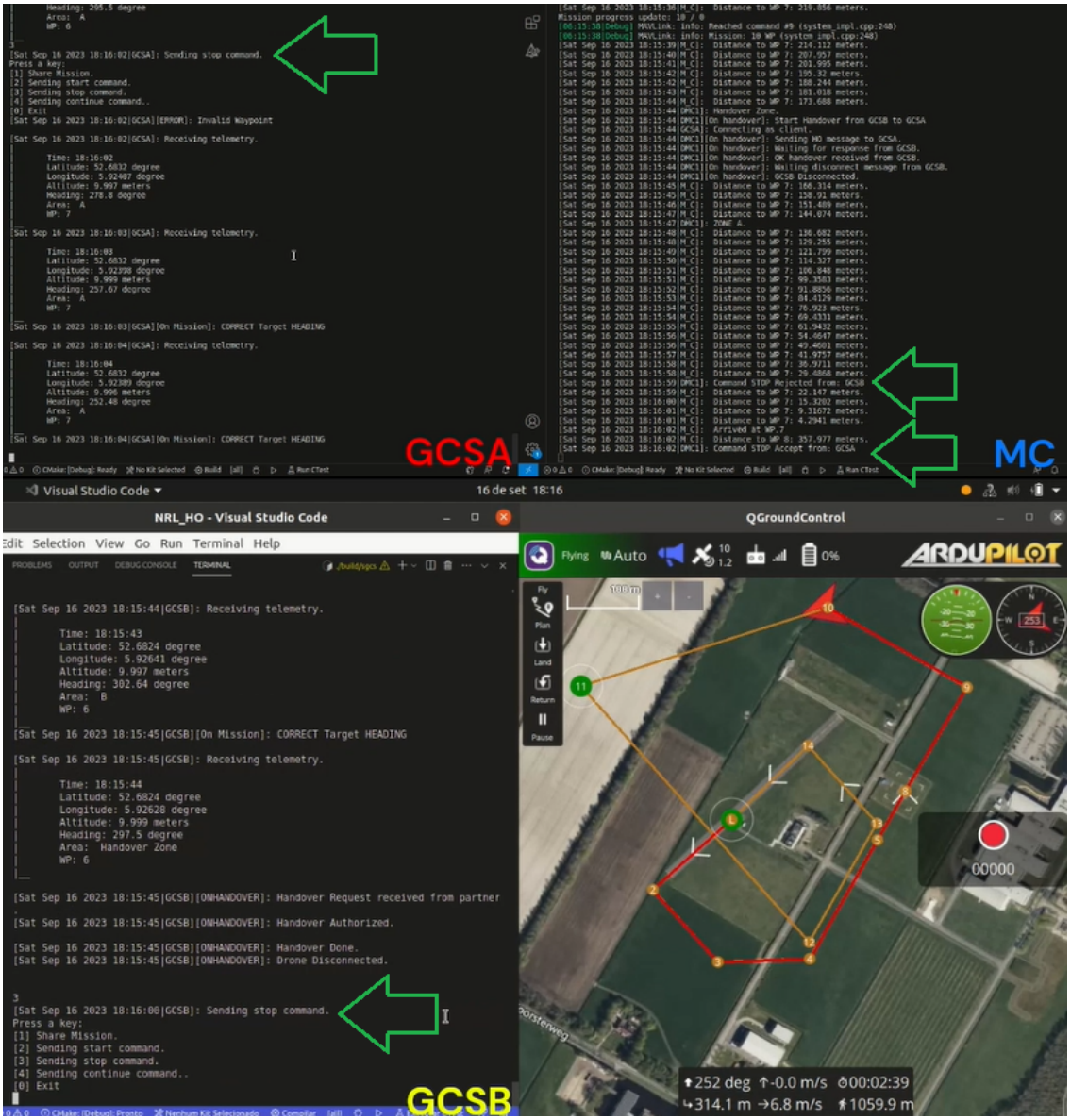


Figure 6.6: Authority Check GCSA

Figure 6.7: Authority Check GCSB

6.1.3 Emergency Procedure

In this study, the emergency procedure was exclusively activated when the drone operated in an area without GCS coverage. Figure 6.8, illustrates the drone flying out from the coverage area of GCSA.

The green arrow marks the point at which the mission computer detected that the vehicle had exited the geofence flight zone and signaled the execution of the emergency procedure. It is important to note that this study did not implement the MAVLINK commands sent from the Mission Computer to the autopilot.

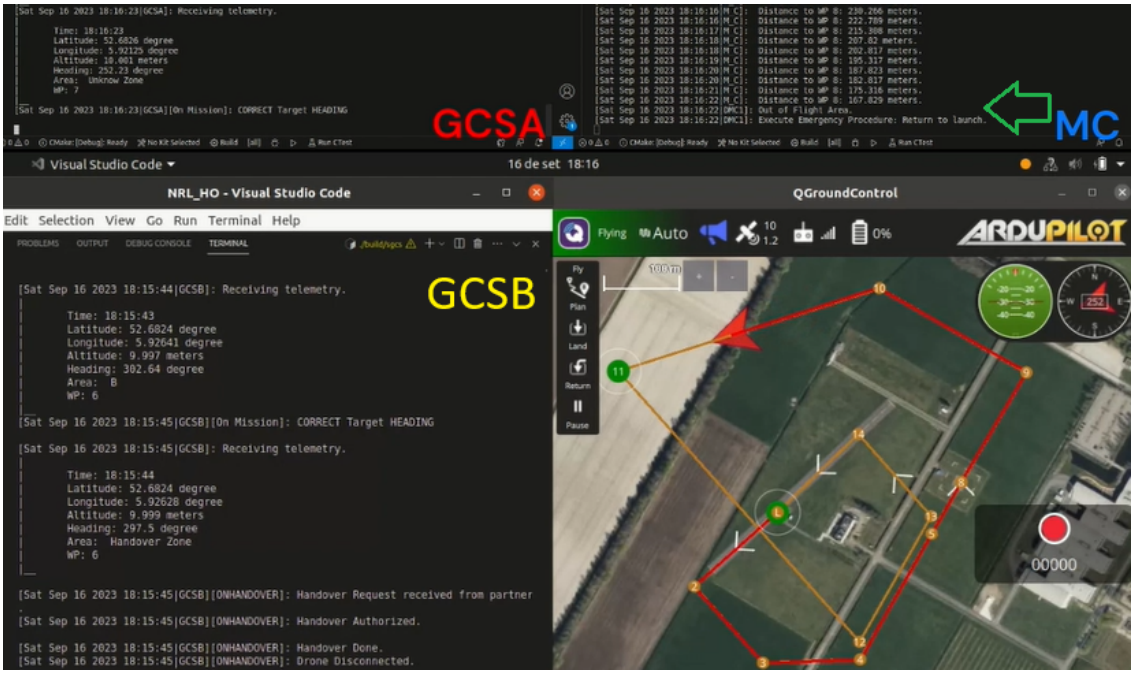


Figure 6.8: Emergency Procedure

6.2 Flight Controller Integration Results

The hardware integration results were obtained in August 2023. The drone was not allowed to fly in this area for security reasons. Since the handover only needs the GPS variation to work, the handover procedure was done by moving the drone GAIA MP160 and its flight controller through the ground, and real coordinates from GPS were obtained. The mission plan used is shown in Figure 6.9.

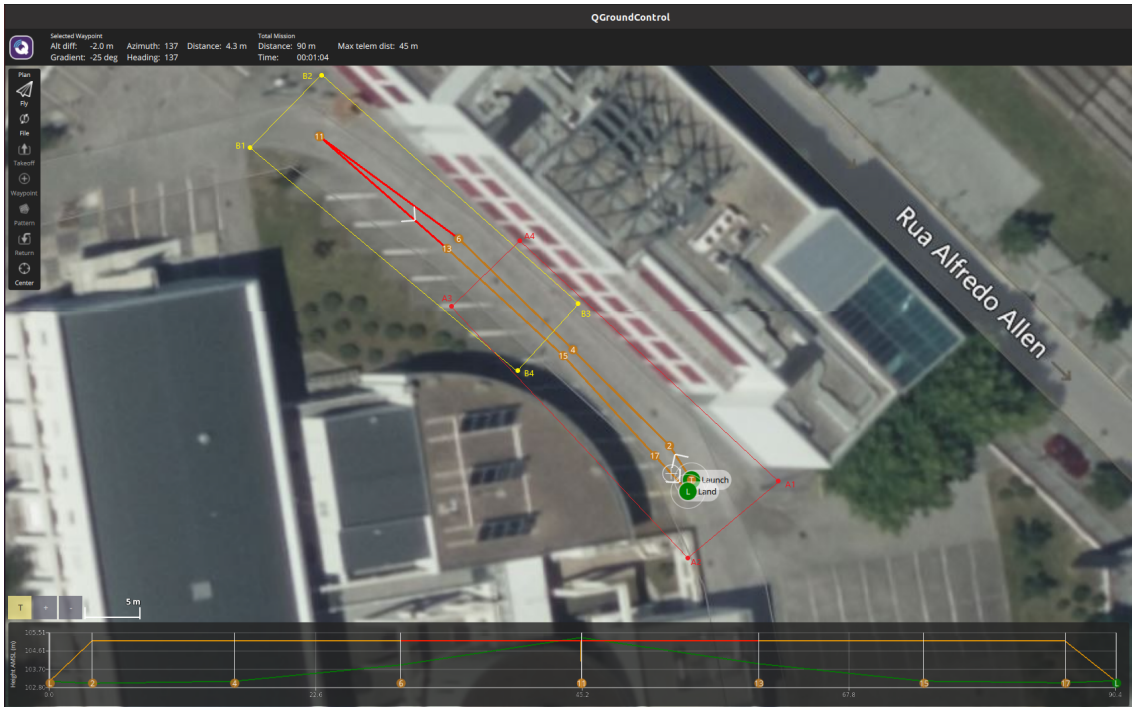


Figure 6.9: GCSA and Mission Computer: Delivery flight Handover

The red polygon and the vertices Ax represent the GCSA. The yellow polygon represents the GCSB coverage area with the vertices identified by Bx. The drone path is in orange. The takeoff and the land location are the same. The way-point number 11 represents the delivery point.

The handover region is the interception of both polygons of GCSA and GCSB. The vehicle crossed the handover region only two times, one on the delivery flight and the other on the trip back. Figure6.10 shows the handover procedure occurring in the simulation of the delivery flight, and Figure 6.11 shows the authority handover procedure occurring on the trip back.

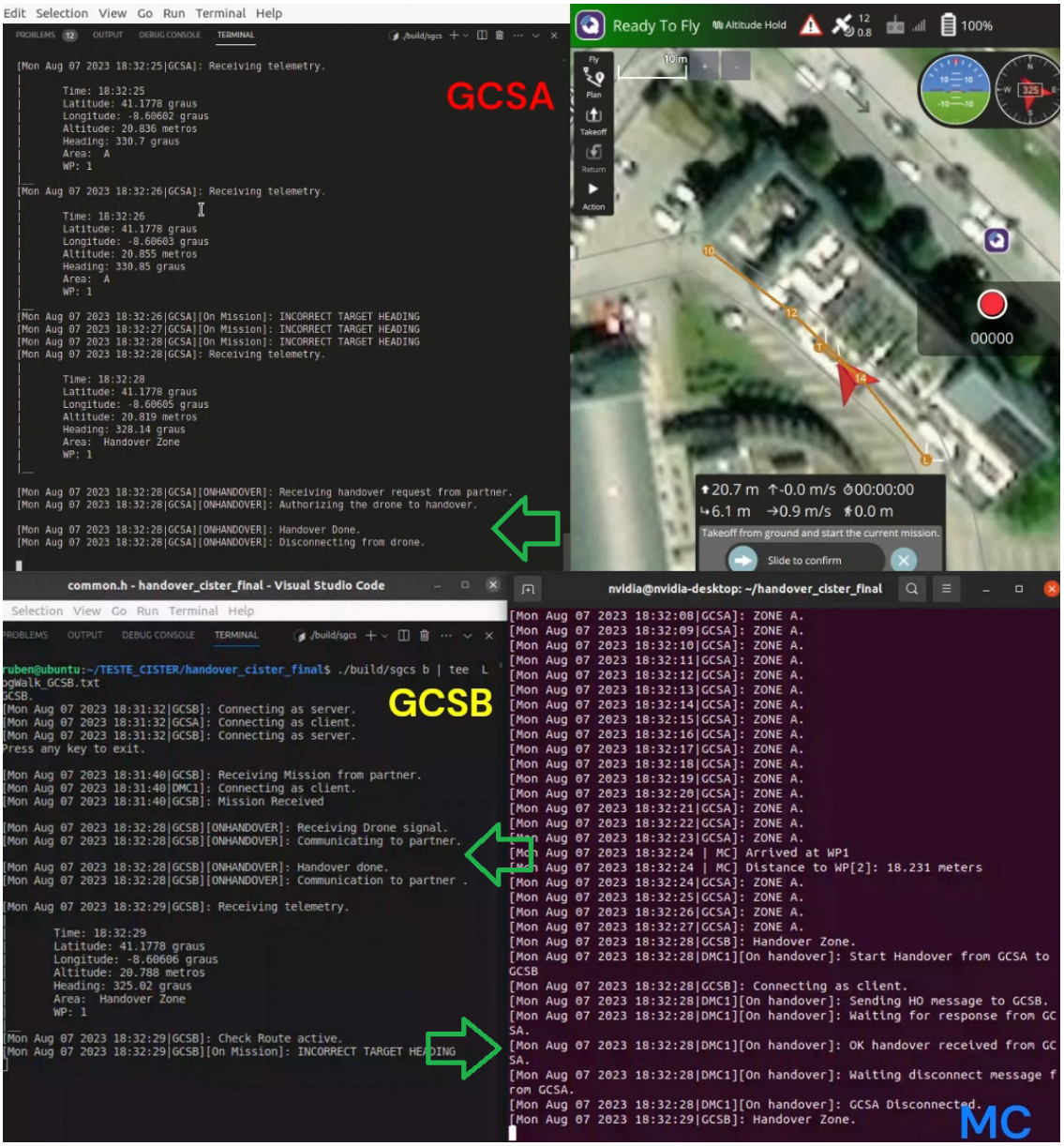


Figure 6.10: Delivery flight Handover

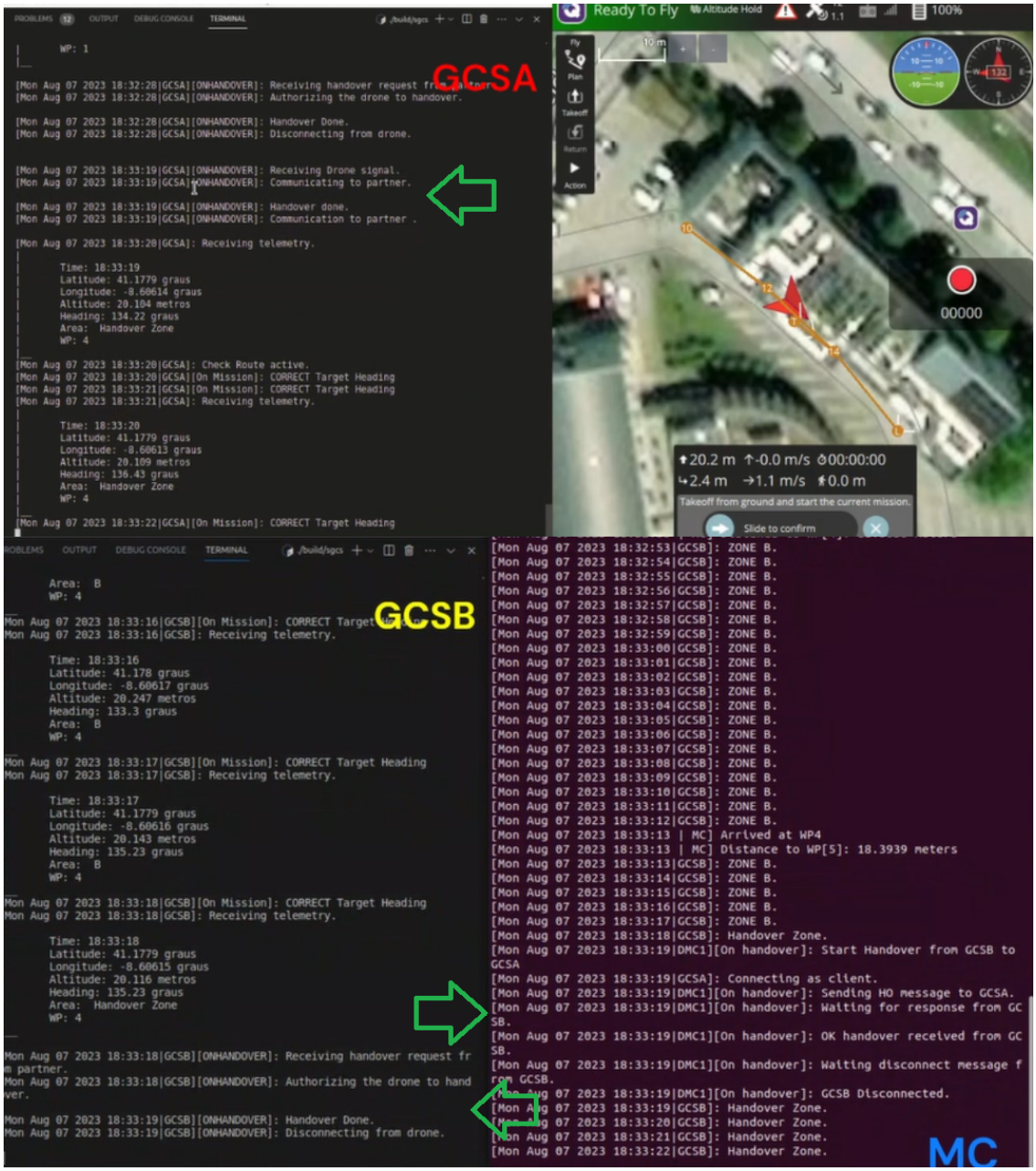


Figure 6.11: Return Flight Handover

6.3 Integration Results with Adacorsa Partners

The integration results with Adacorsa partners were obtained on 22 September 2023. The main objective was to integrate hardware and software between the partners.

Figure 6.12 shows another GAIA MP with a different hardware configuration. The vehicle payload is composed of six boxes. During this integration, the drone flew a similar path as shown in Figure 6.1, with the difference that after way point 12, the drone returned to launch due to the low battery.

The integration with the TCP server could be tested. Figure 6.13 shows both ground control stations receiving real-time data from the Mission Computer connected with the Flight Controller using the TCP server. Figure 6.14 shows the log file obtained from Mission Computer. The authority handover was done in both cases. The logs also show the command Stop being accepted or rejected regarding the drone's position.

Figure 6.12: Integration Adacorsa Partner Results: Drone

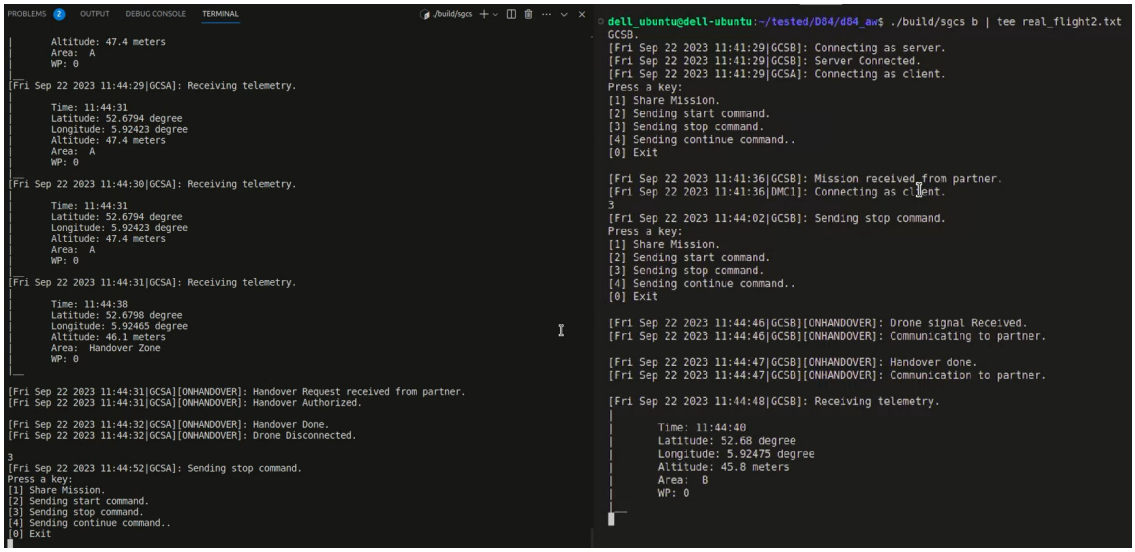


Figure 6.13: Integration Adacorsa Partner Results

```
Telemetry Server.
[MC]: TCP Connected.
[Fri Sep 22 2023 11:41:24|DMC1]: Connecting as server.
[Fri Sep 22 2023 11:41:24|DMC1]: Server Connected.
[Fri Sep 22 2023 11:41:24|DMC1]: Trying to connect with parent GCS.
[Fri Sep 22 2023 11:41:24|GCSA]: Connecting as client.
TCP Server initialized.
[Fri Sep 22 2023 11:41:25|DMC1]: Parent connected successfully.
[Fri Sep 22 2023 11:41:25|DMC1]: Out of Flight Area.
[Fri Sep 22 2023 11:41:25|DMC1]: Execute Emergency Procedure: Return to launch.
[Fri Sep 22 2023 11:41:34|DMC1]: Mission received.
[Fri Sep 22 2023 11:41:34|M_C]:  Check distance actived.
[Fri Sep 22 2023 11:41:34|GCSA]: Check geofence active.
[Fri Sep 22 2023 11:41:34|GCSB]: Check geofence active.
[Fri Sep 22 2023 11:41:36|DMC1]: Command START Accept from: GCSA
[Fri Sep 22 2023 11:41:36|DMC1]: Start Telemetry.
[Fri Sep 22 2023 11:41:39|M_C]:  Distance to WP 1: 149.427 meters.
[Fri Sep 22 2023 11:41:40|DMC1]: ZONE A.
[Fri Sep 22 2023 11:43:18|M_C]:  Distance to WP 1: 133.255 meters.
[Fri Sep 22 2023 11:43:20|M_C]:  Distance to WP 1: 127.464 meters.
[Fri Sep 22 2023 11:43:32|M_C]:  Distance to WP 1: 121.922 meters.
[Fri Sep 22 2023 11:43:34|M_C]:  Distance to WP 1: 114.677 meters.
[Fri Sep 22 2023 11:43:35|M_C]:  Distance to WP 1: 108.99 meters.
[Fri Sep 22 2023 11:43:42|M_C]:  Distance to WP 1: 54.2891 meters.
[Fri Sep 22 2023 11:43:43|M_C]:  Distance to WP 1: 47.2769 meters.
[Fri Sep 22 2023 11:43:44|M_C]:  Distance to WP 1: 40.3127 meters.
[Fri Sep 22 2023 11:43:45|M_C]:  Distance to WP 1: 33.3638 meters.
[Fri Sep 22 2023 11:43:46|M_C]:  Distance to WP 1: 26.3689 meters.
[Fri Sep 22 2023 11:43:56|DMC1]: Command STOP Accept from: GCSA
[Fri Sep 22 2023 11:44:00|DMC1]: Command STOP Rejected from: GCSB
[Fri Sep 22 2023 11:44:44|DMC1]: Handover Zone.
[Fri Sep 22 2023 11:44:44|DMC1][On handover]: Start Handover from GCSA to GCSB
[Fri Sep 22 2023 11:44:44|GCSB]: Connecting as client.
[Fri Sep 22 2023 11:44:44|DMC1][On handover]: Sending HO message to GCSB.
[Fri Sep 22 2023 11:44:44|DMC1][On handover]: Waiting for response from GCSA.
[Fri Sep 22 2023 11:44:45|DMC1][On handover]: OK handover received from GCSA.
[Fri Sep 22 2023 11:44:45|DMC1][On handover]: Waiting disconnect message from GCSA.
[Fri Sep 22 2023 11:44:45|DMC1][On handover]: GCSA Disconnected.
[Fri Sep 22 2023 11:44:46|DMC1]: ZONE B.
[Fri Sep 22 2023 11:45:05|DMC1]: Command STOP Rejected from: GCSA
[Fri Sep 22 2023 11:45:09|DMC1]: Command STOP Accept from: GCSB
[Fri Sep 22 2023 11:45:15|DMC1]: Command STOP Accept from: GCSB
[Fri Sep 22 2023 11:45:19|DMC1]: Command STOP Accept from: GCSB
[Fri Sep 22 2023 11:46:51|DMC1]: Command STOP Accept from: GCSB
[Fri Sep 22 2023 11:46:54|DMC1]: Command STOP Rejected from: GCSA
[Fri Sep 22 2023 11:48:14|DMC1]: Handover Zone.
[Fri Sep 22 2023 11:48:14|DMC1][On handover]: Start Handover from GCSB to GCSA
[Fri Sep 22 2023 11:48:14|GCSA]: Connecting as client.
[Fri Sep 22 2023 11:48:14|DMC1][On handover]: Sending HO message to GCSA.
[Fri Sep 22 2023 11:48:14|DMC1][On handover]: Waiting for response from GCSB.
[Fri Sep 22 2023 11:48:14|DMC1][On handover]: OK handover received from GCSB.
[Fri Sep 22 2023 11:48:14|DMC1][On handover]: Waiting disconnect message from GCSB.
[Fri Sep 22 2023 11:48:15|DMC1][On handover]: GCSB Disconnected.
[Fri Sep 22 2023 11:48:26|DMC1]: ZONE A.
[Fri Sep 22 2023 11:49:34|DMC1]: Command STOP Rejected from: GCSB
[Fri Sep 22 2023 11:49:37|DMC1]: Command STOP Accept from: GCSA
```

Figure 6.14: Integration Adacorsa Partner: Log Mission Compute

Chapter 7

Conclusion and Future Work

In conclusion, this thesis represents a significant step forward in Unmanned Aerial Vehicle technology and Beyond Visual Line of Sight (BVLOS) operations. The primary focus of this research was to develop techniques and procedures for a drone to perform an authority handover between multiple ground control stations, a valuable aspect of the BVLOS operations.

Throughout this study, several objectives were achieved. First, an overview of Unmanned Aerial Vehicle architectures was provided, setting the foundation for the subsequent work. The challenges associated with BVLOS operations were discussed, highlighting the importance of addressing these challenges to advance drone technology.

The research also involved surveying handover techniques applicable to BVLOS operations and examining simulation technologies relevant to Unmanned Aircraft Systems (UAS). Afterward, an authority handover procedure was implemented and integrated with partner projects of the Adacorsa project. The validation process included simulations and integration with real hardware, ensuring the developed procedure met the specified requirements.

The successful demonstration of the authority handover procedure in three scenarios validates the software component. Although challenges were encountered during the hardware integration phase, the software consistently behaved as expected and fulfilled its intended purpose.

Future works can explore scenarios involving communication losses with the ground control stations, forcing the vehicle to execute emergency procedures. Testing the safety and security layers in missions where unauthorized ground control stations attempt to connect with the vehicle would be a valuable exploration scenario.

This thesis advances the understanding of drone authority handover procedures and lays the groundwork for future research and development. The knowledge and insights gained from this work can serve as a solid foundation for addressing the evolving challenges and opportunities in drone technology and BVLOS operations.

Bibliography

Adacorsa (2023). *Project vision*. https://adacorsa.eu/. Accessed: 2023-01-18.

Allouch, Azza et al. (2019). "MAVSec: Securing the MAVLink protocol for ardupilot/PX4 unmanned aerial systems". In: *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE, pp. 621–628.

ANAC (2022). *Categoria Específica (SPEC)*. https://www.anac.pt/vPT/Generico/drones/categoria_especifica/Paginas/CategoriaEspecifica.aspx. Accessed: 2023-01-18.

Andersen, Frederik Mazur et al. (2021). "Towards SORA-compliant BVLOS communication". In: *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, pp. 1509–1519.

Andreou, Andreas et al. (2023). "UAV Trajectory Optimisation in Smart Cities using Modified A* Algorithm Combined with Haversine and Vincenty Formulas". In: *IEEE Transactions on Vehicular Technology*.

Angjo, Joana et al. (2021). "Handover management of drones in future mobile networks: 6G technologies". In: *IEEE access* 9, pp. 12803–12823.

Ardupilot (2023a). *Ardupilot Documentation*. https://ardupilot.org. Accessed: 2023-05-23.

– (2023b). *Mavproxy Documentation*. https://ardupilot.org/mavproxy/. Accessed: 2023-05-23.

– (2023c). *The Cube Orange/+ With ADSB-In Overview*. https://ardupilot.org/copter/docs/common-thecubeorange-overview.html. Accessed: 2023-05-23.

Auterion (2023). *The story of PX4 and Pixhawk*. https://auterion.com/company/the-history-of-pixhawk/. Accessed: 2023-01-04.

Ayamga, Matthew, Selorm Akaba, and Albert Apotele Nyaaba (2021). "Multifaceted applicability of drones: A review". In: *Technological Forecasting and Social Change* 167, p. 120677. issn: 0040-1625. doi: https://doi.org/10.1016/j.techfore.2021.120677. url: https://www.sciencedirect.com/science/article/pii/S0040162521001098.

Benarbia, Taha and Kyandoghere Kyamakya (2022). "A literature review of drone-based package delivery logistics systems and their implementation feasibility". In: *Sustainability* 14.1, p. 360.

Bigazzi, Luca et al. (2022). "Fast Obstacle Detection System for UAS Based on Complementary Use of Radar and Stereoscopic Camera". In: *Drones* 6.11, p. 361.

Cister (2023). *ADACORSA - Airborne data collection on resilient system architectures*. http://www.cister.isep.ipp.pt/projects/adacorsa/. Accessed: 2023-01-18.

Cunha Rocha, Marcia et al. (2023). "A WSSL Implementation for Critical Cyber-Physical Systems Applications". In: *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023*. CPS-IoT Week '23. San Antonio, TX, USA: Association for Computing Machinery, pp. 192–197. doi: 10.1145/3576914.3587507. url: https://doi.org/10.1145/3576914.3587507.

Dronecode (2023a). *MAVLink*. https://mavlink.io/en/. Accessed: 2023-02-04.

Dronecode (2023b). *MAVSDK Guide*. https://mavsdk.mavlink.io/main/en/index.html. Accessed: 2023-02-04.
– (2023c). *Open Source Autopilot*. https://px4.io/. Accessed: 2023-02-04.
– (2023d). *QGROUNDCONTROL*. http://qgroundcontrol.com/. Accessed: 2023-01-18.
Dronecode Foundation (2023). *The Dronecode Foundation*. https://www.dronecode.org/. Accessed: 2023-01-04.
EASA (2022). *"Easy Access Rules for Airworthiness and Environmental Certification (Regulation (EU) No 748/2012)."* European Union Aviation Safety Agency. European Union.
Elmeseiry, Nourhan, Nancy Alshaer, and Tawfik Ismail (2021). "A Detailed Survey and Future Directions of Unmanned Aerial Vehicles (UAVs) with Potential Applications". In: *Aerospace* 8.12. issn: 2226-4310. doi: 10.3390/aerospace8120363. url: https://www.mdpi.com/2226-4310/8/12/363.
Fu, Qixi et al. (2019). "A geofence algorithm for autonomous flight unmanned aircraft system". In: *2019 International Conference on Communications, Information System and Computer Engineering (CISCE)*. IEEE, pp. 65–69.
Hartley, Robin John ap Lewis, Isaac Levi Henderson, and Chris Lewis Jackson (2022). "BVLOS Unmanned Aircraft Operations in Forest Environments". In: *Drones* 6.7, p. 167.
Kwon, Young-Min et al. (2018). "Empirical Analysis of MAVLink Protocol Vulnerability for Attacking Unmanned Aerial Vehicles". In: *IEEE Access* 6, pp. 43203–43212. doi: 10.1109/ACCESS.2018.2863237.
Lorenz Meier (2023). *Pixhawk*. https://pixhawk.org/. Accessed: 2023-01-04.
Matalonga, Santiago et al. (2022). "A review of the legal, regulatory and practical aspects needed to unlock autonomous beyond visual line of sight unmanned aircraft systems operations". In: *Journal of Intelligent & Robotic Systems* 106.1, pp. 1–13.
Neji, Najett, Tumader Mostfa, and Yasmina Bestaoui Sebbane (Apr. 2019). "Technology Assessment for Radio Communication between UAV and Ground: Qualitative Study and Applications". en. In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*. Kuala Lumpur, Malaysia: IEEE, pp. 1–6. isbn: 978-1-72811-217-6. doi: 10.1109/VTCSpring.2019.8746306. url: https://ieeexplore.ieee.org/document/8746306/.
Noor, Fazal et al. (2020). "A review on communications perspective of flying ad-hoc networks: key enabling wireless technologies, applications, challenges and open research topics". In: *Drones* 4.4, p. 65.
Nvidia (2023). *Nvidia Jetson*. https://www.nvidia.com/. Accessed: 2023-05-23.
Open Robotics (2023). *Gazebo*. https://gazebosim.org/home. Accessed: 2023-01-04.
Politi, Elena et al. (2021). "A Survey of UAS Technologies to Enable Beyond Visual Line Of Sight (BVLOS) Operations." In: *VEHITS*, pp. 505–512.
PX4 (2023a). *Gazebo Simulation*. https://docs.px4.io/main/en/simulation/gazebo.html. Accessed: 2023-01-04.
– (2023b). *PX4 Documentation*. https://docs.px4.io/main/en/flight_controller/cubepilot_cube_orange.html. Accessed: 2023-05-23.
Rejeb, Abderahman et al. (2022). "Drones in agriculture: A review and bibliometric analysis". In: *Computers and Electronics in Agriculture* 198, p. 107017.
Shayea, Ibraheem et al. (2022). "Handover Management for Drones in Future Mobile Networks—A Survey". In: *Sensors* 22.17, p. 6424.
Stevens, Mia N, Hossein Rastgoftar, and Ella M Atkins (2017). "Specification and evaluation of geofence boundary violation detection algorithms". In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, pp. 1588–1596.

Suzel Tunes (2021). *Fora do campo de visão*. `https://revistapesquisa.fapesp.br/fora-do-campo-de-visao/`. Accessed: 2023-01-18.