UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Enhancing Unmanned Aerial Vehicle (UAV) Mission Planning and Communication: Integration of PX4 Autopilot and Mission Planner Ground Control Station with MAVLink Protocol

Diploma Thesis

Zafeiri Stamatia Varvara

Supervisor: Stamoulis Georgios

Volos 2024

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Enhancing Unmanned Aerial Vehicle (UAV) Mission Planning and Communication: Integration of PX4 Autopilot and Mission Planner Ground Control Station with MAVLink Protocol

Diploma Thesis

Zafeiri Stamatia Varvara

Supervisor: Stamoulis Georgios

Volos 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Βελτιστοποίηση Σχεδιασμού Αποστολών και Επικοινωνίας για μη Επανδρωμένα Εναέρια Οχήματα (UAV): Ενοποίηση του PX4 AutoPilot και του Σταθμού Εδάφους Mission Planner με το Πρωτόκολλο MAVLink

Διπλωματική Εργασία

Ζαφείρη Σταματία Βαρβάρα

Επιβλέπων/πουσα: Σταμούλης Γεώργιος

Βόλος 2024

Approved by the Examination Committee:

Supervisor **Stamoulis Georgios**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Evmorfopoulos Nestor**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Kolomvatsos Konstantinos**
MSc Instructor, Department of Electrical and Computer Engineering, University of Thessaly

Date of approval: 8-3-2024

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Georgios Stamoulis for their unwavering support, guidance, and mentorship not only throughout this research but also during the entirety of my academic journey.

I am profoundly grateful to Professor Evmorfopoulos Nestor and Professor Kolomvatsos Konstantinos for being a member of the examination committee of my thesis. I extend my sincere appreciation to the faculty members of the Department of Electrical and Computer Engineering at Univercity of Thessaly for their encouragement, knowledge sharing, and constructive feedback that enriched the quality of this work.

I am deeply thankful to my family and friends for their encouragement, patience, and belief in my abilities. I would like to extend a special acknowledgment to my cousins, whose constant presence has been a wellspring of motivation, inspiration, and encouragement, and my best friend for always being there to cheer me up during challenging times. Additionally, I would like to express profound gratitude to a special person in my life who has been a constant pillar of support and love throughout every step of this thesis.

Lastly, I would like to acknowledge the countless researchers, authors, and contributors whose work has shaped the landscape of this field. Their pioneering efforts have paved the way for this study.

This thesis is dedicated with deep affection to my beloved grandmother, who embarked on her journey to heaven earlier than anticipated, missing the chance to witness my dreams becoming reality. Her memory remains a guiding light, inspiring me to reach for the heights she believed I could attain.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Zafeiri Stamatia Varvara

Diploma Thesis

Enhancing Unmanned Aerial Vehicle (UAV) Mission Planning and Communication: Integration of PX4 Autopilot and Mission Planner Ground Control Station with MAVLink Protocol

Zafeiri Stamatia Varvara

Abstract

Unmanned Aerial Vehicles (UAVs) have gained significant prominence in various industries, driven by advancements in autopilot systems, communication protocols and ground control systems. This thesis explores the integration of the PX4 Autopilot system and Mission Planner Ground Control Station (GCS) through the MAVLink Protocol to enhance UAV mission planning and communication. The investigation focuses on the MAVLink Protocol, examining its implementation in both the PX4 Autopilot and Mission Planner GCS and important Protocol discrepancies are identified and analyzed, coming to the conclusion that those differences affect the correctness of the mission uploaded but not the uploading process. The insights gained pave the way for successful connection, improved operational efficiency and mission success.

Διπλωματική Εργασία

Βελτιστοποίηση Σχεδιασμού Αποστολών και Επικοινωνίας για μη Επανδρωμένα Εναέρια Οχήματα (UAV): Ενοποίηση του PX4 AutoPilot και του Σταθμού Εδάφους Mission Planner με το Πρωτόκολλο MAVLink

Ζαφείρη Σταματία Βαρβάρα

Περίληψη

Τα μη επανδρωμένα εναέρια οχήματα (UAV) έχουν κερδίσει σημαντική θέση σε διάφορες βιομηχανίες, χάρη στην πρόοδο των συστημάτων αυτόματου πιλότου, των πρωτόκολλων επικοινωνίας και των σταθμών εδάφους. Αυτή η διπλωματική εργασία διερευνά την ενοποίηση του συστήματος αυτόματου πιλότου PX4 AutoPilot και του σταθμού εδάφους (GCS) Mission Planner μέσω του Πρωτοκόλλου MAVLink για την βελτιστοποίηση του σχεδιασμού αποστολών και επικοινωνίας. Η έρευνα επικεντρώνεται στο πρωτόκολλο MAVLink, εξετάζοντας την εφαρμογή του τόσο στο PX4 Autopilot όσο και στο Mission Planner GCS και σημαντικές διαφορές στο πρωτόκολλο εντοπίζονται και αναλύονται, καταλήγοντας στο συμπέρασμα ότι αυτές οι διαφορές επηρεάζουν την ορθότητα της αποστολής στάλθηκε αλλά όχι τη διαδικασία αποστολής. Οι γνώσεις που αποκτήθηκαν ανοίγουν το δρόμο για επιτυχημένη σύνδεση, βελτιωμένη λειτουργικότικα και επιτυχία αποστολών.

Table of contents

List of figures

List of tables

Abbreviations

GCS	Ground Control Station
MP	Mission Planner
SITL	Software In The Loop
UAV	Unmanned Aerial Vehicle
WP	Waypoint

Chapter 1

Introduction

1.1 Motivation

Unmanned Aerial Vehicles (UAVs), commonly known as drones, have transformed industries worldwide, offering unprecedented capabilities in various applications. From aerial photography and surveillance to precision agriculture and delivery services, UAVs have redefined how we interact with the world around us. However, the seamless and safe operation of these unmanned aircraft presents significant challenges, especially in complex and dynamic environments.

At the heart of efficient UAV operation lies the significance of advanced autopilot systems. UAV autopilot systems, comprising sophisticated hardware and software components, play a pivotal role in automating and controlling flight dynamics, navigation and mission execution. They serve as the technological backbone, enabling autonomous or semi-autonomous flight and ensuring optimal performance under diverse conditions.

The importance of UAV autopilot systems emerges from their ability to provide precise control, stability, and responsiveness to unmanned aerial vehicles. By continuously monitoring and adjusting the aircraft's attitude, altitude, and heading, these systems ensure stable flight dynamics, regardless of challenging weather conditions or turbulent environments. Furthermore, they incorporate advanced navigation algorithms, such as GPS and inertial sensors, facilitating accurate position estimation and efficient path planning. This capability allows UAVs to follow predefined flight paths or waypoints, making them ideal for aerial surveying, mapping, or search and rescue missions.

Autonomous operations represent a key aspect of UAV autopilot systems, empowering

unmanned aircraft to execute pre-programmed missions or tasks with minimal human intervention. By leveraging sophisticated automation algorithms, these systems enable drones to perform complex tasks efficiently, such as surveillance over large areas, infrastructure inspections, or delivering essential supplies to remote locations. Additionally, autopilot systems enhance operational safety through the inclusion of fail-safe mechanisms, geofencing, and collision avoidance systems, mitigating risks and preventing accidents even in critical situations.

Moreover, UAV autopilot systems facilitate the integration and synchronization of various sensors and payloads, offering an extensive range of data collection capabilities. By seamlessly incorporating cameras, LiDAR, thermal imagers or scientific instruments, these systems enable UAVs to collect valuable data for research, environmental monitoring or data-driven decision-making.

1.2 Contribution

This thesis aims to provide significant contributions to the field of UAV autopilot systems. The following contributions are anticipated:

Enhanced Connectivity and Compatibility: The research focuses on developing a robust connection between PX4 autopilot and Mission Planner, improving interoperability and compatibility. The proposed mechanism enables seamless collaboration and integration between the autopilot and ground control station, enhancing mission planning, monitoring, and control efficiency for unmanned aerial vehicles.

Practical Implementation and Documentation: The research provides practical implementation guidelines and comprehensive documentation for connecting PX4 autopilot with Mission Planner. This includes step-by-step instructions and configuration details to facilitate successful replication of the connection process. The aim is to support wider adoption and practical implementation of the connected system in real-world scenarios.

Through these contributions, this thesis advances the field of UAV autopilot systems by improving connectivity, compatibility, and practical implementation. The outcomes provide valuable insights and resources for researchers, developers, and industry professionals integrating PX4 autopilot with Mission Planner, ultimately enhancing the functionality, efficiency, and effectiveness of unmanned aerial vehicles in various applications.

1.3 Thesis Structure

The rest of the thesis is structured as follows:

- Chapter 2 provides a comprehensive overview and detailed explanation of the key components involved in unmanned aerial vehicle (UAV) operations, including UAV autopilot systems, ground control stations (GCS), and the MAVLink communication protocol.
- Chapter 3 delves into the PX4 autopilot system and Mission Planner ground control station (GCS), along with their respective MAVLink protocols.
- Chapter 4 focuses on the installation and setup process for the PX4 autopilot system and Mission Planner ground control station (GCS).
- Chapter 5 presents the methodology and testing process for establishing communication between the PX4 autopilot system and the Mission Planner ground control station (GCS) using their respective MAVLink protocols.
- Chapter 6 gives an overview of related work.
- Chapter 7 provides a conclusion of this Thesis.

Chapter 2

Background

2.1 UAVs

UAVs, which stands for Unmanned Aerial Vehicle, often known as drones, have garnered significant consideration in various military and civilian service disciplines due to their improved stability and endurance in many missions. A UAV is a term used to describe a type of pilotless aircraft that is able to fly and maintain altitude without the need for a human operator on board, offers more cost-effective operations than comparable manned systems, and performs important tasks cost-effectively without endangering human life. UAVs can be flown remotely by receiving control instructions from a ground control station (GCS) through a remote control. The autopilot and other sensors, including the global positioning system (GPS) and inertial measurement units (IMU), enable the UAVs to carry out control tasks onboard as well. [3]

2.2 Autopilot

An autopilot is a key component of unmanned aerial vehicles (UAVs) that is responsible for controlling and guiding the aircraft's flight operations. It is a sophisticated system consisting of hardware and software that automates various aspects of the flight, reducing the need for direct human intervention.

The autopilot system continuously monitors and adjusts the UAV's flight parameters, ensuring stability, navigation, and control throughout the mission. It relies on sensors, such as accelerometers, gyroscopes, GPS receivers, and altimeters, to gather data about the UAV's

position, orientation, velocity, and environmental conditions. This information is then processed by the autopilot's software algorithms, which make real-time decisions and generate appropriate control commands.

Open-source autopilot frameworks are incredibly flexible and can be used to build specialized autopilots. These autopilots are ”MOTS” or ”modifiable-off-the-shelf” solutions since they can be enhanced with specialized software and hardware. These autopilots' source code and in-depth computer science understanding are required for modification. As there is often a well-documented procedure for integrating modules that comply to a certain interface, integrating new sensors is quite simple. However, altering the UAV's behavior might be difficult and come with possible safety risks, especially for any functionality between state estimation and actuation. [4]

2.3 Ground Control Station

A Ground Control Station (GCS) is an important component of UAV operations, serving as the interface for controlling and monitoring the UAV during missions. It provides operators with real-time data, visualization, control capabilities and the abillity to define flight paths, waypoints and specific tasks for the UAV from a remote location.

During missions, the GCS receives telemetry data from the UAV in real-time, including altitude, speed, battery status, and sensor readings. This data provides operators with a comprehensive understanding of the UAV's performance, allowing for informed decision-making and timely responses to critical situations. This direct interaction empowers operators to adapt to changing conditions and execute mission objectives on-the-fly. Additionally, the GCS often includes features such as data logging, video streaming, and post-flight analysis. These capabilities enable operators to record flight data and video footage, facilitating performance evaluation, troubleshooting, and reporting.

Its mission planning, real-time monitoring, and control functionalities enable efficient and safe UAV operations, ultimately contributing to the success of unmanned aerial missions.

2.4 MAVLink

MAVLink, short for Micro Air Vehicle Link, is a lightweight communication protocol specifically designed for UAVs and systems that operate in the unmanned vehicle domain. It is an open-source protocol that facilitates the exchange of information between onboard systems and ground control stations.

The MAVLink protocol defines a set of message types and communication rules that enable the transmission of data and commands between the UAV's autopilot system and the ground control station. These messages cover a wide range of information, including telemetry data such as vehicle position, altitude, velocity, battery status, sensor readings, and more.

One of the key advantages of MAVLink is its flexibility and scalability. The protocol allows for the customization and addition of new message types, making it adaptable to various types of UAV platforms and payloads. This flexibility enables seamless integration with different hardware and software components, fostering interoperability among UAV systems from different manufacturers. The MAVLink protocol supports various communication interfaces, including serial ports, UDP/IP networks, and wireless links. It is designed to be lightweight and efficient, allowing for low-latency communication even in bandwidth-limited environments. [5]

The following diagram shows how MAVLink is used. More specifically Flight Controller is using the MAVLink Protocol to connect to :

- GCS through Telemetry Radio.
- Payload (Cameras)

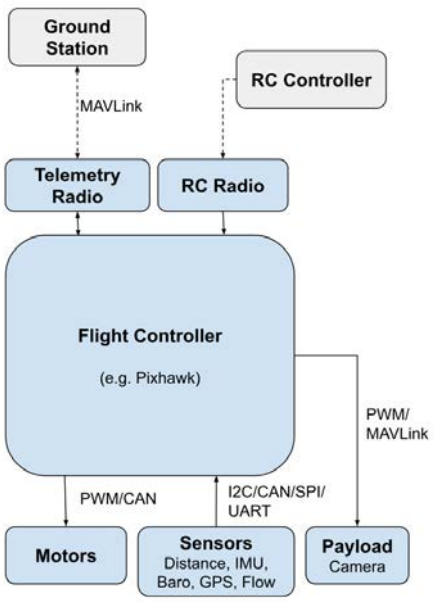

Figure 2.1: Connection through MAVLink [1]

Chapter 3

System overview

3.1 PX4 Autopilot

PX4 Autopilot is an open-source flight control software designed for autonomous control and navigation of drones and unmanned aerial vehicles (UAVs). It provides a comprehensive collection of features and algorithms for developers to exchange technology to create specialized drone application solutions. It supports a variety of vehicles kinds, fully manual, partially aided, and fully autonomous flight modes. PX4 offers a standard for distributing drone hardware support and software stacks, enabling an ecosystem to develop and maintain hardware and software in a scalable manner. PX4 Autopilot is frequently used in many different applications, such as aerial photography, mapping, surveillance, and research, thanks to its strong and adaptable design. [6]

3.2 MAVLink Protocol of PX4

The MAVLink protocol is essential for establishing communication between the PX4 Autopilot and the ground control station (GCS).In order to facilitate the interchange of telemetry data, orders, and status updates between the UAV and the GCS, it offers a standard and lightweight messaging architecture. The C programming language is used in PX4 Autopilot to construct the MAVLink protocol. The source code for the MAVLink module can be found at the following location within the PX4 source tree:

src/modules/mavlink/mavlink_mission.cpp

As of the time of writing, the implementation status of the MAVLink protocol in PX4

Autopilot is as follows:

- Flight plan missions: In accordance with the MAVLink protocol's standards, the protocol provides uploading, downloading, clearing missions, and keeping track of mission progress.

- Geofence missions: The protocol enables the design and enforcement of geographic boundaries by supporting geofence missions as specified in the MAVLink standard.

- Rally point missions: At this time, PX4 Autopilot's protocol does not handle rally point missions. Predetermined GPS coordinates are referred to as rally points and can be utilized for navigation or other purposes.

Depending on the type of operation, the implementation treats mission operation cancellation differently. The system is placed in an idle state after a mission download cancellation. However, PX4 Autopilot currently resends the mission request until the operation times out, therefore mission upload cancellation is not entirely allowed. Understanding the present state of the MAVLink protocol within PX4 Autopilot, including the supported mission features and the behavior of mission operation cancellation, is made possible by these implementation details. In conclusion, the MAVLink protocol, which enables dependable and consistent communication between the UAV and the GCS, is a crucial component of the PX4 Autopilot system. [2]

3.3 Mission Planner

Mission Planner is a powerful ground control station (GCS) software application designed for unmanned aerial vehicles (UAVs) and offers a user-friendly interface and supports various autopilot systems. With Mission Planner operators can plan and customize UAV missions by defining waypoints, flight paths, and specific actions and be provided with real-time telemetry data, including UAV position, altitude, battery status and sensor readings. The software also supports advanced features such as geofencing and mission simulation. With its capabilities, Mission Planner enhances the efficiency and safety of UAV operations by facilitating effective mission planning, execution, and monitoring. [7]

3.4 MAVLink Protocol of ArduPilot/Mission Planner

In the context of Mission Planner, the MAVLink protocol is a fundamental component that enables seamless communication between the ground control station (GCS) software and the unmanned aerial vehicle (UAV). The MAVLink protocol in Mission Planner also supports advanced features such as geofencing, allowing operators to define and enforce geographic boundaries to ensure safe and controlled UAV operations.

Code source:

/libraries/GCS_MAVLink/GCS_Common.cpp

Mission upload, download, clearing missions, and monitoring progress are supported.

ArduPilot implements also partial mission upload using MISSION_WRITE_PARTIAL_LIST, but not partial mission download (MISSION_REQUEST_PARTIAL_LIST). Partial mission upload/download is not an official/standardised part of the mission service. [2]

3.5 Differences Between the MAVLink protocols

3.5.1 Flight Plan Missions

This section defines the specifications of the following MAVLink protocol operations.

Upload a Mission to the Vehicle

The diagram below shows the communication sequence to upload a mission to a drone (assuming all operations succeed).

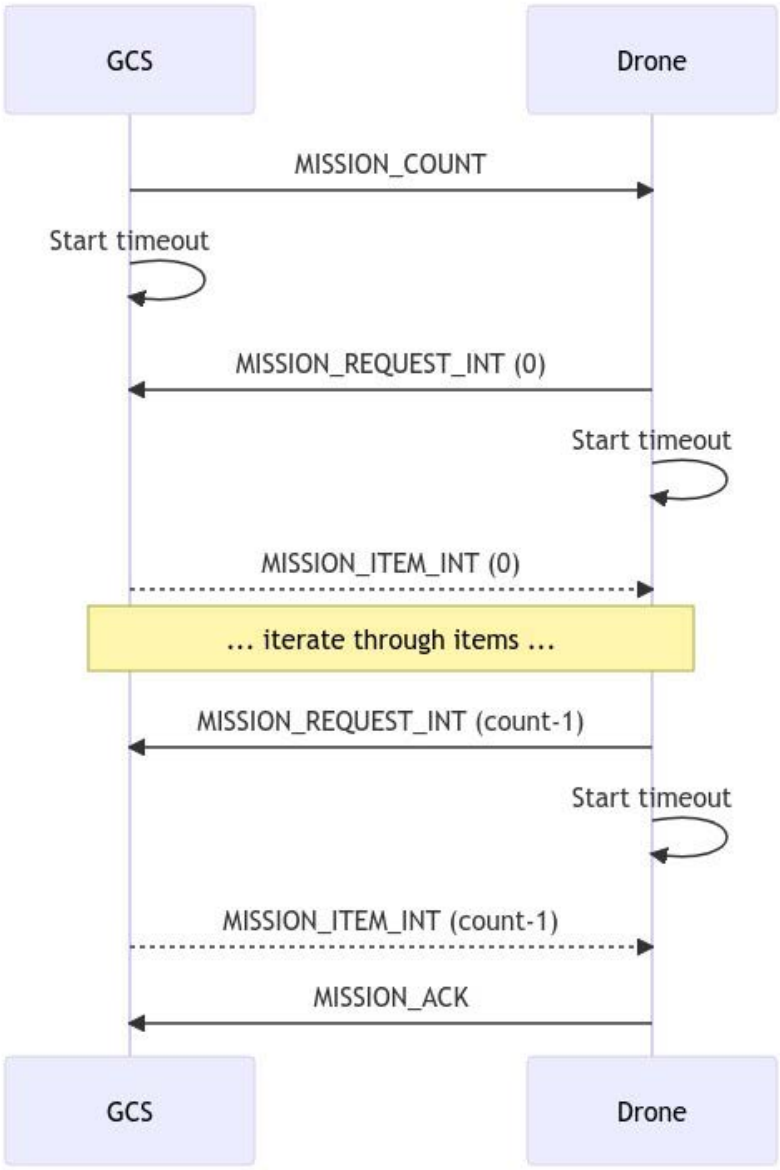

Figure 3.1: Mission upload diagram [2]

In more detail, the sequence of operations is:

1. GCS sends MISSION_COUNT including the number of mission items to be uploaded (count).
    - A timeout must be started for the GCS to wait on the response from Drone (MISSION_REQUEST_INT).
2. Drone receives message and responds with MISSION_REQUEST_INT requesting the first mission item (seq==0).
    - A timeout must be started for the Drone to wait on the MISSION_ITEM_INT

response from GCS.

3. GCS receives MISSION_REQUEST_INT and responds with the requested mission item in a MISSION_ITEM_INT message.

4. Drone and GCS repeat the MISSION_REQUEST_INT/MISSION_ITEM_INT cycle, iterating seq until all items are uploaded (seq==count-1).

5. After receiving the last mission item the drone responds with MISSION_ACK with the type of MAV_MISSION_ACCEPTED indicating mission upload completion/success.

    - The drone should set the new mission to be the current mission, discarding the original data.
    - The drone considers the upload complete.

6. GCS receives MISSION_ACK containing MAV_MISSION_ACCEPTED to indicate the operation is complete. [2]

Download a Mission from the Vehicle

The diagram below shows the communication sequence to download a mission from a drone (assuming all operations succeed).

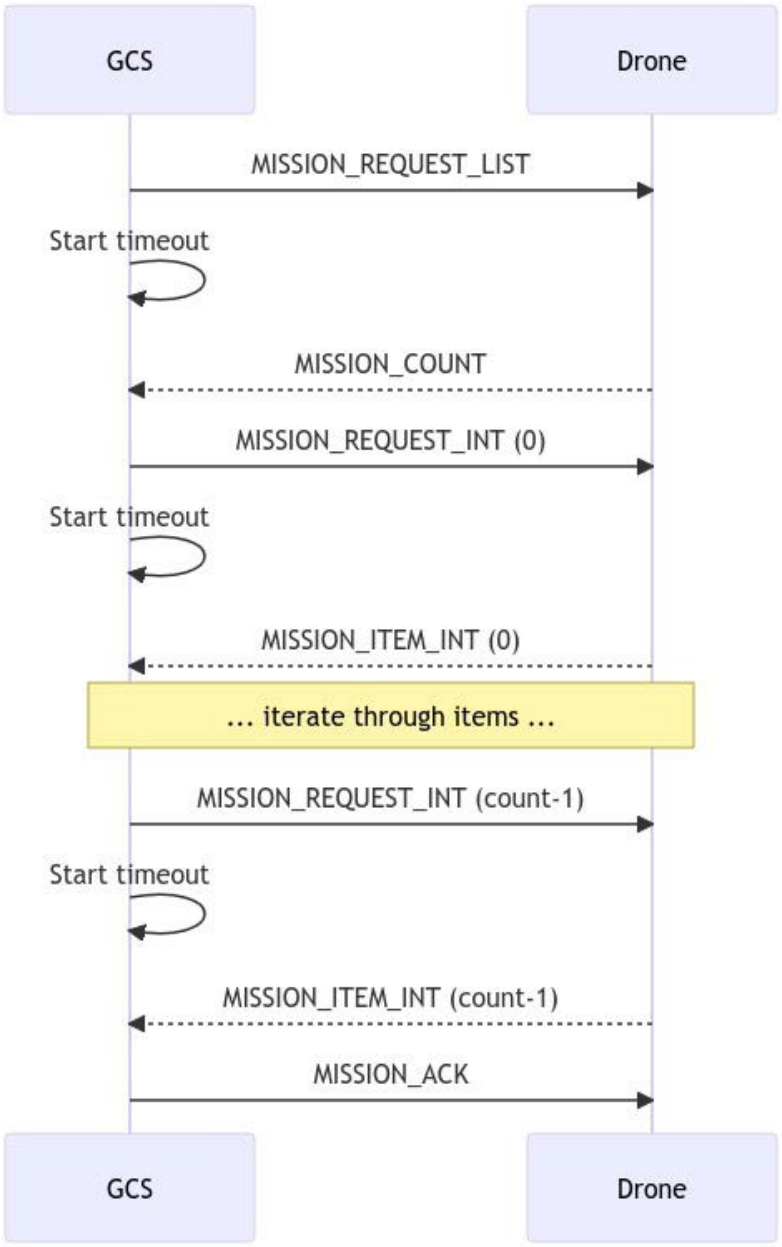

Figure 3.2: Mission download diagram [2]

The sequence is similar to that for uploading a mission. The main difference is that the client (e.g. GCS) sends MISSION_REQUEST_LIST, which triggers the autopilot to respond with the current count of items. This starts a cycle where the GCS requests mission items, and the drone supplies them. [2]

Monitor Mission Progress

GCS/developer API can monitor progress by handling the appropriate messages sent by the drone:

- The vehicle must broadcast a MISSION_ITEM_REACHED message whenever a new mission item is reached. The message contains the seq number of the current mission item.

- The vehicle must also broadcast a MISSION_CURRENT message if the current mission item is changed. [2]

Clear Missions

The diagram below shows the communication sequence to clear the mission from a drone (assuming all operations succeed).

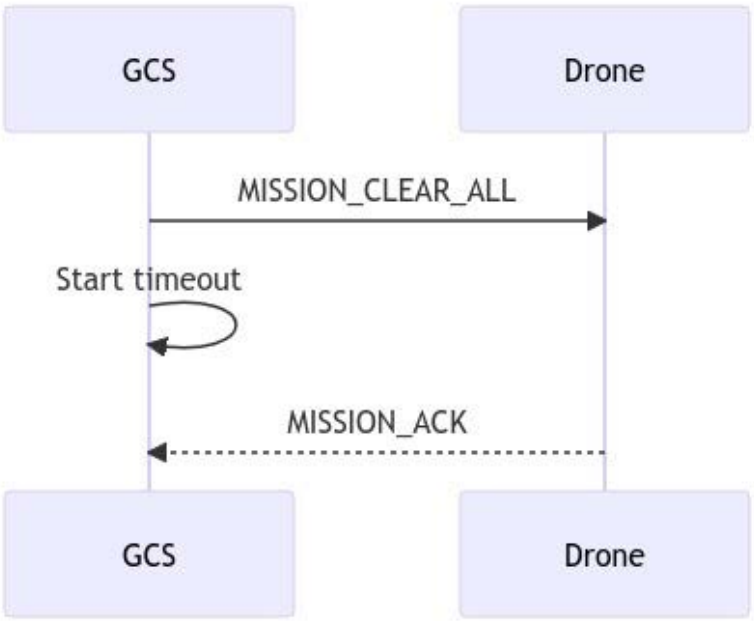

Figure 3.3: Clear mission diagram [2]

In more detail, the sequence of operations is:

1. GCS/API sends MISSION_CLEAR_ALL

    - A timeout is started for the GCS to wait on MISSION_ACK from Drone.

2. Drone receives the message, and clears the mission from storage.

3. Drone responds with MISSION_ACK with result type of MAV_MISSION_ ACCEPTEDMAV_MISSIO_RESULT.

4. GCS receives MISSION_ACK and clears its own stored information about the mission. The operation is now complete. [2]

3.5.2 How ArduPilot differs

ArduPilot's implementation differs from this specification (non-exhaustively):

- The first mission sequence number (seq==0) is populated with the home position of the vehicle instead of the first mission item.
- Mission uploads are not "atomic". An upload that fails (or is canceled) part-way through will not match the pre-update state. Instead it may be a mix of the original and new mission.
- Even if upload is successful, the vehicle mission may not match the version on the uploading system (and if the mission is then downloaded it will differ from the original).
  - There is rounding on some fields (and in some cases internal maximum possible values due to available storage space). Failures can occur if you do a straight comparison of the float params before/after upload.
- A MISSION_ACK returning an error value (NACK) does not terminate the upload (i.e. it is not considered an unrecoverable error). As long as ArduPilot has not yet timed-out a system can retry the current mission item upload.
- A mission cannot be cleared while it is being executed (i.e. while in Auto mode). Note that a new mission can be uploaded (even a zero-size mission - which is equivalent to clearing).
- Explicit cancellation of operations is not supported. If one end stops communicating the other end will eventually timeout and reset itself to an idle/ready state. [2]

Chapter 4

Setup

4.1 Prerequisites

- Development computer OSX 10.15 or later Ubuntu Linux 18.04 to 22.04
- The following instructions set up a PX4 development environment on the Ubuntu Linux LTS (opens new window) versions supported by PX4. This includes: 18.04 (Bionic Beaver), 20.04 (Focal Fossa), and Ubuntu 22.04 (Jammy Jellyfish).
- Bash scripts are provided to simplify the process. They are intended to be run on clean Ubuntu LTS installations, and may not work if run ”on top” of an existing system, or on a different Ubuntu release.
- Access the terminal and proceed by installing the ’git’ command using the following command:

```
$ sudo apt install git
```

4.2 Installation of PX4 Autopilot

4.2.1 Cloning the source code

Ensure that you have Git installed on your system. To download the PX4 source code, execute the following command using ’git clone’:

```
$ git clone https://github.com/PX4/PX4-Autopilot.git --recursive
```

```
admtina@ubuntu:~$ git clone https://github.com/PX4/PX4-Autopilot.git --recursive
Cloning into 'PX4-Autopilot'...
remote: Enumerating objects: 442993, done.
remote: Counting objects: 100% (1786/1786), done.
remote: Compressing objects: 100% (969/969), done.
remote: Total 442993 (delta 1114), reused 1241 (delta 796), pack-reused 441207
Receiving objects: 100% (442993/442993), 209.71 MiB | 23.71 MiB/s, done.
Resolving deltas: 100% (327072/327072), done.
```

Figure 4.1: Clone PX4 command

4.2.2 Installing PX4

Once the repository has been cloned, execute the 'ubuntu.sh' script in a bash shell without any arguments. This will initiate the installation of PX4 autopilot along with the Gazebo classic simulator and all their dependencies. During the script execution, you will encounter several prompts; respond to these prompts by entering "Y" into the terminal.

```
$ bash ./PX4-Autopilot/Tools/setup/ubuntu.sh
```

```
admtina@ubuntu:~$ bash ./PX4-Autopilot/Tools/setup/ubuntu.sh
Ubuntu 20.04

Installing PX4 general dependencies
```

Figure 4.2: Install PX4

After the script has finished its execution, it is recommended to restart your computer. Please note that an error occurred while running the previous command.
We can disregard it for now, and we'll assess whether it has any impact on our operations later.

```
Successfully built cerberus empy future
ERROR: pandas 2.0.3 has requirement python-dateutil>=2.8.2, but you'll have python-dateutil 2.7.3 which
is incompatible.
```

Figure 4.3: Python panda error

4.2.3 Testing PX4

Once the script has run successfully, it's time to test the PX4 installation. We will accomplish this by launching PX4 within the Gazebo Classic simulator.
Start by navigating to the PX4 directory using the 'cd' command.

```
$ cd /PX4-Autopilot/
```

Initiate PX4 with Gazebo Classic by utilizing the ’make’ command.

```
$ make px4_sitl gazebo-classic
```

```
admtina@ubuntu:~/PX4-Autopilot$ make px4_sitl gazebo-classic
-- PX4 version: v1.14.0-beta2-288-g0196241c10 (1.14.0)
```

Figure 4.4: Start PX4 with Gazebo Classic

Upon launching Gazebo, you will encounter a window that represents the simulator interface.

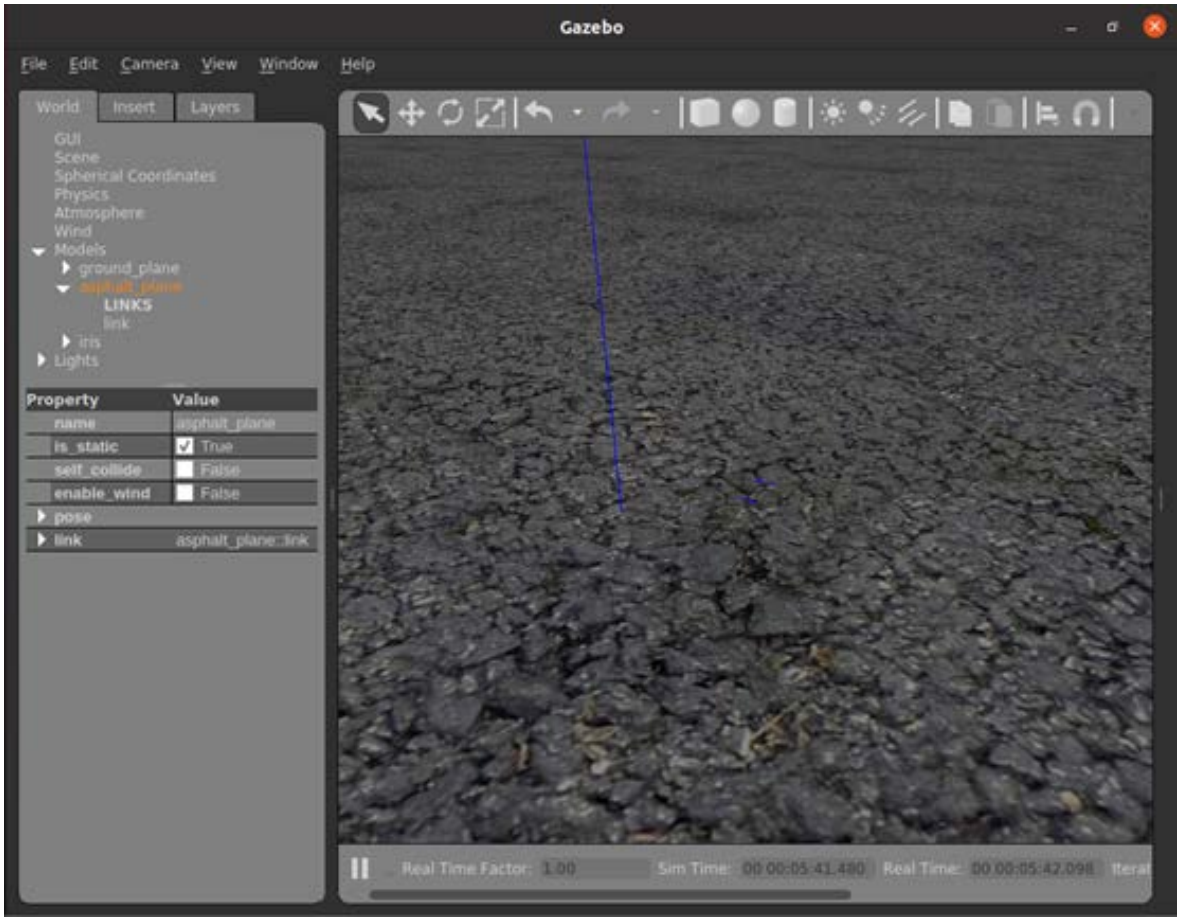

Figure 4.5: Gazebo simulator

Now that the simulator is launched, we can assess its functionality by executing the take-off command for the drone. This can be accomplished using the command ’commander take-off’.

```
$ Commander takeoff
```

```
pxh> commander takeoff
pxh> INFO  [commander] Armed by internal command
INFO  [navigator] Using minimum takeoff altitude: 2.50 m
INFO  [tone_alarm] arming warning
INFO  [commander] Takeoff detected
```

Figure 4.6: Takeoff command

In the output, we can observe that the sensors have successfully detected the takeoff, confirming that PX4 is operational. We can now move forward with installing the Ground Control Station (GCS) to gain greater control over the drone.

4.3 Installation of Mission Planner GCS

4.3.1 Installation Mono

To utilize Mission Planner, the installation of Mono is required since Mission Planner relies on the .NET framework, which is not default in Ubuntu. To facilitate this, we will initially add the Mono repository to our package manager for streamlined installation and updates.

To begin, we'll add the necessary certificates and repository keys to our system.

```
$ sudo apt install ca-certificates gnupg
$ sudo gpg --homedir /tmp --no-default-keyring --keyring /usr/
share/keyrings/mono-official-archive-keyring.gpg --keyserver
hkp://keyserver.ubuntu.com:80 --recv-keys3FA7E0328081BFF6A14
DA29AA6A19B38D3D831EF
```

```
admtina@ubuntu:~/PX4-Autopilot$ sudo gpg --homedir /tmp --no-default-keyring --
keyring /usr/share/keyrings/mono-official-archive-keyring.gpg --keyserver hkp:/
/keyserver.ubuntu.com:80 --recv-keys 3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF
gpg: keybox '/usr/share/keyrings/mono-official-archive-keyring.gpg' created
gpg: /tmp/trustdb.gpg: trustdb created
gpg: key A6A19B38D3D831EF: public key "Xamarin Public Jenkins (auto-signing) <r
eleng@xamarin.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

Figure 4.7: Adding certificates and repository keys

Next, we will incorporate the repository into our package manager by adding the repository link to the file where our package manager searches for repositories.

```
$ echo "deb [signed-by=/usr/share/keyrings/mono-official-archive
-keyring.gpg] https://download.mono-project.com/repo/ubuntu
stable-focal main" | sudo tee /etc/apt/sources.list.d/mono-
official-stable.list
```

```
admtina@ubuntu:~/PX4-Autopilot$ echo "deb [signed-by=/usr/share/keyrings/mono-o
fficial-archive-keyring.gpg] https://download.mono-project.com/repo/ubuntu stab
le-focal main" | sudo tee /etc/apt/sources.list.d/mono-official-stable.list
deb [signed-by=/usr/share/keyrings/mono-official-archive-keyring.gpg] https://d
ownload.mono-project.com/repo/ubuntu stable-focal main
```

Figure 4.8: Adding the repository to our package manager

Now, the final step is to update our package manager using the command 'sudo apt update'. After updating the repository, we can proceed with the installation of Mono. In this step, we are selecting the complete edition of Mono to ensure that we have the full capabilities of Mono integrated into our system.

```
$ sudo apt install mono-complete
```

4.3.2 Installation of Mission Planner

Following the successful installation of Mono, we can proceed with the installation of Mission Planner, the ground control station. To initiate this, we will begin by downloading the Mission Planner code using the 'curl' command.

```
$ Curl -o  MissionPlanner-latest.zip https://firmware.ardupilot
.org/Tools/MissionPlanner/MissionPlanner-latest.zip
```

```
admtina@ubuntu:~$ curl -o MissionPlanner-latest.zip https://firmware.ardupilot.
org/Tools/MissionPlanner/MissionPlanner-latest.zip
µ  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
t
                                 Dload  Upload   Total   Spent    Left  Speed
100  109M  100  109M    0     0  19.1M      0  0:00:05  0:00:05 --:--:-- 23.8M
```

Figure 4.9: Download mission planner code

Once the download is completed, the next step is to extract the files to a directory using the 'unzip' command.

```
$ Unzip MissionPlanner-latest.zip -d MissionPlanner
```

```
admtina@ubuntu:~$ unzip MissionPlanner-latest.zip -d MissionPlanner
Archive:  MissionPlanner-latest.zip
```

Figure 4.10: Unzip mission planner code

Testing Mission Planner

Now that everything is installed and the files are unpacked, we can test Mission Planner. To do so, navigate to the directory where Mission Planner was unpacked and execute the 'mono' command to launch Mission Planner with the .NET framework.

```
$ mono MissionPlanner.exe
```

Upon executing this command, the Mission Planner user interface (UI) will appear, confirming that the installation was successful.

Figure 4.11: Mission Planner window

Once we have confirmed that both Mission Planner and PX4 are operational, it's time to establish the connection between the two.

4.4 Connect PX4 Autopilot with Mission Planner

To conduct UAV simulations, we utilize the PX4 Software-in-the-Loop (SITL) with Gazebo as our virtual environment. As there is no physical flight controller involved, we only select the frame ”Quadrotor (Default)” without requiring firmware for the simulator. The simulation is initiated by executing the following command:

```
$ make px4_sitl gazebo-classic
```

Regarding the communication ports, we use UDP port 14540 to the upper right corner to facilitate data exchange between the PX4 autopilot and the Mission Planner ground control station. after we see the connected icon like following we are ready to plan a mission.

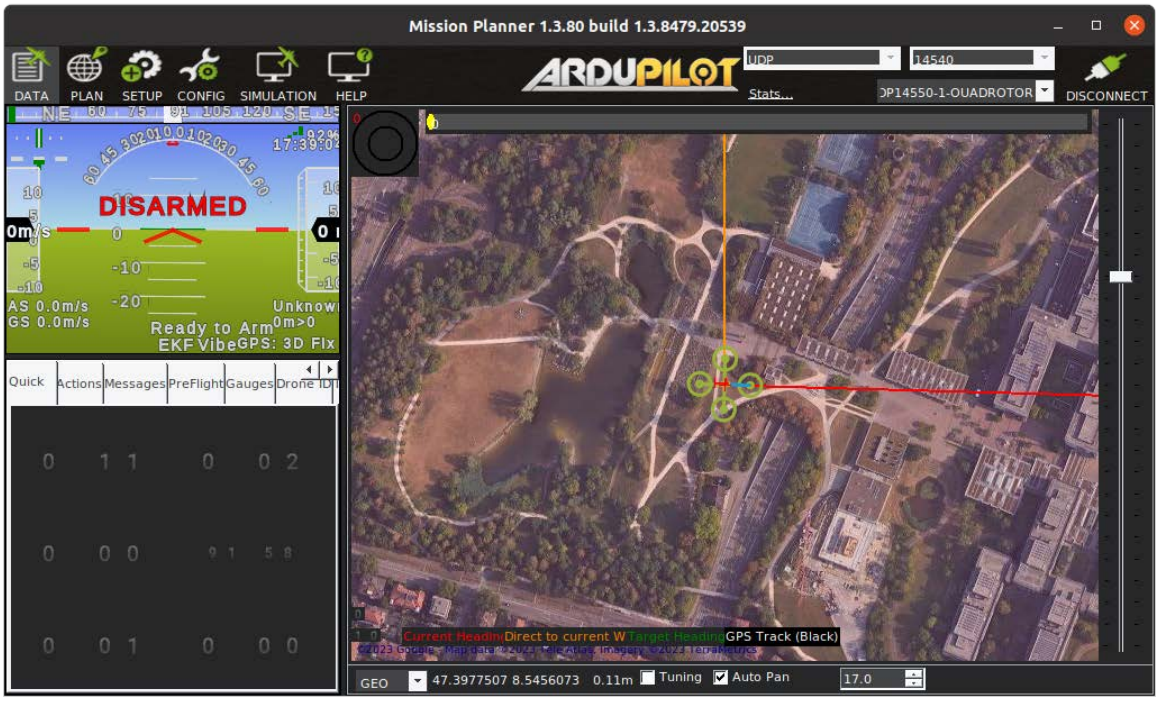

Figure 4.12: Mission planner connected

To create a mission in Mission Planner, follow these steps:

1. In the upper-left corner, select ”Plan Mission.”
2. Add the following waypoints to your mission:
    - Takeoff point (e.g., altitude of 10 meters)
    - Waypoint 1 (e.g., coordinates and altitude)

- Waypoint 2 (e.g., coordinates and altitude)
- Return to launch point

3. Once you have added these waypoints, click on ”Upload” to transfer the mission to your UAV and test it with this simple mission.

Figure 4.13: Test mission

However, we encountered an issue with mission planning due to differences in the MAVLink protocol used by the two components.

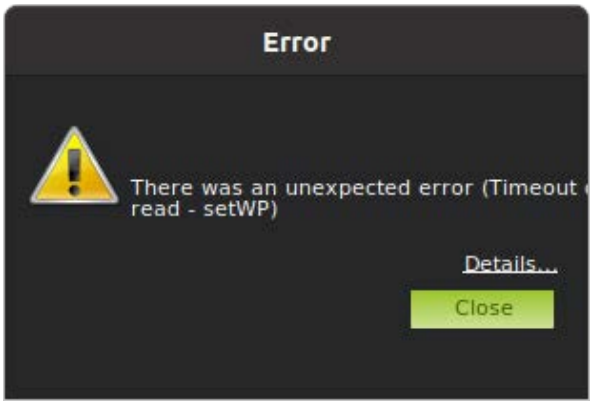

Figure 4.14: SetWP error

4.5 WireShark

Wireshark is a robust and popular network protocol analyzer that offers in-depth understanding of the communication flow over a computer network. Wireshark's graphical user interface and packet capture feature let users to record, view, and analyze network traffic either in real-time or from previously recorded files. Wireshark displays the contents of each communication, including protocols, headers, payloads, and time information, by capturing and analyzing network packets. This tool is useful for identifying network difficulties, resolving communication challenges, and confirming that network protocols are being followed to the required standards. [8]

4.5.1 Installation of WireShark

We will now install Wireshark to facilitate the examination of variances in the MAVLink protocol.

To begin, it is necessary to add the repository to our package manager. This can be achieved through the following command:

```
$ sudo add-apt-repository ppa:wireshark-dev/stable
```

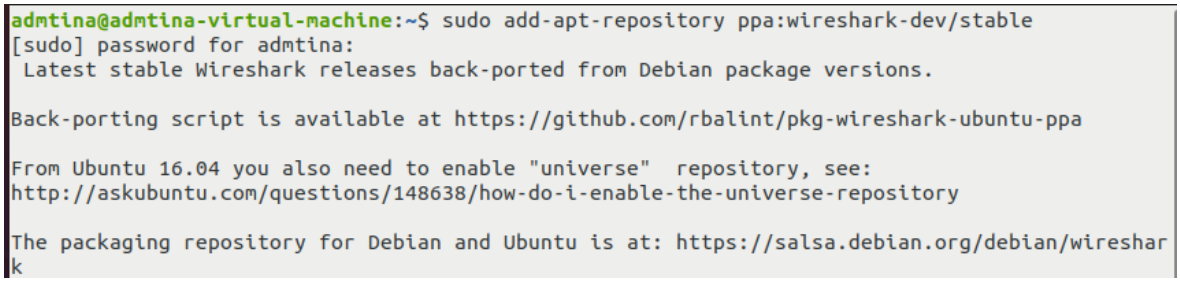

Figure 4.15: Adding the repo to our packagemanager

After successfully adding the repository, we can proceed with the installation of Wireshark by using the following command:

```
$ sudo apt install wireshark
```

```
admtina@admtina-virtual-machine:~$ sudo apt install wireshark
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure 4.16: Install WireShark

4.5.2 Installation of the plugin

Now that Wireshark has been successfully installed, the subsequent procedure involves integrating the MAVLink plugin to detect and filter MAVLink packets. This plugin can be obtained by downloading it from its dedicated GitHub repository.

https://gist.github.com/dagar/c006ce6014fd56fbdab92af062bd8e19

After obtaining the plugin, it is necessary to relocate it to the designated directory. This relocation can be accomplished through the ’cp’ (copy) command.

```
$ sudo cp mavlink_common.lua /usr/lib/x86_64-linux-gnu/
wireshark/plugins/
```

```
admtina@admtina-virtual-machine:~/Downloads$ sudo cp mavlink_common.lua /usr/lib/x86_64-linux-
gnu/wireshark/plugins/
[sudo] password for admtina:
```

Figure 4.17: Move plugin

Now that the plugin has been correctly placed in the designated directory, Wireshark can be initiated using the following command:

```
$ sudo apt install wireshark
```

To verify if the plugin has been correctly installed, you can follow these steps:

1. Launch Wireshark.
2. Navigate to the ”Help” menu.
3. Select ”About Wireshark.”
4. In the ”About Wireshark” window, locate and click on the ”Plugins” tab.
5. You should see the MAVLink plugin listed among the installed plugins, confirming its successful installation.

This will confirm whether the plugin has been correctly installed and is available for use in Wireshark.

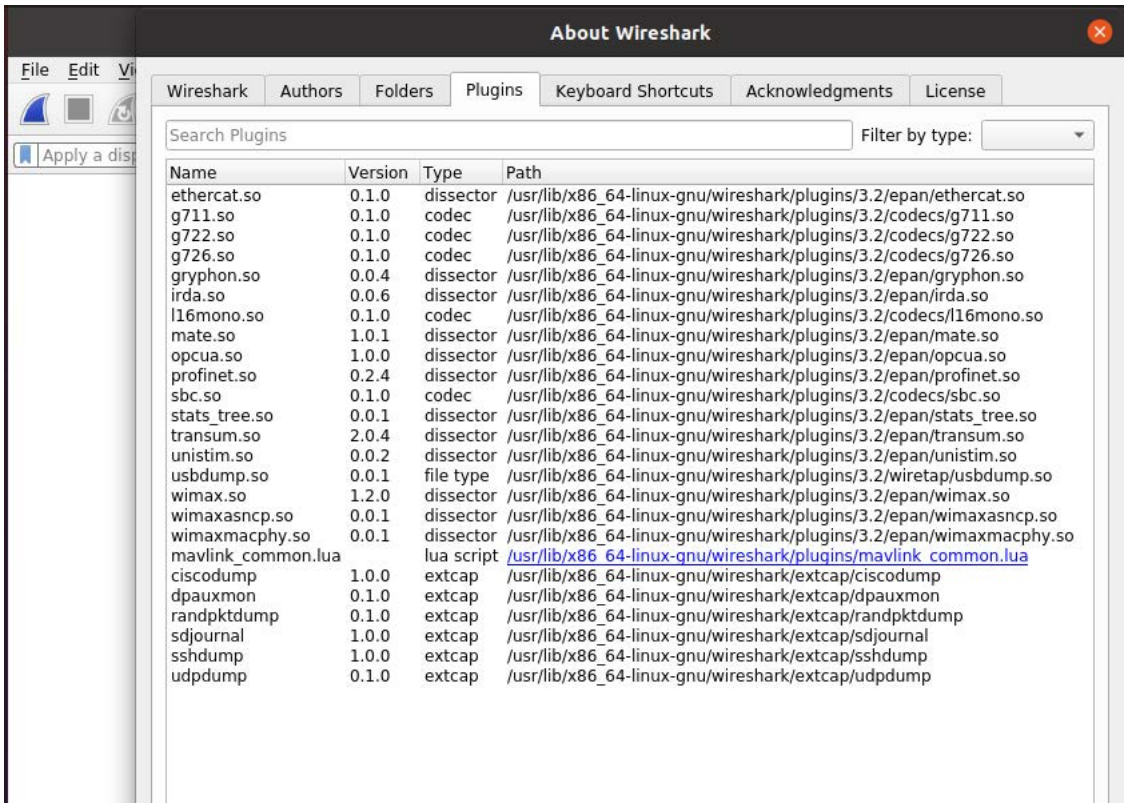

Figure 4.18: WireShark plugin

4.6 Installation of ArduPilot Autopilot

Ardupilot is an open-source autopilot software suite for unmanned aerial vehicles (UAVs) and other autonomous vehicles that are supported by Mission Planner and we are going to use it for testing. [9]

4.6.1 Installation of Gazebo

Initially, it is imperative to incorporate Gazebo into our package management system to facilitate seamless installations and future updates. To commence this process, we shall add the repository to our system by executing the following command:

```
$ sudo wget https://packages.osrfoundation.org/gazebo.gpg -O
/usr/share/keyrings/pkgs-osrf-archive-keyring.gpg
```

```
admtina@admtina-virtual-machine:~$ sudo wget https://packages.osrfoundation.org/
gazebo.gpg -O /usr/share/keyrings/pkgs-osrf-archive-keyring.gpg
--2023-08-06 16:06:33--  https://packages.osrfoundation.org/gazebo.gpg
Resolving packages.osrfoundation.org (packages.osrfoundation.org)...
52.52.171.73
Connecting to packages.osrfoundation.org (packages.osrfoundation.org)|52.52.171.
73|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1222 (1,2K) [application/octet-stream]
Saving to: '/usr/share/keyrings/pkgs-osrf-archive-keyring.gpg'

/usr/share/keyrings 100%[===================>]   1,19K  --.-KB/s    in 0s

2023-08-06 16:06:58 (318 MB/s) - '/usr/share/keyrings/pkgs-osrf-archive-keyring.
gpg' saved [1222/1222]
```

Figure 4.19: Adding the Gazebo repo

After adding the repository to our system, it is necessary to include the repository key using the following command:

```
$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr
/share/keyrings/pkgs-osrf-archive-keyring.gpg] http://packages
.osrfoundation.org/gazebo/ubuntu-stable $(lsb_release -cs)
main" | sudo tee /etc/apt/sources.list.d/gazebo-stable.list >
/dev/null
```

```
admtina@admtina-virtual-machine:~$ echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/pkgs-osrf-archive-keyring.gpg] http://packages.osr
foundation.org/gazebo/ubuntu-stable $(lsb_release -cs) main" | sudo tee /etc/apt
/sources.list.d/gazebo-stable.list > /dev/null
```

Figure 4.20: Adding repo key

Subsequently, to finalize the addition of the repository, it is essential to update our package manager. This can be achieved by executing the following command:

```
$ sudo apt-get update
```

Now, Gazebo can be installed using the following command:

```
$ sudo apt-get install gz-garden
```

```
admtina@admtina-virtual-machine:~$ sudo apt-get install gz-garden
[sudo] password for admtina:
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure 4.21: Install Gazebo

You can assess the Gazebo installation by executing the following command:

`$ gz sim -v4 -r shapes.sdf This will launch the gazebo simulator with a blank canvas.`

```
admtina@admtina-virtual-machine:~$ gz sim -v4 -r shapes.sdf
[Msg] Gazebo Sim GUI    v7.5.0
[Msg] Copied installed config [/usr/share/gz/gz-sim7/gui/gui.config] to default
config [/home/admtina/.gz/sim/7/gui.config].
[Dbg] [gz.cc:163] Subscribing to [/gazebo/starting_world].
[Dbg] [gz.cc:165] Waiting for a world to be set from the GUI...
[Dbg] [Gui.cc:260] Waiting for subscribers to [/gazebo/starting_world]...
[Msg] Received world [shapes.sdf] from the GUI.
```

Figure 4.22: Testing Gazebo

4.6.2 Installation of ArduPilot plugin

Before installing the plugin, it's necessary to install some dependencies, namely 'ibgz-sim7-dev' and 'rapidjson-dev'.

`$ sudo apt install libgz-sim7-dev rapidjson-dev`

```
admtina@admtina-virtual-machine:~$ sudo apt install libgz-sim7-dev rapidjson-dev
[sudo] password for admtina:
Reading package lists... Done
Building dependency tree
Reading state information... Done
libgz-sim7-dev is already the newest version (7.5.0-1~focal).
libgz-sim7-dev set to manually installed.
```

Figure 4.23: Install dependencies

Now, it is imperative to establish a dedicated directory for our plugin and navigate into this directory.

`$ mkdir -gz_ws/src && cd gz_ws/src`

```
admtina@admtina-virtual-machine:~$ mkdir gz_ws/src && cd gz_ws/src
```

Figure 4.24: Create directory

After creating the directory, we can proceed to download the plugin repository using Git.

```
$ git clone https://github.com/ArduPilot/ardupilot_gazebo
```

```
admtina@admtina-virtual-machine:~$ git clone https://github.com/ArduPilot/ardupi
lot_gazebo
Cloning into 'ardupilot_gazebo'...
```

Figure 4.25: Download repo

Next, it's necessary to establish a directory for building the plugin on your system. Create a directory named 'build' within the 'ardupilot_gazebo' directory.
Inside this 'build' directory, utilize the 'cmake' command to initiate the build process for the plugin.

```
$ cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

```
admtina@admtina-virtual-machine:~/gz_ws/src/ardupilot_gazebo/build$ cmake .. -DCMAKE_BUILD_TYPE=RelWithD
ebInfo
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
```

Figure 4.26: Build plugin

To complete the building process of your plugin, execute the following command:

```
$ make -j4
```

```
admtina@admtina-virtual-machine:~/gz_ws/src/ardupilot_gazebo/build$ make -j4
Scanning dependencies of target ArduPilotPlugin
Scanning dependencies of target ParachutePlugin
[ 16%] Building CXX object CMakeFiles/ArduPilotPlugin.dir/src/ArduPilotPlugin.cc.o
[ 33%] Building CXX object CMakeFiles/ArduPilotPlugin.dir/src/Util.cc.o
[ 50%] Building CXX object CMakeFiles/ParachutePlugin.dir/src/ParachutePlugin.cc.o
[ 66%] Building CXX object CMakeFiles/ArduPilotPlugin.dir/src/Socket.cpp.o
[ 83%] Linking CXX shared library libParachutePlugin.so
[ 83%] Built target ParachutePlugin
[100%] Linking CXX shared library libArduPilotPlugin.so
[100%] Built target ArduPilotPlugin
```

Figure 4.27: Run -j4

4.6.3 Installation of SITL

To initiate the installation of ArduPilot, we must first download it using the Git version control system.

```
$ git clone --recurse-submodules https://github.com/ArduPilot/
ardupilot.git
```

```
admtina@admtina-virtual-machine:~$ git clone --recurse-submodules https://github.com/ArduPilot/ardupilot
.git
Cloning into 'ardupilot'...
```

Figure 4.28: Download ArduPilot

After successfully cloning the repository, navigate into the directory and execute the dependencies script. This script will handle the installation of all necessary dependencies.

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

```
admtina@admtina-virtual-machine:~/ardupilot$ Tools/environment_install/install-prereqs-ubuntu.sh -y
---------- Tools/environment_install/install-prereqs-ubuntu.sh start ----------
```

Figure 4.29: Run dependencies script

With all the installations completed, the next step involves configuring the Gazebo environment. This can be accomplished by exporting the necessary paths to your system's environment variables.

```
$ export GZ_SIM_SYSTEM_PLUGIN_PATH=$HOME/gz_ws/src/ardupilot
_gazebo/build:$GZ_SIM_SYSTEM_PLUGIN_PATH
$ export GZ_SIM_RESOURCE_PATH=$HOME/gz_ws/src/ardupilot_gazebo
/models:$HOME/gz_ws/src/ardupilot_gazebo/worlds:$GZ_SIM_
RESOURCE_PATH
```

With all the necessary components installed, you can now proceed to run ArduPilot with the Gazebo simulator. To begin, start Gazebo.

```
$ gz sim -v4 -r iris_runway.sdf
```

```
admtina@admtina-virtual-machine:~/gz_ws/src/ardupilot_gazebo$ gz sim -v4 -r iris
_runway.sdf
[Msg] Gazebo Sim GUI    v7.5.0
[Dbg] [gz.cc:163] Subscribing to [/gazebo/starting_world].
[Dbg] [gz.cc:165] Waiting for a world to be set from the GUI...
[Dbg] [Gui.cc:260] Waiting for subscribers to [/gazebo/starting_world]...
[Msg] Received world [iris_runway.sdf] from the GUI.
[Dbg] [gz.cc:169] Unsubscribing from [/gazebo/starting_world].
[Msg] Gazebo Sim Server v7.5.0
[Msg] Loading SDF world file[/home/admtina/gz_ws/src/ardupilot_gazebo/worlds/iri
s_runway.sdf].
```

Figure 4.30: Start Gazebo

After Gazebo is running, you can initiate Software-in-the-Loop (SITL) with ArduPilot using the following command:

```
$ sim_vehicle.py -v ArduCopter -f gazebo-iris --model JSON --map
--console
```

```
admtina@admtina-virtual-machine:~/gz_ws/src/ardupilot_gazebo$ sim_vehicle.py -v
ArduCopter -f gazebo-iris --model JSON --map --console
SIM_VEHICLE: Start
SIM_VEHICLE: Killing tasks
SIM_VEHICLE: Starting up at SITL location
SIM_VEHICLE: WAF build
SIM_VEHICLE: Configure waf
SIM_VEHICLE: "/home/admtina/ardupilot/modules/waf/waf-light" "configure" "--boar
d" "sitl"
```

Figure 4.31: Start SITL

Subsequently, you can launch Mission Planner and create a mission for your UAV.

4.7 Installation of QGS

QGroundControl (QGC) is a ground control station compatible with PX4 AutoPilot. [10] To install QGC, you should execute the following commands. These commands will install the required dependencies and configure user permissions as necessary.

```
$ sudo usermod -a -G dialout $USER
$ sudo apt-get remove modemmanager -y
```

```
$ sudo apt install gstreamer1.0-plugins-bad gstreamer1.0-libav gstreame
$ sudo apt install libqt5gui5 -y
$ sudo apt install libfuse2 -y
```

After executing these commands, it's essential to log out and log in again for the changes to take effect.

Once you've completed this step, you can proceed to download QGroundControl from the provided link.

https://d176tv9ibo4jno.cloudfront.net/latest/QGroundControl.AppImage

After downloading the QGroundControl application, it is imperative to configure the file permissions to grant execution rights. This can be accomplished through the following command:

```
$ chmod +x ./QGroundControl.AppImage
```

Figure 4.32: Configure the rights

Following this configuration, you can proceed by either double-clicking the file to launch QGroundControl or running it from the terminal, depending on your preference.

```
$ ./QGroundControl.AppImage
```

Figure 4.33: Start QGC

Chapter 5

Methodology and Testing Process

5.1 Establishing MAVLink Protocol Compatibility for Mission Planning

In response to the challenge of incompatible MAVLink protocols between the PX4 autopilot and the Mission Planner, we are proactively addressing this issue by working towards establishing compatibility. Our ongoing efforts are centered on locationg the differences of the MAVLink protocol of the Mission Planner and modifying it to align with the original MAVLink protocol used by the PX4 autopilot. By making these adjustments, we aim to enable seamless mission planning functionality while ensuring continuous availability of sensor data. By achieving compatibility and resolving the communication issue, we aim to create a robust and efficient UAV simulation environment. Our goal is to enable mission planning capabilities while benefiting from sensor data integration, thereby enhancing the overall effectiveness of the UAV simulation setup.

5.2 Using WireShark to locate the differences to the MAVLink Protocol

By capturing and inspecting network packets exchanged between the ground control station (GCS) and the UAV's autopilot system, WireShark enables the detailed examination of the communication process. WireShark's ability to filter and dissect MAVLink messages aids in pinpointing potential inconsistencies and variations between the expected protocol speci-

fications and the actual implementation. This proactive approach allows resolving compatibility issues, ensuring a smooth and accurate MAVLink communication in UAV operations.

Using WireShark as a network protocol analyzer, we plan to focus specifically on the mission upload process as the test case for locating differences in the MAVLink Protocol. By capturing and analyzing the network packets exchanged during mission upload between the ground control station (GCS) and the UAV's autopilot system, WireShark will enable us to delve deep into the communication flow. This focused examination will help us identify any variations, inconsistencies, or non-compliant messages that may arise during the mission upload process. This approach will enable us to make informed adjustments, resolve any compatibility issues, and fine-tune the protocol implementation to achieve seamless and reliable mission upload functionality in the UAV system.

For the test case, we have designed a mission with a sequence of waypoints to demonstrate the MAVLink Protocol functionality. The mission consists of a take-off phase, followed by navigation through two waypoints, and finally returning to the launch point.

5.2.1 MAVLink Messaging between Mission Planner and PX4 AutoPilot

To conduct the steps outlined in section 4.4, ”Connect PX4 Autopilot with Mission Planner,” it is essential to launch Wireshark before clicking ”write.” You can initiate Wireshark using the following command:

```
$ sudo wireshark
```

After launching Wireshark, double-click on ”loopback” to select it as the interface for capturing network traffic. Due to the project being in a test environment, all traffic sent and received will stay inside the host. Therefore we are using the ”loopback” interface also known as ”localhost”(127.0.0.1).

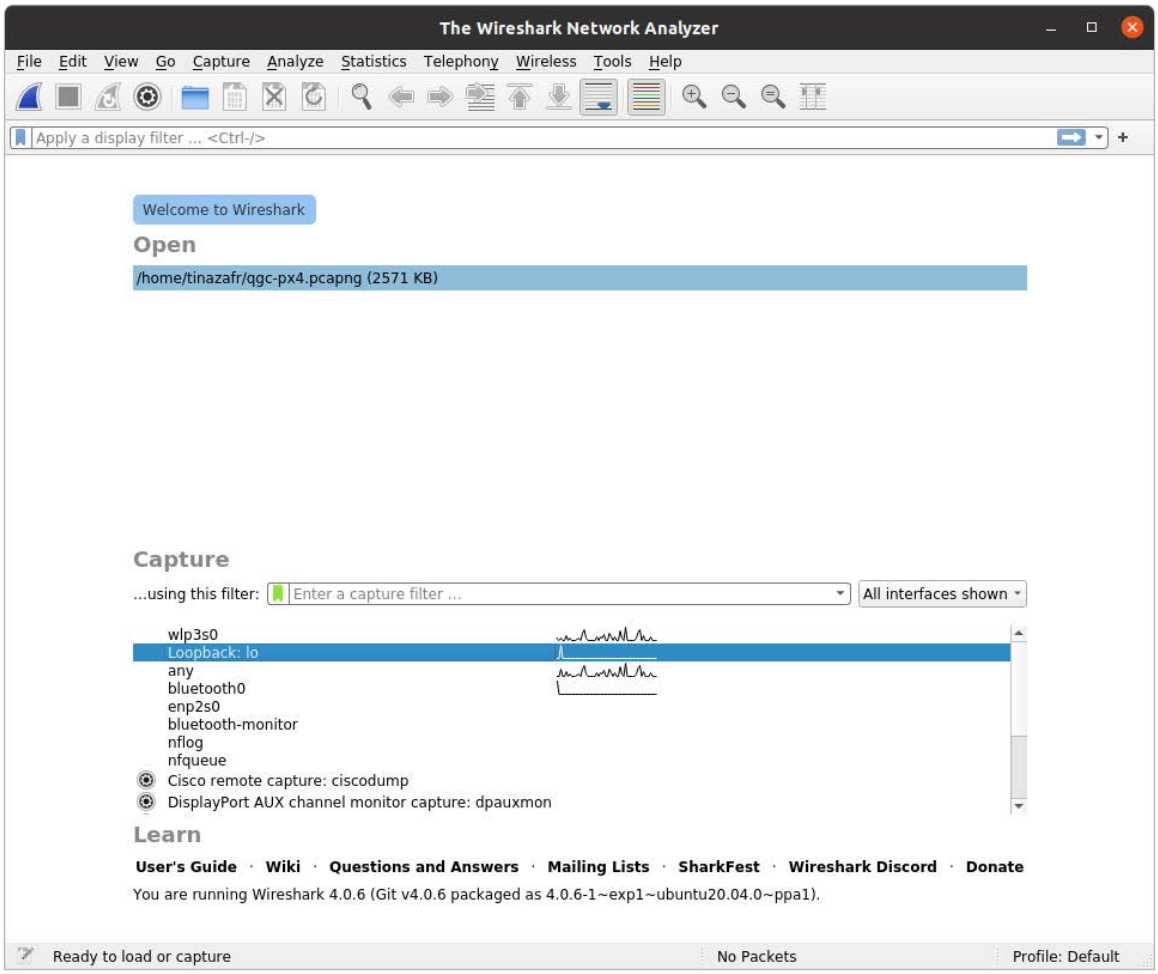

Figure 5.1: Launch WireShank

Once WireShark is actively capturing data, proceed to click ”write.”

After the error message appears, stop the data capture by clicking on the red square located in the upper-left corner of WireShark. Then, filter the captured data for MAVLink packets to locate the specific messages related to mission upload.

More specifically MISSION_COUNT, MISSION_REQUEST and MISSION_ITEM_INT must be located and the information of the message is in the Payload sector.

MISSION_COUNT

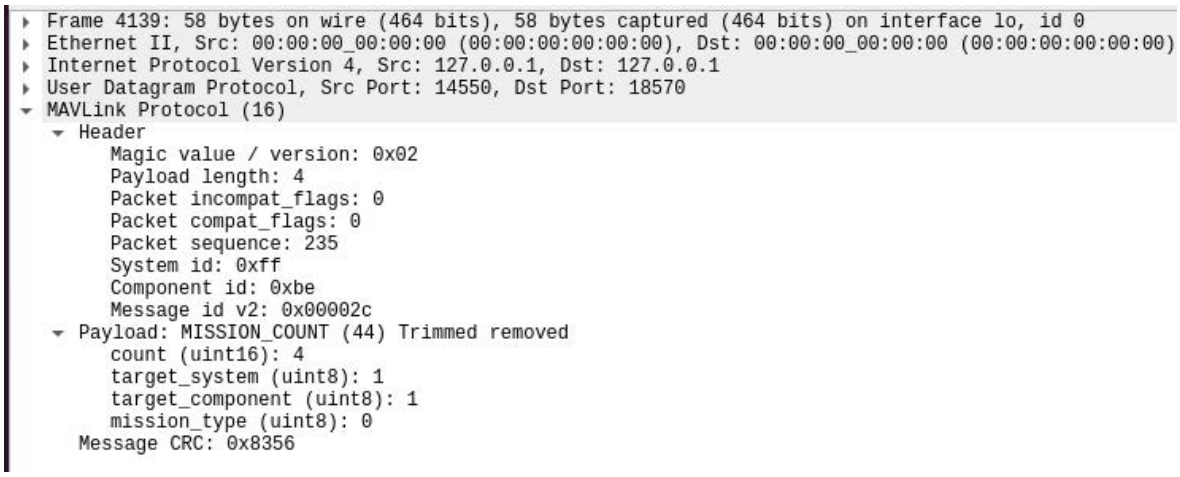

Figure 5.2: MISSION_COUNT for Mission Planner-PX4

The following information must be kept and compared:

Payload: MISSION_COUNT (44) Trimmed removed

count (uint16): 4

target_system (uint8): 1

target_component (uint8): 1

mission_type (uint8): 0

Message CRC: 0x8356

MISSION_REQUEST(0)

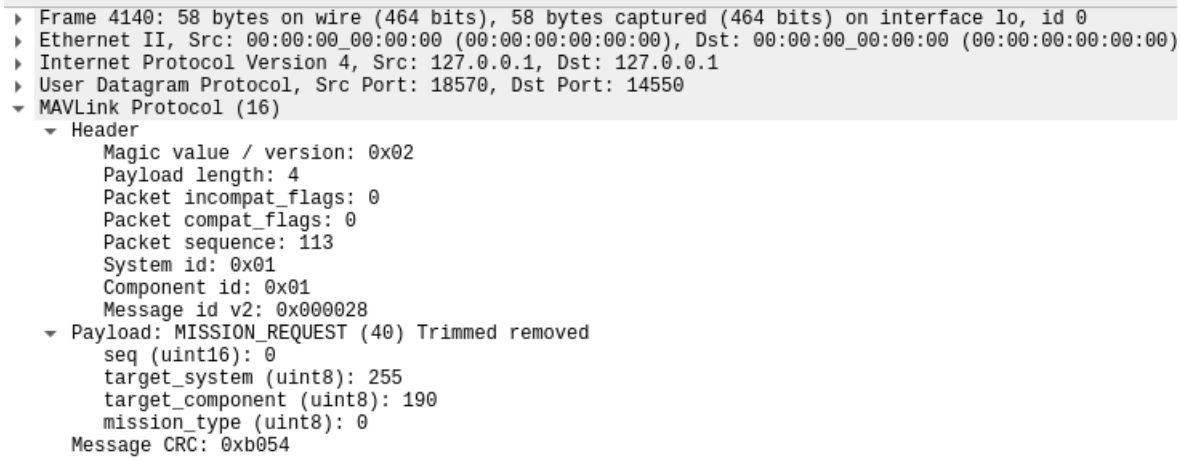

Figure 5.3: MISSION_REQUEST(0) for Mission Planner-PX4

The following information must be kept and compared:

Payload: MISSION_REQUEST (40) Trimmed removed

seq (uint16): 0

target_system (uint8): 255

target_component (uint8): 190

mission_type (uint8): 0

Message CRC: 0xb054

MISSION_ITEM_INT(0)

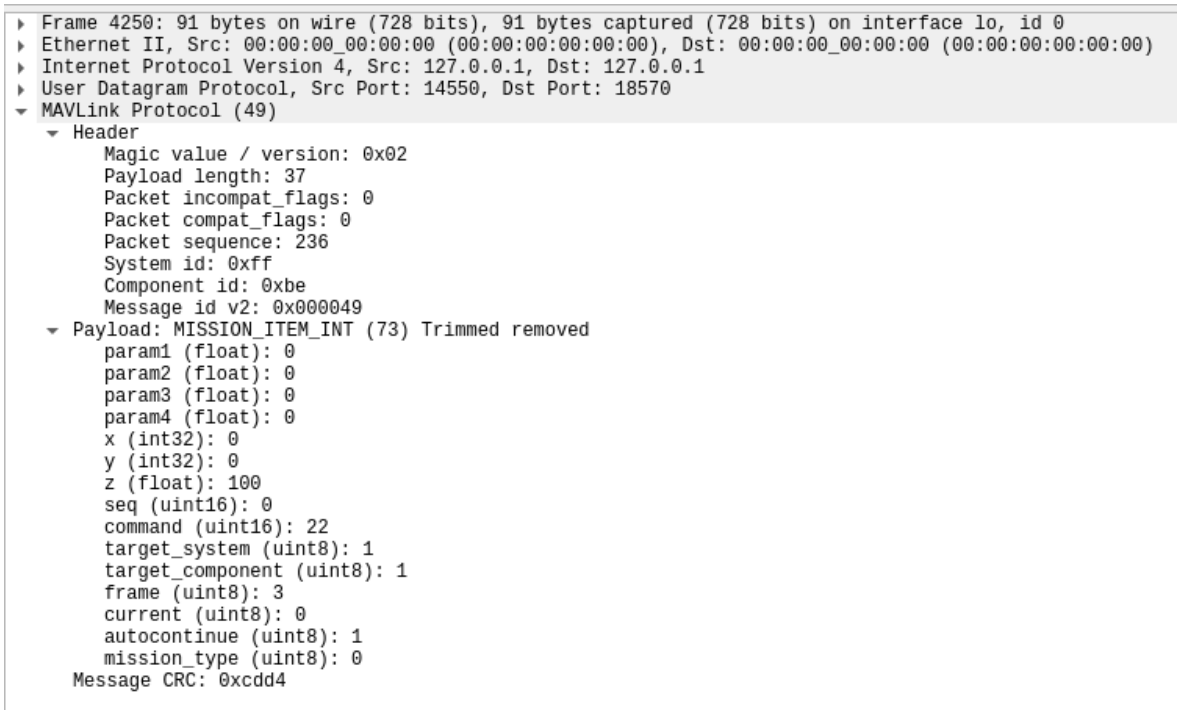

Figure 5.4: MISSION_ITEM_INT(0) for Mission Planner-PX4

The following information must be kept and compared:

Payload: MISSION_ITEM_INT (73) Trimmed removed

param1 (float): 0

param2 (float): 0

param3 (float): 0

param4 (float): 0

x (int32): 0

y (int32): 0

z (float): 100

seq (uint16): 0

command (uint16): 22

target_system (uint8): 1

target_component (uint8): 1

frame (uint8): 3

current (uint8): 0

autocontinue (uint8): 1

mission_type (uint8): 0

Message CRC: 0xcdd4

MISSION_REQUEST(1)

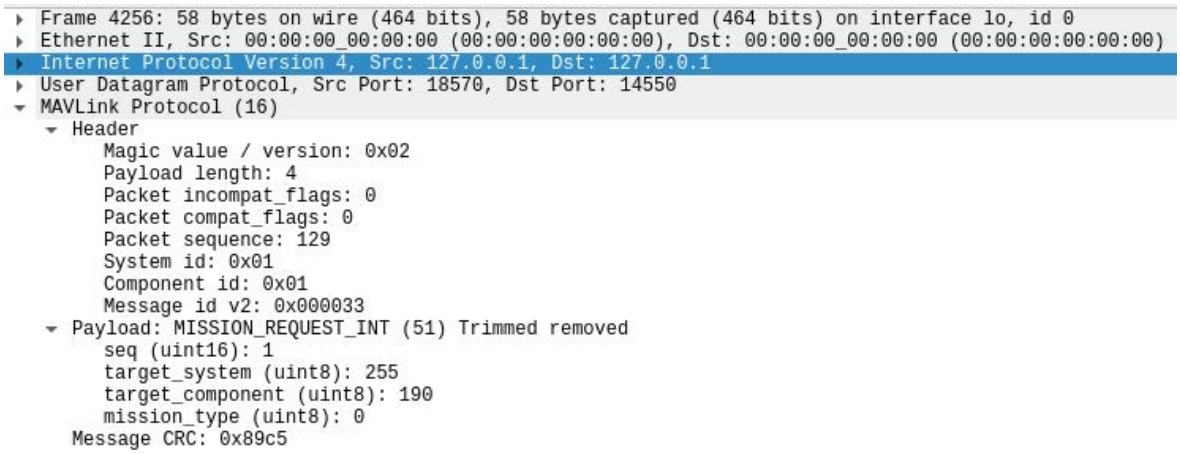

Figure 5.5: MISSION_REQUEST(1) for Mission Planner-PX4

The following information must be kept and compared:

Payload: MISSION_REQUEST (51) Trimmed removed

seq (uint16): 1

target_system (uint8): 255

target_component (uint8): 190

mission_type (uint8): 0

Message CRC: 0x89c5

Following the transmission of the MISSION_REQUEST message with seq=1 from the AutoPilot to the Ground Control Station (GCS), the AutoPilot transitions into a waiting state, expecting the subsequent MISSION_ITEM message that should encompass the details of the second item within the mission. Regrettably, the GCS does not transmit the anticipated MISSION_ITEM message, consequently culminating in a disruption of the process and ultimately

resulting in an unsuccessful mission upload.

5.2.2 MAVLink Messaging between Mission Planner and compatible AutoPilot

In this section, we provide an overview of the expected sequence of messages during the mission upload process to demonstrate the correct communication flow between the autopilot and the ground control station (GCS). The following messages are crucial for reference:

MISSION_COUNT

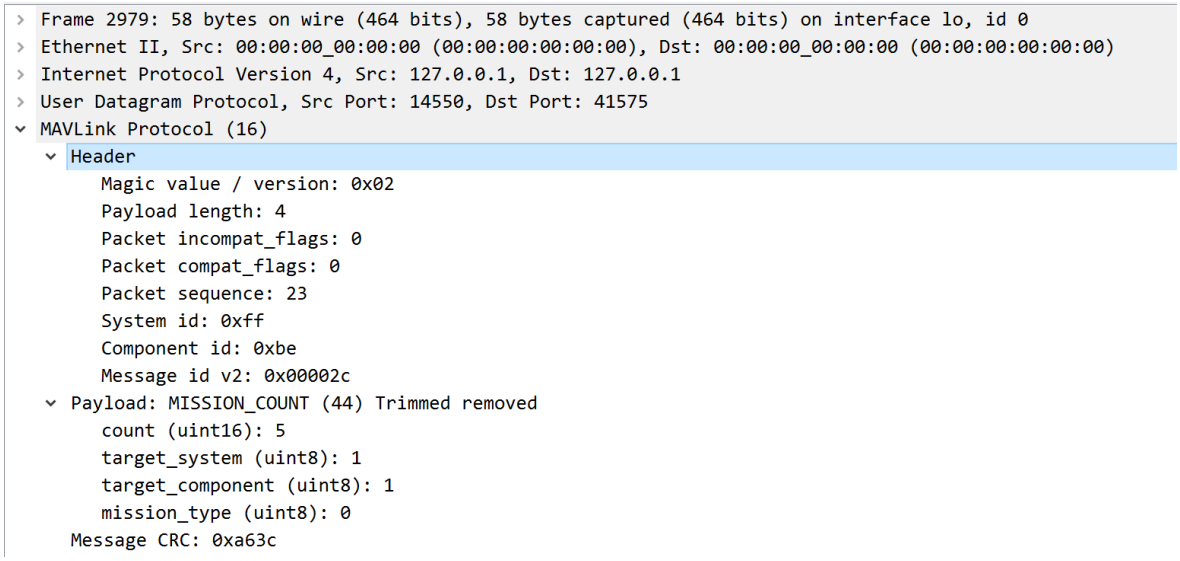

Figure 5.6: MISSION_COUNT for MissionPlanner-ArduPilot

The following information must be kept and compared:

Payload: MISSION_COUNT (44) Trimmed removed

count (uint16): 5

target_system (uint8): 1

target_component (uint8): 1

mission_type (uint8): 0

Message CRC: 0xa63c

MISSION_REQUEST(0)

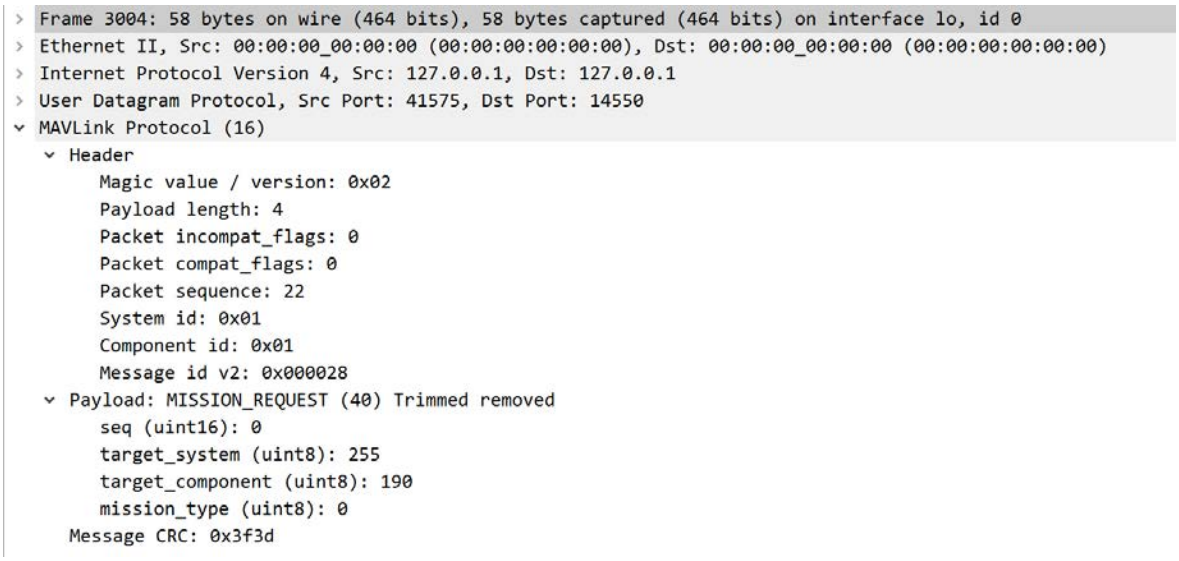

Figure 5.7: MISSION_REQUEST(0) for MissionPlanner-ArduPilot

The following information must be kept and compared:

Payload: MISSION_REQUEST (40) Trimmed removed

seq (uint16): 0

target_system (uint8): 255

target_component (uint8): 190

mission_type (uint8): 0

Message CRC: 0x3f3d

MISSION_ITEM_INT(0)

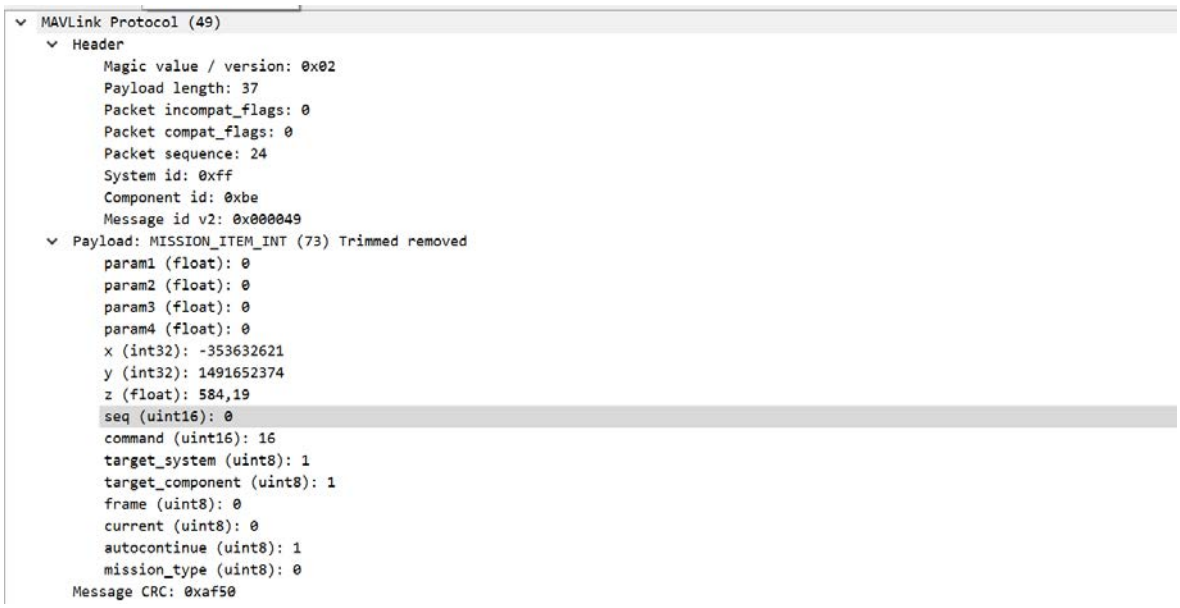

Figure 5.8: MISSION_ITEM_INT(0) for MissionPlanner-ArduPilot

The following information must be kept and compared:

Payload: MISSION_ITEM_INT (73) Trimmed removed

param1 (float): 0

param2 (float): 0

param3 (float): 0

param4 (float): 0

x (int32): -353632621

y (int32): 1491652374

z (float): 584,19

seq (uint16): 0

command (uint16): 16

target_system (uint8): 1

target_component (uint8): 1

frame (uint8): 0

current (uint8): 0

autocontinue (uint8): 1

mission_type (uint8): 0

Message CRC: 0xaf50

MISSION_REQUEST(1)

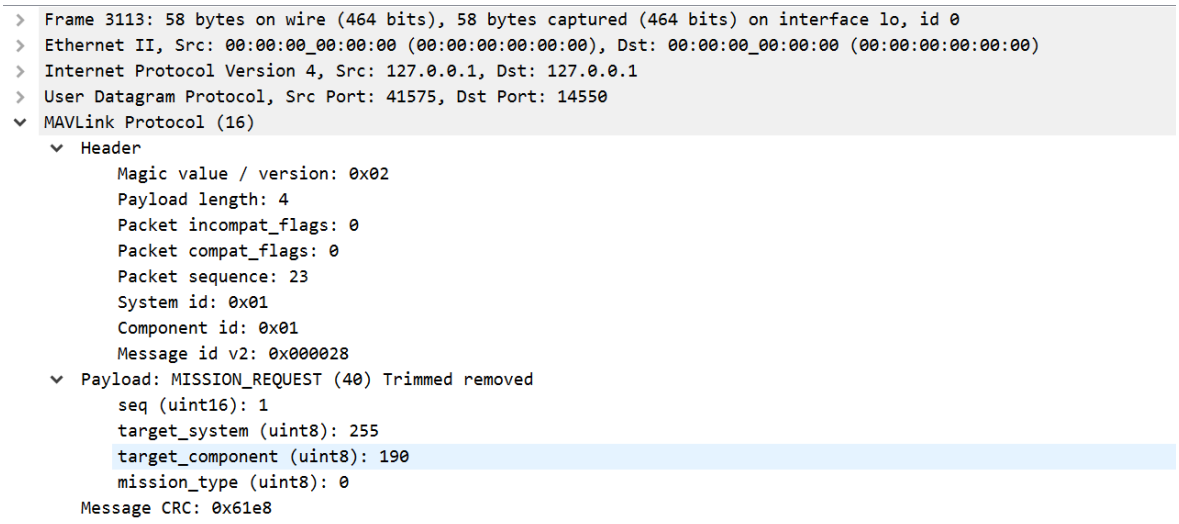

Figure 5.9: MISSION_REQUEST(1) for MissionPlanner-ArduPilot

The following information must be kept and compared:

Payload: MISSION_REQUEST (40) Trimmed removed

seq (uint16): 1

target_system (uint8): 255

target_component (uint8): 190

mission_type (uint8): 0

Message CRC: 0x61e8

MISSION_ITEM_INT(1)

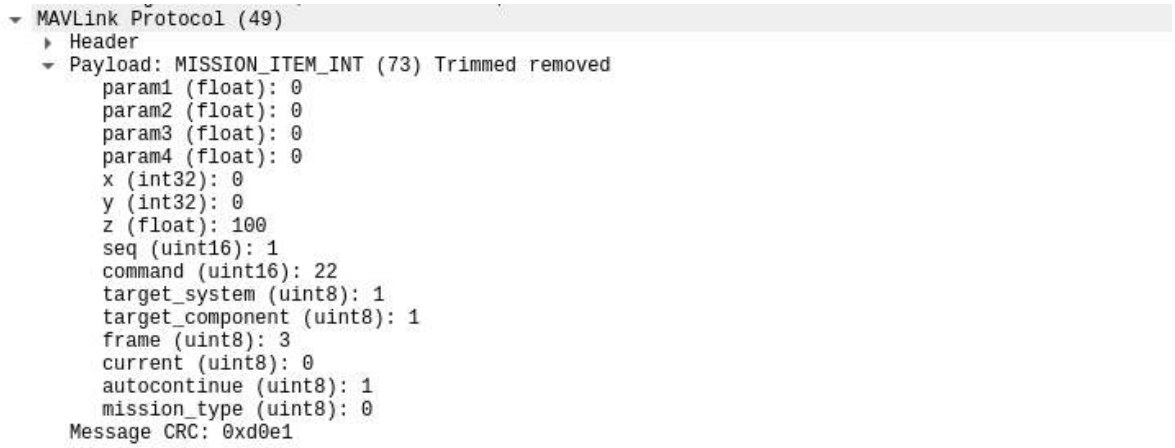

Figure 5.10: MISSION_ITEM_INT(1) for MissionPlanner-ArduPilot

The following information must be kept and compared:

Payload: MISSION_ITEM_INT (73) Trimmed removed

param1 (float): 0

param2 (float): 0

param3 (float): 0

param4 (float): 0

x (int32): 0

y (int32): 0

z (float): 100

seq (uint16): 1

command (uint16): 22

target_system (uint8): 1

target_component (uint8): 1

frame (uint8): 3

current (uint8): 0

autocontinue (uint8): 1

mission_type (uint8): 0

Message CRC: 0xd0e1

The communication between PX4 AutoPilot and Mission Planner fails for MISSION_ITEM_INT(1), so more data from other this case are not needed for comparison.

MISSION_ACK

```
> Frame 3305: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface lo, id 0
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 41575, Dst Port: 14550
v MAVLink Protocol (14)
  v Header
      Magic value / version: 0x02
      Payload length: 2
      Packet incompat_flags: 0
      Packet compat_flags: 0
      Packet sequence: 31
      System id: 0x01
      Component id: 0x01
      Message id v2: 0x00002f
  v Payload: MISSION_ACK (47) Trimmed removed Trimmed removed
      target_system (uint8): 255
      target_component (uint8): 190
      type (uint8): 0
      mission_type (uint8): 0
    Message CRC: 0x3d60
```

Figure 5.11: MISSION_ACK for MissionPlanner-ArduPilot

The following information must be kept and compared:

Payload: MISSION_REQUEST (47) Trimmed removed

target_system (uint8): 255

target_component (uint8): 190

type (uint8): 0

mission_type (uint8): 0

Message CRC: 0x3d60

5.2.3 MAVLink Messaging between PX4 AutoPilot and compatible GCS

In this section, we provide an overview of the expected sequence of messages during the mission upload process to demonstrate the correct communication flow between the autopilot and the ground control station (GCS). The following messages are crucial for reference:

1. MISSION_REQUEST: The autopilot requests the mission items from the GCS.
2. MISSION_ITEM: The GCS responds with the details of a specific mission item.
3. MISSION_ACK: The autopilot acknowledges the successful receipt of mission items.

To initiate the process, you can launch PX4-AutoPilot and then either double-click the file to start QGroundControl or run it from the terminal, depending on your preferred method.

```
$ ./QGroundControl.AppImage
```

Subsequently, you can proceed by clicking on the ”Plan” option.

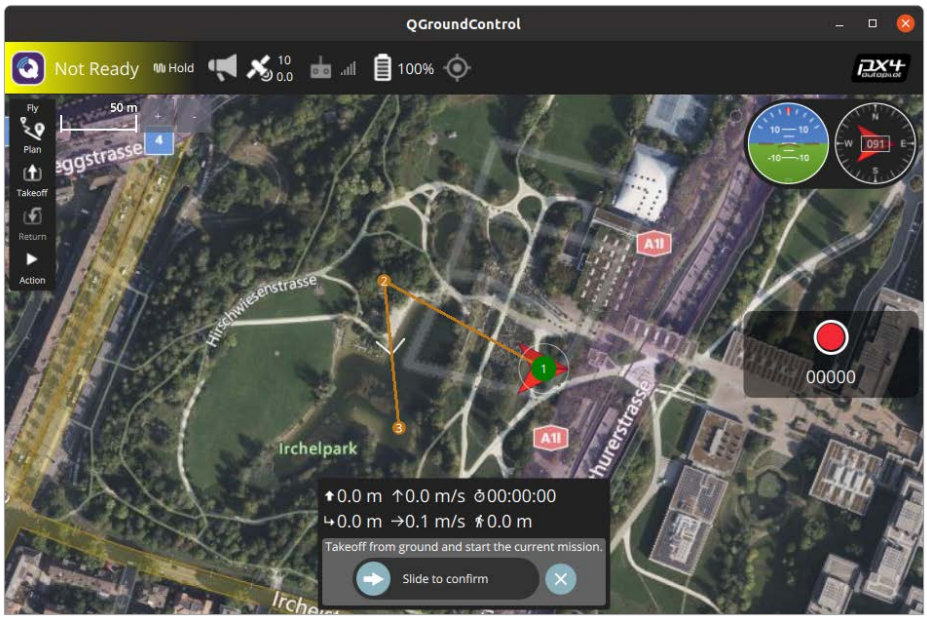

Figure 5.12: Launch QGC

Plan the test mission

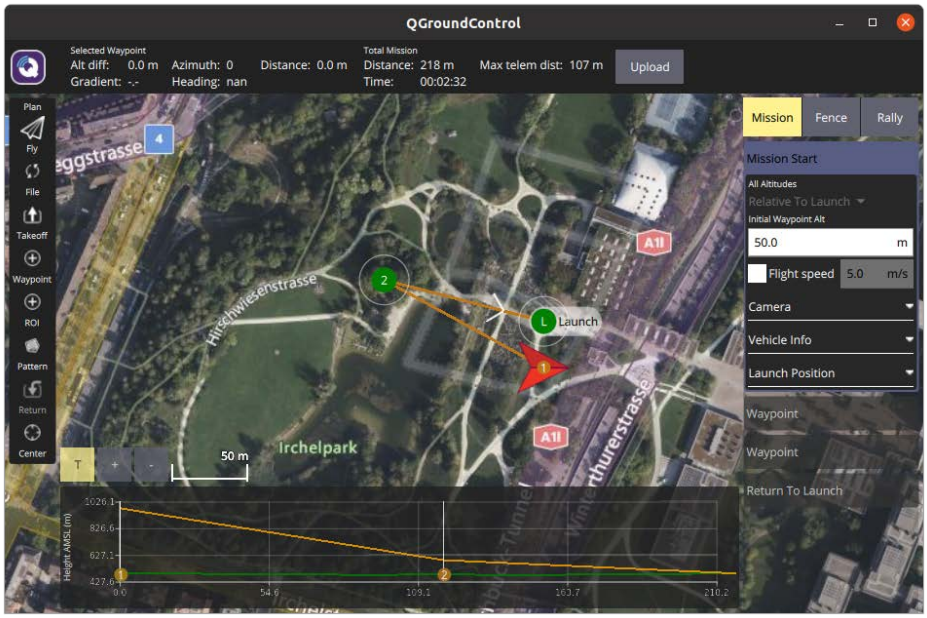

Figure 5.13: Plan mission on QGC

Following this, launch Wireshark as previously instructed and click the ”Upload” button within QGroundControl. Once the mission has been successfully uploaded, halt the data capture in Wireshark using the same method as before. Filter the MAVLink messages and identify the relevant messages as previously specified.

MISSION_COUNT

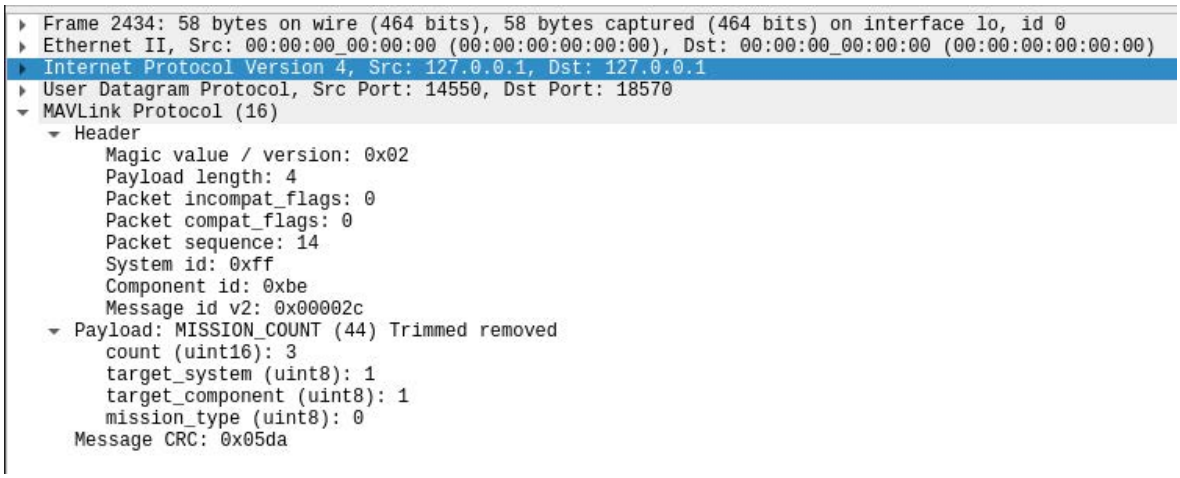

Figure 5.14: MISSION_COUNT for QGC-PX4

The following information must be kept and compared:

Payload: MISSION_COUNT (44) Trimmed removed

count (uint16): 3

target_system (uint8): 1

target_component (uint8): 1

mission_type (uint8): 0

Message CRC: 0x05da

MISSION_REQUEST(0)

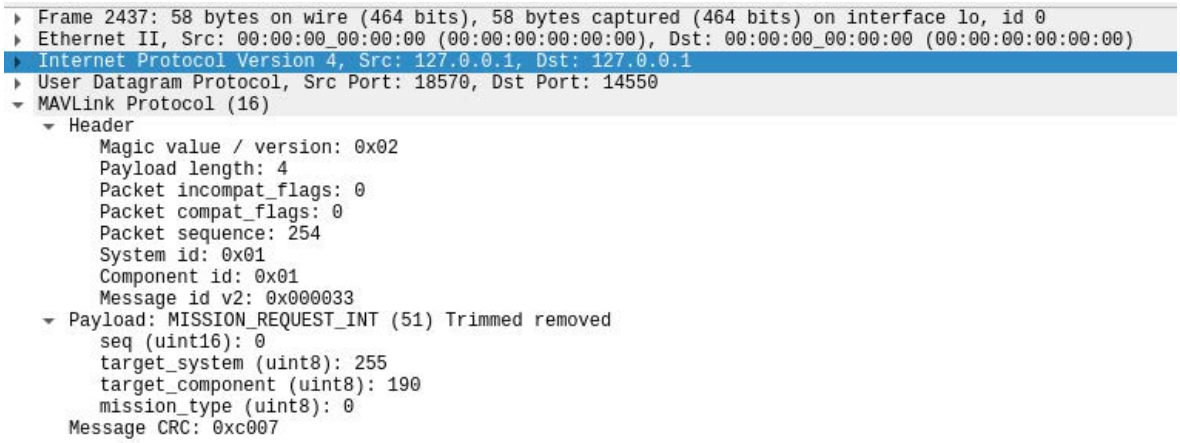

Figure 5.15: MISSION_REQUEST(0) for QGC-PX4

The following information must be kept and compared:

Payload: MISSION_REQUEST (51) Trimmed removed

seq (uint16): 0

target_system (uint8): 255

target_component (uint8): 190

mission_type (uint8): 0

Message CRC: 0xc007

MISSION_ITEM_INT(0)

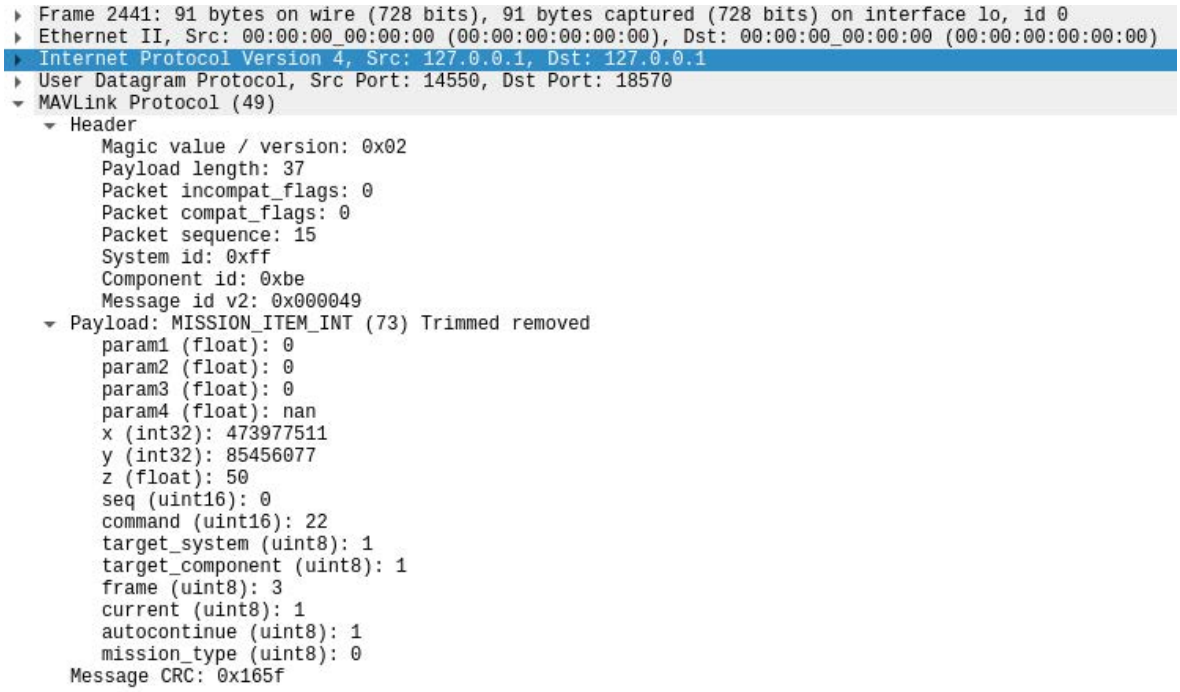

Figure 5.16: MISSION_ITEM_INT(0) for QGC-PX4

The following information must be kept and compared:

Payload: MISSION_ITEM_INT (73) Trimmed removed

param1 (float): 0

param2 (float): 0

param3 (float): 0

param4 (float): nan

x (int32): 473977511

y (int32): 85456077

z (float): 50

seq (uint16): 0

command (uint16): 22

target_system (uint8): 1

target_component (uint8): 1

frame (uint8): 3

current (uint8): 1

autocontinue (uint8): 1

mission_type (uint8): 0

Message CRC: 0x165f

MISSION_REQUEST(1)

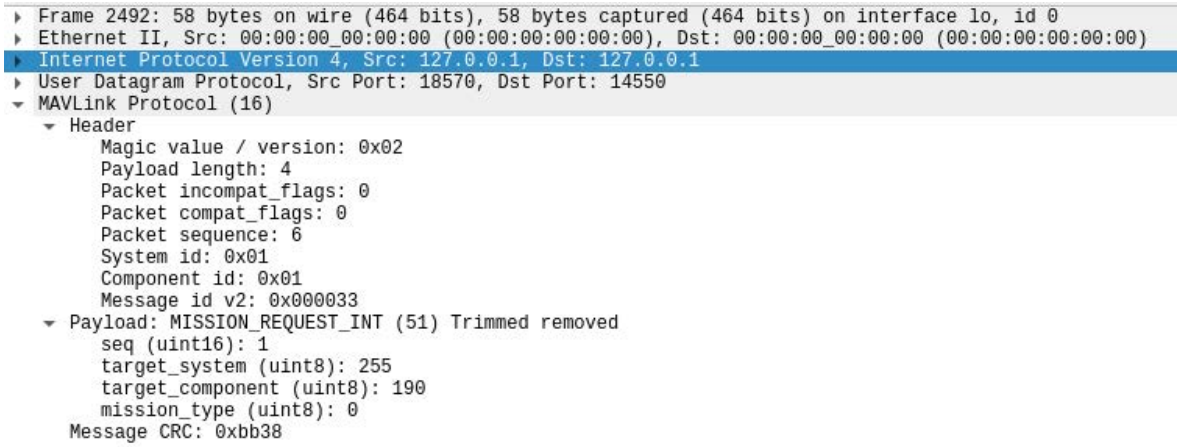

Figure 5.17: MISSION_REQUEST(1) for QGC-PX4

The following information must be kept and compared:

Payload: MISSION_REQUEST (51) Trimmed removed

seq (uint16): 1

target_system (uint8): 255

target_component (uint8): 190

mission_type (uint8): 0

Message CRC: 0xcbb38

MISSION_ITEM_INT(1)

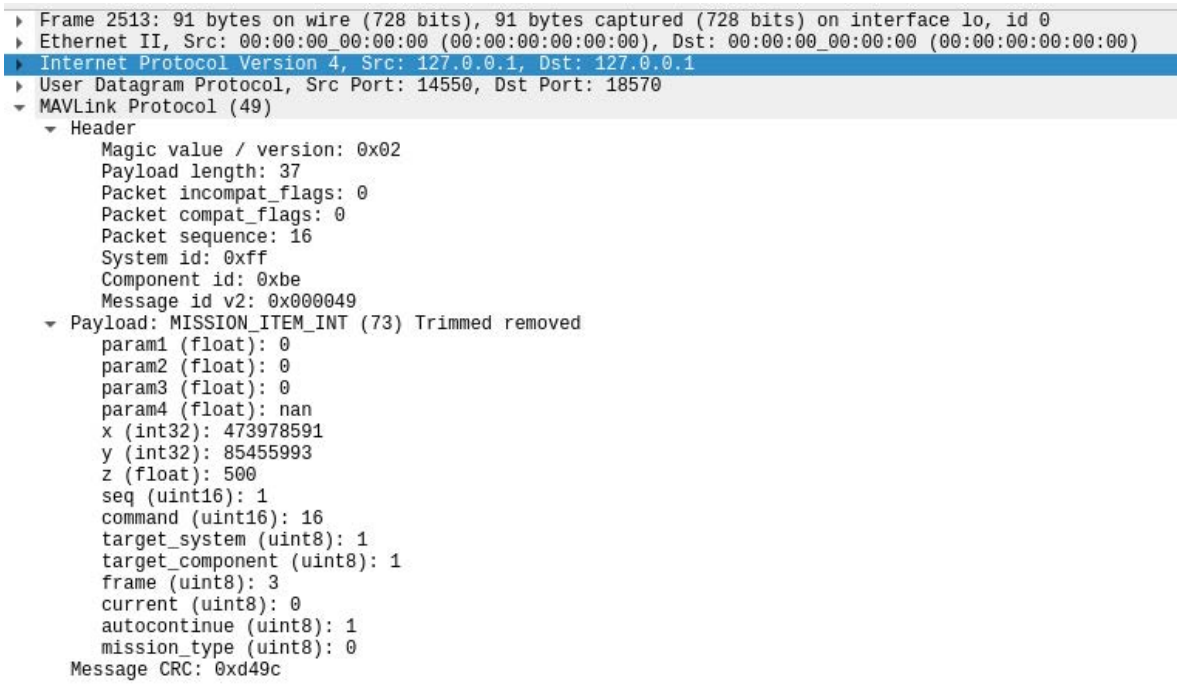

Figure 5.18: MISSION_ITEM_INT(1) for QGC-PX4

The following information must be kept and compared:

Payload: MISSION_ITEM_INT (73) Trimmed removed

param1 (float): 0

param2 (float): 0

param3 (float): 0

param4 (float): nan

x (int32): 473978591

y (int32): 85455993

z (float): 500

seq (uint16): 1

command (uint16): 16

target_system (uint8): 1

target_component (uint8): 1

frame (uint8): 3

current (uint8): 0

autocontinue (uint8): 1

mission_type (uint8): 0

Message CRC: 0xd49c

The communication between PX4 AutoPilot and Mission Planner fails for MISSION_ITEM_INT(1), so more data, from this case, are not needed for comparison.

5.3 Identification and Localization of Variations

The following tables shows the value of each variable in each case.

Table 5.1: MISSION_COUNT

	MP - PX4	MP - ArduPilot	PX4 - QGC
count	4	5	3
target_system	1	1	1
target_component	1	1	1
mission_type	0	0	0
Message CRC	0x8356	0xa63c	0x05da

Table 5.2: MISSION_REQUEST(0)

	MP - PX4	MP - ArduPilot	PX4 - QGC
seq	0	0	0
target_system	255	255	255
target_component	190	190	190
mission_type	0	0	0
Message CRC	0xb054	0x3f3d	0xc007

Table 5.3: MISSION_ITEM_INT(0)

	MP - PX4	MP - ArduPilot	PX4 - QGC
param1	0	0	0
param2	0	0	0
param3	0	0	0
param4	0	0	nan
x	0	-353632621	473977511
y	0	1491652374	85456077
z	100	584,19	50
seq	0	0	0
command	22	16	22
target_system	1	1	1
target_component	1	1	1
frame	3	0	3
current	0	0	1
autocontinue	1	1	1
mission_type	0	0	0
Message CRC	0xcdd4	0xaf50	0x165f

Table 5.4: MISSION_REQUEST(1)

	MP - PX4	MP - ArduPilot	PX4 - QGC
seq	1	1	1
target_system	255	255	255
target_component	190	190	190
mission_type	0	0	0
Message CRC	0x89c5	0x61e8	0xcbb38

Table 5.5: MISSION_ITEM_INT(1)

	MP - PX4	MP - ArduPilot	PX4 - QGC
param1	-	0	0
param2	-	0	0
param3	-	0	0
param4	-	0	nan
x	-	0	473978591
y	-	0	85455993
z	-	100	500
seq	-	1	1
command	-	22	16
target_system	-	1	1
target_component	-	1	1
frame	-	3	3
current	-	0	0
autocontinue	-	1	1
mission_type	-	0	0
Message CRC	-	0xd0e1	0xd49c

The communication between PX4 AutoPilot and Mission Planner fails for MISSION_ITEM_INT(1), so more data from other two cases are not needed for comparison.

Table 5.6: MISSION_ACK

	MP - PX4	MP - ArduPilot	PX4 - QGC
target_system	-	255	-
target_component	-	190	-
type	-	0	-
mission_type	-	0	-
Message CRC	-	0x3d60	-

5.3.1 Identifying the Affected from the Differences Components

These messages impact the structure of the mission rather than the mission upload process itself. Consequently, the uploaded mission may contain inaccuracies.

Differences between Mission Planner and compatible GCS (QGC)

Comparing Mission Planner and QGC using PX4 AutoPilot:

1. MISSION_COUNT is +1 with Mission Planner.

2. MISSION_ITEM_INT(0), with QGC pram4 = nan, x,y = first waypoint and current = 1 (=true), with Mission Planner param4 = 0, x,y = 0 and current = 0 (=false).

Differences between PX4 and compatible AutoPilot (ArduPilot)

The message sent by Autopilot is MISSION_REQUEST and there is no difference between PX4 AutoPliot and ArduPilot, but there are differences located in other messages. More specifically comparing PX4 AutoPilot and ArduPilot using Mission Planner:

1. MISSION_COUNT is +1 with ArduPilot.

2. MISSION_ITEM_INT(0), with Ardupilot ,is the home position , command = 16 (the scheduled action for the waypoint) and frame = 0 (The coordinate system of the waypoint. Global (WGS84) coordinate frame + MSL altitude. First value / x: latitude, second value / y: longitude, third value / z: positive altitude over mean sea level (MSL).), with PX4 AutoPilot is the altitude , command = 22 (the scheduled action for the waypoint) and frame = 3 (The coordinate system of the waypoint. Global (WGS84)

coordinate frame + altitude relative to the home position. First value / x: latitude, second value / y: longitude, third value / z: positive altitude with 0 being at the altitude of the home position.).

3. MISSION_ACK is send only by ArduPilot

5.4 Summary

After a thorough comparison of the data in these cases, it becomes evident that these discrepancies are not the primary cause of the error. The critical issues are twofold:

1. From the perspective of the autopilot, these messages are merely data and do not directly cause the error.

2. The core problem lies in the unresponsiveness of the Ground Control Station (GCS), which disrupts the communication flow.

It is increasingly likely that there exist additional implementation differences beyond those already identified.

Chapter 6

Related Work

The field of UAVs and their associated technologies has witnessed remarkable growth, fueled by advancements in autonomous flight control, communication protocols and ground control systems, but because of that is not possible to set limits and protocols for the smooth communication of different modules of a UAV system. This section should provide a comprehensive overview of pertinent studies and developments that serve as the bedrock for the present research, however there is no research or documentation, other than MAVLink's messaging protocol [5], about the connectivity between Mission Planner GCS and PX4 AutoPilot.

Building upon that knowledge, the present investigation sets out to address important gaps and challenges concerning UAV Autopilot Systems and connectivity, and the need to provide to the user more options for their UAV setup. By adopting a holistic perspective, our study aspires to contribute to the establishment of seamless communication between the most commonly used UAV framwork and interface.

Chapter 7

Conclusion

7.1 Summary and Conclusions

The purpose of this thesis was to enhance communication between PX4 AutoPilot and Mission Planner GCS. After research was done about the communication used between them, the differences of the MAVLink Protocol were tested and analyzed, coming to the conclusion that those differences affect the correctness of the mission uploaded but not the uploading process. Given the current limitations in documentation and resources, the discrepancies that are the cause of the communication issue remain challenging to pinpoint conclusively, however this research is giving an insight to the problem and the path for locating it.

7.2 Future Extensions

In the future, the code responsible for the MAVLink messages can be located and analyzed, while checking for possible messages that affect the communication but do not belong in the mission upload process. This way, the reason that GCS does not send the second waypoint can be found and the code of either GCS or PX4 AutoPilot can be edited, so we can achive communication.

Bibliography

[1] Px4 user guide. https://docs.px4.io/main/en/concept/px4_systems_architecture.html.

[2] Mavlink. https://mavlink.io/en/services/command.html.

[3] Mohsan SAH, Khan MA, Noor F, Ullah I, and Alsharif MH. Towards the unmanned aerial vehicles (uavs): A comprehensive review. *Drones*, 6(6):147, 2022.

[4] Soulimane Kamni, Antoine Bertout, Emmanuel Grolleau, Gautier Hattenberger, and Yassine Ouhammou. Easing the tuning of drone autopilots through a model-based framework. *Journal of Computer Languages*, 77(101240):2590–1184, Nov. 2023.

[5] Mavlink. https://mavlink.io/en/.

[6] Px4 autopilot. https://px4.io/software/software-overview/.

[7] Mission planner home. https://ardupilot.org/planner/docs/mission-planner-overview.html.

[8] Wireshark user's guide. https://www.wireshark.org/docs/wsug_html/#ChIntroWhatIs.

[9] Ardupilot. https://ardupilot.org/#.

[10] Qgc. http://qgroundcontrol.com/.