Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2018

Nuno Humberto Mendes Gonçalves Paula

Controlo Multi-drones com Suporte a Missões Autónomas

Multi-drone Control with Autonomous Mission Support

Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2018

Nuno Humberto Mendes Gonçalves Paula

Controlo Multi-drones com Suporte a Missões Autónomas

Multi-drone Control with Autonomous Mission Support

"*When you are developing a new technique, there are no recipes to copy, textbooks to consult, or manuals to read to pass on those little tips and secrets that guarantee success. You end up having to try any and every permutation of conditions and ingredients. You are never quite sure which of the many factors is really significant, how they act with and against one another, and so on. To sort out all those variables requires carefully designed trials. This is basic experimentation at its toughest and, if you succeed, at its best.*"

— John Craig Venter

Universidade de Aveiro
2018

Departamento de Eletrónica,
Telecomunicações e Informática

Nuno Humberto Mendes Gonçalves Paula

Controlo Multi-drones com Suporte a Missões Autónomas

Multi-drone Control with Autonomous Mission Support

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica da Doutora Susana Sargento, Professora Associada com Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor André Braga Reis, investigador do Instituto de Telecomunicações.

Apoio financeiro do POCTI no âmbito do III Quadro Comunitário de Apoio.

Apoio financeiro da FCT e do FSE no âmbito do III Quadro Comunitário de Apoio.

o júri / the jury

presidente / president

Prof. Dr. José Luís Costa Pinto de Azevedo

professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Prof. Dr. Luís Miguel Pinho de Almeida

professor associado do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Prof. Dra. Susana Sargento

professora associada com agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos /
acknowledgements

Obrigado mãe, pai e Ana. Obrigado a todos os meus amigos pelo apoio e por estarem presentes, de uma forma ou outra. À professora Susana Sargento, obrigado pela integração no fantástico grupo que é o NAP e pelo tempo gasto na orientação e correção deste documento. Um obrigado especial ao Bruno Areias, André Reis e André Martins, que ajudaram no desenvolvimento deste trabalho diariamente e não descansaram enquanto não estivesse concluído. Muito obrigado.

Palavras Chave

drones, controlo remoto, vôo autónomo, missões autónomas, multi-drone, missões colaborativas

Resumo

Com os avanços recentes na redução do tamanho dos sensores e instrumentos, assim como na redução de custos dos mesmos, a utilização de drones é cada vez mais comum e abrange um número cada vez superior de casos de utilização tais como missões de procura e resgate, agricultura e monitorização ambiental. Contudo, a maior parte das alternativas existentes para controlo ainda exigem a atenção constante de um piloto, limitando a conveniência da execução de missões complexas, sobretudo quando nelas participa mais que um drone.

Os principais fabricantes de drones e controladores de vôo, no entanto, disponibilizam cada vez mais frequentemente interfaces para a execução de funções básicas de vôo autónomo, como por exemplo a descolagem, aterragem e a navegação baseada em coordenadas geográficas. A existência cada vez mais comum destas interfaces permite a integração de drones em plataformas que têm como objectivo a abstração do seu controlo directo.

Esta dissertação propõe uma solução modular completa para controlo de um ou mais drones, permitindo a um utilizador inexperiente o planeamento, execução e monitorização de missões complexas com vários participantes, implementando também a funcionalidade necessária para a colaboração de vários drones na execução de uma missão.

A solução proposta consiste numa plataforma modular, composta por componentes que são executados de forma independente. Cada componente é individualmente desenvolvido para executar tarefas específicas como a comunicação com o controlador de vôo, a aquisição e armazenamento de telemetria e o planeamento de missões. Os componentes realizam as suas interações através da utilização de filas de mensagens, enquanto a interação com o utilizador é realizada através de aplicações intuitivas web ou mobile.

As funcionalidades da solução proposta são avaliadas através da execução de quatro testes distintos, que representam cenários típicos em que a plataforma de controlo pode ser usada. Estes testes cobrem a utilização de um ou mais drones, sendo que as duas primeiras tarefas são executadas apenas por um drone e as últimas duas representam cenários em que vários drones colaboram para alcançar um objectivo comum.

Keywords

drones, remote control, autonomous flight, autonomous missions, multi-drone, collaborative missions.

Abstract

Recent advancements regarding miniaturization of sensors and instruments, as well as the reduction of their cost, promoted a growth in the usage of drones in an increasingly wide range of scenarios such as search and rescue, agriculture and environmental monitoring. However, most currently available mechanisms for drone control still require a constantly aware pilot, thus limiting the convenience of executing complex missions, especially when more than one drone is involved.

Major drone and flight controller manufacturers, however, are displaying an increasing interest in providing programming interfaces and development kits that enable the execution of basic autonomous flight, including commands such as taking off, landing and waypoint navigation. These interfaces facilitate the integration of said drones in platforms that aim to abstract manual control from their users.

This dissertation proposes a complete and modular solution for controlling one or more drones, enabling an inexperienced user to plan, execute and monitor complex missions with various participants, also implementing the required functionality for the collaboration of a set of drones in the execution of such missions.

The proposed solution consists in a modular platform composed of loosely coupled components. Each component is individually designed to handle specific tasks such as flight control hardware interfacing, telemetry acquisition and storage, and mission planning. Components accomplish their interactions by using message brokers, while user interaction is achieved through intuitive web and mobile applications.

The functionality of the solution is evaluated through the completion of four experiments, which represent typical scenarios where the control platform may be used. These experiments cover both single-drone and multi-drone functionality, with the first two covering tasks carried out by one drone, while the last two represent scenarios where multiple drones collaborate towards a common goal.

Contents

List of Figures

List of Tables

Acronyms

ADC	Analog-to-Digital Converter
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
BEC	Battery Eliminator Circuit
CPU	Central Processing Unit
CSI	Camera Serial Interface
ESC	Electronic Speed Controller
FPV	First-Person View
GCS	Ground Control Station
GPIO	General Purpose Input Output
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
JSON	JavaScript Object Notation
LLA	Latitude Longitude Altitude
LXC	Linux Containers
LiPo	Lithium Polymer
MAC	Media Access Control
MCU	Microcontroller Unit
MQTT	Message Queuing Telemetry Transport
NDIR	Nondispersive Infrared
NED	North East Down
PID	Proportional Integral Derivative
PPM	Pulse Period Modulation
REST	Representational State Transfer
RTOS	Real-Time Operating System
SBC	Single-Board Computer
SDK	Software Development Kit
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
VCP	Virtual COM Port
VTOL	Vertical Take-off and Landing
XML	eXtensible Markup Language

CHAPTER 1

Introduction

This chapter introduces the topics covered in this dissertation. It starts by describing the context and motivation for this work, followed by what objectives are expected to be accomplished with it. This chapter also describes the prototypes which resulted from this work. Finally, a document outline is presented, providing an overview of each chapter.

1.1 Motivation

Since the beginning of aviation, controlling an aircraft without requiring a pilot aboard has been a difficult challenge. These unmanned vehicles typically go by the name of Unmanned Aerial Vehicle (UAV), or simply drone.

Although initially developed for usage in military situations during the First World War, recent miniaturization improvements and cost reductions have made drones affordable for civil purposes, popularizing their usage in professional photography or even as toys.

Recent research efforts have also promoted the usage of drones in an increasingly wide range of scenarios such as search and rescue, building inspection, environmental monitoring, agriculture, area patrolling and surveying.

Drones can be controlled manually, using the sticks of a radio remote control. This is the most common method of controlling a drone, requiring, however, an experienced pilot. More advanced control mechanisms allow whole missions to be assigned to a drone. These missions are composed of several high-level flight commands which are sequentially executed by a drone. Ideally, these flight commands are executed autonomously, requiring no user interaction once the mission is initiated.

Most commercially available drones still depend on manual remote control from a limited distance, requiring a constantly aware pilot dedicated to each drone and thus narrowing the convenience for execution of complex or repetitive missions, particularly when they require the participation of more than one drone.

Basic navigation functionality, however, is increasingly being supported by major flight controller manufacturers. This, along with the increasing availability of APIs and Software

Development Kits (SDKs) offered by such manufacturers, enables the integration of drones in platforms that abstract and simplify their control.

Chaumette (2017) in the chapter "Collaboration Between Autonomous Drones and Swarming" in "UAV Networks and Communications" [1] suggests following a research direction which seeks combining the advantages of a single UAV with the advantages of a swarm. These include the possibility of continuously executing a mission even when an unexpected event causes one drone to land or cancel its execution, inspiring an autonomous replacement scenario. Here, the potential for increased mission flexibility is also suggested, in which a set of drones dynamically adapts to a mission, for example, to increase the area coverage capacity.

In order to overcome the current limitation and lack of abstraction in the control of one or more drones, this dissertation proposes a modular control solution which enables an unexperienced user to plan, execute and monitor both simple and complex missions which may involve one or more participating drones, while also implementing the functionality required for the collaboration scenarios described above.

1.2 Objectives

With the goal of developing a drone control solution, a series of objectives must be accomplished. These include:

- Building a drone system which can easily be integrated in the proposed control platform.
- Abstracting drone control through the usage of high-level commands.
- Setting up the communication mechanisms required for monitoring drone status and delivering high-level control commands.
- Developing mechanisms for mission execution which are not dependent on user interaction.
- Creating collaboration methods which enable more than one drone to participate in a mission, either simultaneously or in replacement scenarios.
- Providing a set of tools and applications which enable monitoring and controlling connected drones through the usage of user-friendly graphical interfaces or a REST API.

This dissertation aims on working towards a platform which implements the functionality required for each of its objectives. Extensive optimization of each of its capabilities can be further achieved as a subject for future work.

1.3 Published Results and Prototypes

Three articles were published or submitted with the work described in this dissertation, the first of which with work carried out before the formal start of this dissertation. These include:

- "Automated Flying Drones Platform for Automatic and Remote Sensing", published and presented in INForum 2017.

- "Towards an Automated Flying Drones Platform", published and presented in VEHITS 2018.
- "Multi-drone Control with Autonomous Mission Support", submitted to UNmanned Aerial vehicle Applications in the Smart City: from Guidance technology to enhanced system Interaction (UNAGI) Workshop, IEEE International Conference on Pervasive Computing and Communication (IEEE PERCOM 2019), and awaiting approval.

Additionally, and as a result of this work, the following prototypes were developed:

- A REST API for issuing high-level commands to any connecting drone, also featuring a graphical interface.
- A mobile application for both Android and iOS devices that enables a user to summon a drone to his current location.
- A mission planning web application that allows users to drag-and-drop complex sequences of commands (e.g. arming, taking off, waypoint navigation, manually request replacement and collaboration, landing, disarming) and send them to any connected drone.
- A web application that enables a user to monitor the connection status of any drone.
- A mission reviewing web application that enables users to upload log files of previously executed missions and view the followed path in a map.
- An integrated instance of Grafana configured to store and display telemetry data of every connected drone, also allowing the review of historical data.

1.4 Outline

This section provides a description of each chapter along this dissertation.

- **Chapter 1** - Contains an introduction of the dissertation, including its motivation and objectives.
- **Chapter 2** - Describes the actual advancements in areas that are relevant to this work, covering drones, flight controllers and flight controller firmware, along with current remote control solutions.
- **Chapter 3** - Contains a description of the proposed architecture, also covering its scenarios and requirements.
- **Chapter 4** - Describes the implementation details of each proposed architecture component, also covering the building and wiring of the drone system.
- **Chapter 5** - Presents four different experiments that evaluate the performance and functionality of the platform. For each, the description, methods, evaluation procedure and results are detailed.
- **Chapter 6** - Completes this dissertation providing general conclusions and suggestions for future work.

CHAPTER 2

Related Work

This chapter provides an overview of the research efforts and concepts that are essential for understanding the topics related to this area and the solution that is proposed in this dissertation. Firstly, we provide a brief description of drone systems and their usages. A section dedicated to currently used flight controllers is also provided, covering both proprietary and community developed flight controllers, along with their capabilities. Next, a section covers flight controller firmware, including the currently available firmware choices, their usage, activity and recent progress. Ground control solutions are also featured in a section, covering existing options for ground control that are in production or active in the market. Despite the focus on commercially available and community developed platforms, components and systems, the chapter finishes with a brief overview of a few related research efforts towards drone control platforms, along with a brief discussion.

2.1 Drones

Drone and UAV are the most popular terms for referring to Unmanned Aerial Systems [2]. These have been a challenge since the beginning of aviation, with the objective of controlling an aircraft without the necessity of a pilot aboard, for usage in both civil and military situations [3]. In the last few years the advancements, miniaturization and cost reduction of sensors and instruments promoted the rapid development and adoption of drones in remote sensing and mapping scenarios [4].

Mostly used drone types include [5]:

- **Multi-rotors**, which enable Vertical Take-off and Landing (VTOL) and are easy to use, but are also limited to short flight times.
- **Fixed-wing**, which are capable of executing very long flights with great speeds, but provide no hovering capabilities and require horizontal space when taking off and landing.

Becoming increasingly common in a broad range of application areas including urban scenarios [6]–[8], actual research efforts also include risk assessment and analyzing the impact of drones on public safety [9]. Application areas which are becoming increasingly assisted by drones include:

- **Search and rescue**, in which drones can be used to reach disaster locations and drop first aid payloads [10], [11].
- **Building inspection**, in which drones may be used for detection of heat leakage, evaluating building isolation and rooftop condition [12].
- **Area patrolling**, in which drones are used to carry out surveillance tasks by police forces.
- **Environmental monitoring**, in which drones may be used for monitoring air quality at high altitudes [13], for inspecting rocky shores [14] or for accessing estuarine environments [15].
- **Agriculture**, in which drones acquire aerial imagery for generating soil maps [16], for estimating positions and heights of tree plantations [17], for accessing forest regeneration [18] or monitoring agronomic parameters remotely [19].
- **Tourism**, where video is recorded live by a drone and transmitted to tourists which may be sitting at home [20].

Nowadays, DJI [21] is the major manufacturer of commercially available drones [22], representing 50 percent of the consumer drone market in North America in 2017 [23]. Most commercially available ready-to-fly drones, however, allow little expandability since they rely on proprietary, closed-source flight controllers. This aspect is further explored in section 2.2.

2.2 Flight Controllers

The flight controller is a board which is responsible for the low-level processing and operations of a drone [24]. It implements sensors which enable the attitude estimation of the drone, and is permanently connected to the Electronic Speed Controllers (ESCs), which enable controlling the motors. By implementing the control logic which, among commercial solutions, typically consists of a Proportional Integral Derivative (PID) controller with tunable parameters, the flight controller is able to keep the drone flying in a stable state. Typically, commercial flight controllers integrate all sensors and processing components in a single board, which proves useful when accounting for size and weight limitations of a drone [25].

2.2.1 Closed Hardware

Major drone manufacturers, such as DJI, fabricate their own flight controllers such as NAZA-M, NAZA-M V2 and NAZA-M Lite [26], which however rely on proprietary firmware and provide no SDK or any means of remotely issuing control commands, unless the user also acquires a 2.4GHz Datalink per drone, which only enables issuing pre-flight waypoints [27]. Figure 2.1 shows NAZA-M V2 with a compatible GPS sensor. The latest generation of flight

Figure 2.1: DJI NAZA-M V2 flight controller with GPS sensor. [28]

Figure 2.2: DJI N3 flight controller with included power management unit and GPS sensor. [29]

controllers developed by DJI, however still relying on closed-source firmware, provide a wider expandability potential. These include the N3 [30] and the A3 [31]. Although having a high price tag of 369€ [32] and 1099€ [33], respectively, N3 and A3 can make use of the DJI Onboard SDK, which enables sending flight control commands such as taking off, waypoints and landing through the usage of a companion board or computer which connects to the flight controller using a Universal Asynchronous Receiver-Transmitter (UART) [34]. DJI also has a Pro version of the A3 available for purchase, featuring triple redundancy of every sensing module, available for 1699€ [35].

Besides DJI, Parrot has also recently launched a flight controller solution, CHUCK, which is available for 788€ [36]. CHUCK is part of a subset of flight controllers which can be programmed with open-source firmware, besides consisting in closed hardware. It relies on advanced sensors for navigation, featuring a pitot tube as an airspeed sensor. CHUCK, like the DJI N3, provides an SDK which enables its integration in user-developed applications [37]. Figure 2.3 shows a representation of CHUCK.

Also belonging to this subset, Edge is the latest flight controller developed by Emlid. Available for 600€, Edge, shown in Figure 2.4, includes an ARM processor and is able to run a Debian-like operating system, thus enabling a user to run his applications directly on it without requiring a companion computer [39].
In spite of their high prices, N3, A3, CHUCK and Edge are all highly capable and expandable flight controllers which could be integrated in a drone control platform such as the one that is proposed in this dissertation.

Figure 2.3: Representation of Parrot CHUCK. [38]

Figure 2.4: Emlid Edge flight controller. [40]

2.2.2 Community-developed Hardware

Along with companies that commercialize flight controllers and drones, several communities and research projects also work towards the creation of flight controller alternatives [41]. An example of such alternative is Pixhawk 4. Based on open hardware design, Pixhawk 4 is an advanced flight controller which uses NuttX as its Real-Time Operating System (RTOS) [42], [43]. Pixhawk flight controllers are officially supported by Dronecode, a flight stack project hosted under the Linux Foundation [44]–[46]. Although being available for a price of approximately 180€ [47], which is high relatively to other open hardware flight controllers, Pixhawk is one of the most adopted flight controllers for research projects [48]–[52]. For computationally intensive tasks, however, a separate companion board or computer is still needed [53]. Pixhawk 4 is shown in Figure 2.5.

Several open hardware alternatives exist which provide navigation capabilities and can be acquired for lower prices than the Pixhawk 4. An example of one of these alternatives is the OpenPilot Revolution. Revolution is a flight controller which features an STM32F4 ARM Microcontroller Unit (MCU). Developed by the OpenPilot community [55], the Revolution features a gyroscope, an accelerometer, a magnetometer and a barometer, along with compatibility with external GPS sensors and compasses [56]. Revolution is a particularly interesting choice for a flight controller, since it provides autonomous flight and navigation capabilities for a reduced price of approximately 47€ [57]. This flight controller also features a Universal Serial Bus (USB) port which enables using a companion computer to input navigation commands, making it appropriate for integration in a drone control platform.

Smaller aircraft which do not require GPS navigation may use simpler and less expensive flight controllers. Among the most popular flight controllers of this type there is the CC3D, which is also developed by the OpenPilot community [58]. This flight controller is available for approximately 9€ [59] and, although extremely limited in its autonomous flight functionality, contains all necessary sensors for manual drone flight, representing an affordable flight controller option for hobbyists. CC3D is shown in Figure 2.7.

Figure 2.5: Representation of Pixhawk 4. [43]

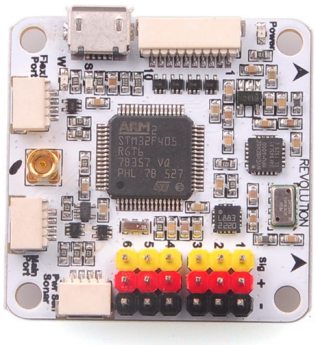

Figure 2.6: OpenPilot Revolution flight controller. [54]

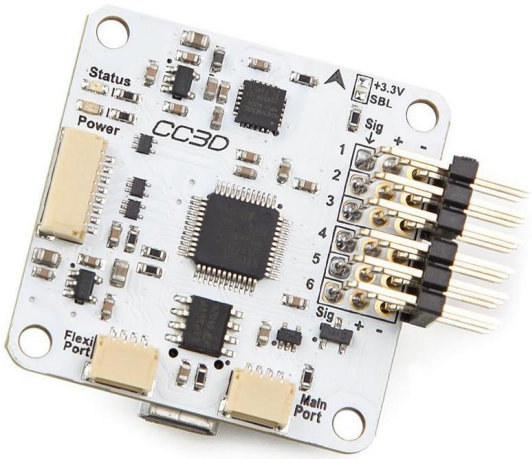

Figure 2.7: OpenPilot CC3D flight controller. [60]

2.3 Flight Controller Firmware

Community-developed hardware, along with certain closed hardware such as Parrot CHUCK and Emlid Edge, enable users to install their own flight controller firmware. This is of particular interest since many open-source flight controller firmware projects are available for different purposes such as GPS navigation, drone racing and acrobatic flight. This section covers some of the most used open-source flight controller firmware, starting with the option which is adopted by both Parrot CHUCK and Emlid Edge, Ardupilot. Ardupilot is one of the most advanced and reliable open-source flight controller software available. Supporting various vehicle systems such as planes, multi-rotors, helicopters and boards, Ardupilot is installed in over 1.000.000 vehicles worldwide [61]. At the time of writing, Ardupilot supports advanced features such as obstacle avoidance, precision landing and the usage of parachutes and grippers [62]. Interfacing with Ardupilot is possible using MAVLink, a lightweight messaging protocol for communicating with drones that is also part of the DroneCode project [63].

Available for Pixhawk flight controllers and also part of the DroneCode project, PX4 is an open-source project that aims to become the most popular flight controller firmware to the industrial, academic and enthusiast communities [67]. PX4 is built for aerial robotics, supporting path planning, obstacle avoidance and GPS based navigation [68]. In a similar way to Ardupilot, MAVLink can be used to interface with the PX4. Although advanced,

Figure 2.8: Ardupilot logo. [64]

Figure 2.9: PX4 logo. [65]

Figure 2.10: dRonin logo. [66]

since PX4 only supports a limited subset of flight controller hardware, using it would imply purchasing flight controllers with a relatively high price, such as the Pixhawk 4.

Also representing an interesting flight controller firmware option, the first release of the dRonin project was launched in early 2016 [69]. It can be used for both drone racing and autonomous flight. Among others, dRonin is compatible with flight controllers that belong to the OpenPilot family, including the OpenPilot Revolution [70]. This means that, using dRonin, it is possible to build a drone with GPS navigation capabilities using inexpensive flight controllers. Interfacing with dRonin can be accomplished using UAVTalk, a flexible open binary protocol designed for communication with drones [71]. When using a flight controller with a USB port such as the OpenPilot Revolution, dRonin enables the usage of a Python API to receive telemetry and send control commands to the flight controller using a companion computer. dRonin also features an autotune mode which enables the ideal constants for the PID controller to be automatically acquired by the flight controller during its first flight.

2.4 Remote Control Solutions

This section describes existing solutions for the remote control and management of drones that are currently in production or active in the market.

2.4.1 Open-source Solutions

Flight controller firmware developers usually also produce their own GCS that allows a user to perform the initial setup and configuration of a drone. These GCSs usually also allow the monitoring and control of a drone to some extent, using a Bluetooth or Wi-Fi wireless link. The DroneCode project features an open-source GCS, QGroundControl, which provides full flight control and vehicle setup for flight controllers running PX4 or Ardupilot [72]. It is shown in Figure 2.11.

QGroundControl, available for Windows, macOS, Linux, Android and iOS, allows a user to monitor the position and heading of a drone, while also tracking its flight and showing it in a map. It also allows sending sequences of commands to a drone prior to its flight and then requesting their execution. In order to wirelessly connect with a drone, a Wi-Fi adapter or a SiK telemetry radio may be used, allowing ranges of about 300m when using out-of-the-box antennas [74].

dRonin also provides its own GCS for Windows, macOS, Linux and Android, which implements similar functionality to QGroundControl. It is shown in Figure 2.12. Neither one of these platforms implements any kind of multi-drone collaboration functionality.

Figure 2.11: Screenshot of the QGroundControl GCS. [73]

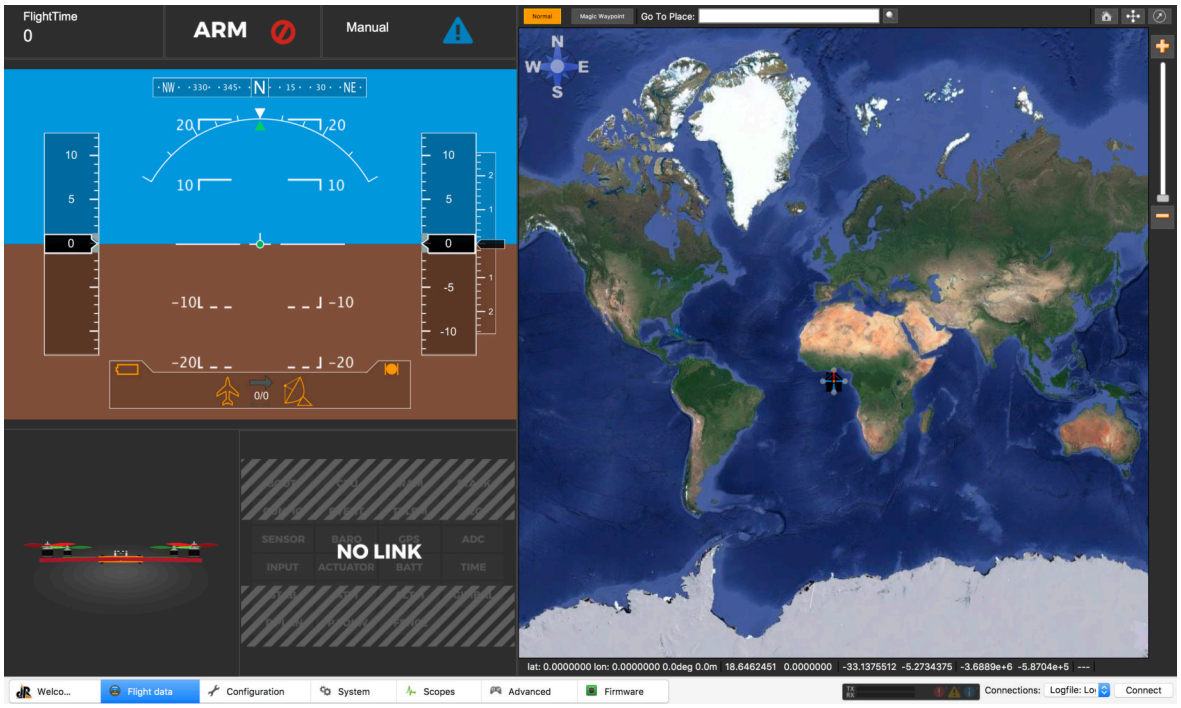

Figure 2.12: Screenshot of the dRonin GCS.

2.4.2 Commercially Available Solutions

Several companies also provide their own solution for controlling drones remotely. Each solution varies in functionality and flight controller compatibility. Some solutions focus on supporting mapping and surveying use cases while others implement no functionality but provide an API for the user to create its applications.

As an example of such solution, DroneDeploy is a cloud-based platform that enables users to create maps and 3D models of large areas. It provides a mobile application for both Android and iOS which allows creating flight plans with automatic take-off, image capture and landing, while also providing a First-Person View (FPV) live video stream [75]. DroneDeploy also provides a platform which enables large-scale flight management and map processing while supporting user management with differentiated role-based access control [76]. Live Map is another useful tool developed by DroneDeploy, shown in Figure 2.13, which allows users to generate maps in real-time without the need for a laptop or an Internet connection [77]. DroneDeploy, however, only supports DJI drones, providing no compatibility with community-developed flight controllers such as Pixhawk and OpenPilot Revolution [78]. DroneDeploy also provides no support for multi-drone interaction or collaborative missions.

Figure 2.13: Real-time map generation using DroneDeploy Live Map [77]

PrecisionHawk is also a company that provides drone-based solutions for various industry sectors such as agriculture, construction, energy, government and insurance [79]. Its main product, PrecisionFlight, provides features that are very similar to the mobile application developed by DroneDeploy [80]. A screenshot of PrecisionFlight is shown in Figure 2.14. PrecisionFlight Pro is also available, claiming to enable users to fly a drone remotely and from anywhere, although providing no details regarding the technologies that are used to accomplish this [81]. PrecisionFlight, just like DroneDeploy, is compatible with DJI drones only, while PrecisionHawk claims PrecisionFlight Pro also supports interacting with PX4 and Ardupilot-

based flight controllers. Like DroneDeploy, neither PrecisionFlight nor PrecisionFlight Pro provide support for multi-drone interaction or collaborative missions.

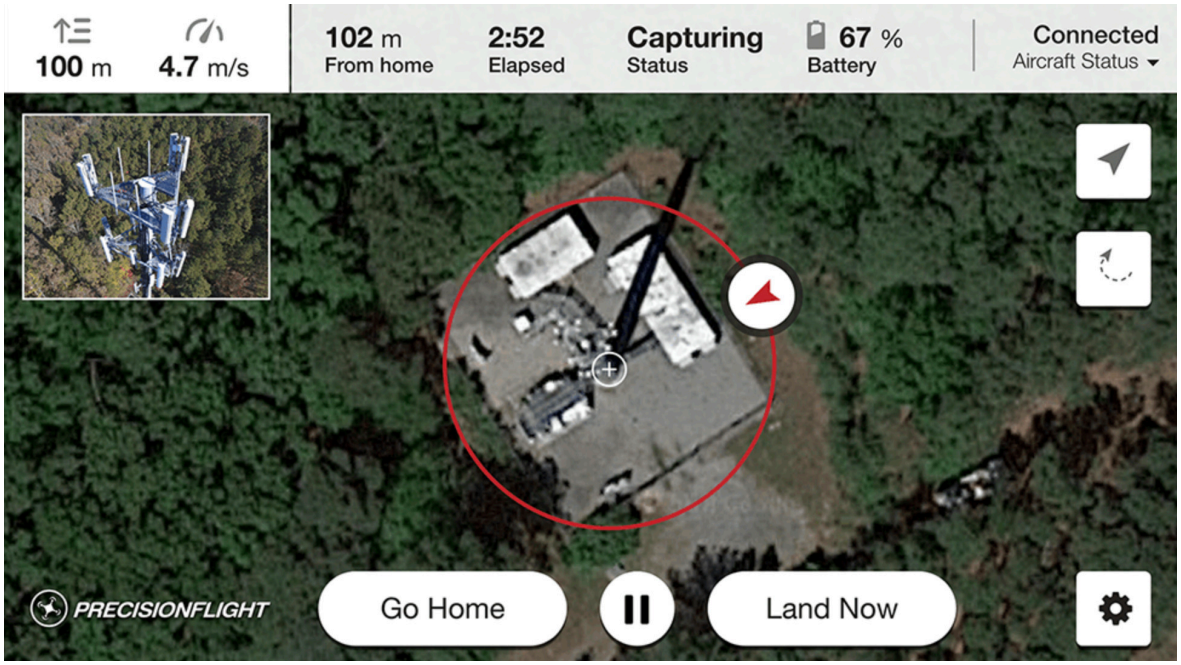

Figure 2.14: Screenshot of PrecisionFlight. [80]

Another interesting control solution is provided by FlytBase, a platform which enables the deployment of drones in cloud-based business applications [82]. The solution provided by FlytBase consists of a companion computer running FlytOS, a customized Linux distribution providing APIs and SDKs for building high-level drone applications [83]. The functionality provided by FlytBase is very extensive, supporting scenarios such as object tracking, video streaming and LiDAR integration. FlytBase, however, focuses on developing tools that enable clients to develop their own applications depending on their needs, instead of providing ready-made applications. FlytBase also claims to support DJI drones, along with PX4 and Ardupilot-based flight controllers. Although mechanisms to implement collaborative mission support are provided by FlytBase, multi-drone interaction tasks are not available out-of-the-box.

2.5 Related Work

This section provides an overview of the current research efforts regarding drone control or automation, or where drones are actively used to assist real-life tasks. These can either be generic or focused in specific use cases such as rescue, infrastructure inspection, surveillance or agriculture. Some examples of these projects are provided below.

2.5.1 ROLFER

The ROLFER project [11] proposes a drone control solution which enables swimmers in distress to call a drone by tapping a button on a smart watch. Communication range limitations are partially defeated since only a Short Message Service (SMS) message is required to call a drone. Its telemetry connection to the base station, however, makes use a local radio link and its range is not specified. Its usage scenarios are also limited since it consists in a rescue support system, providing no multi-drone support and no mission planning capabilities.

2.5.2 Automated Infrastructure Inspection

A system supporting complex mission planning and execution is proposed in [84]. This project specializes in providing a platform for infrastructure inspection using multi-rotor drones. Although it claims to enable a user to define complex missions which can be autonomously executed, it provides no support for multi-drone interaction and has a limited communication range due to the usage of a Wi-Fi access point.

2.5.3 Persistent Aerial Surveillance

Recent research (2018) from the Naval Postgraduate School in Monterey, California, resulted in the development of a platform which enables persistent mobile aerial surveillance [85]. This solution uses intelligent battery health management and drone swapping to maximize the time during which a zone may be covered by, at least, one drone without interruption. This mechanism for autonomous drone replacement enables a set of three quadcopters to take turns hovering above a location, achieving a total flight time of 98 minutes. Its communication range, however, is limited to half a mile.

2.5.4 Continuous Remote Control

The work described in [86] proposes an architecture for controlling a drone remotely. Through the usage of multiple cellular carriers, the authors claim to maintain drone operation even when one flies outside of the communication area of one cellular carrier. This solution enables a user to remotely control a drone while reducing the range limitations of typical communication mechanisms based on Wi-Fi or Bluetooth. The solution provides basic navigation functionality over IP, which includes waypoints (which must be set before flight), take-off, landing and camera control. This platform, however, provides no support for multi-drone interaction or collaborative missions.

2.5.5 Imaging System for Crop Monitoring

For usage in precision agriculture tasks, the work described in [87] presents the development of a multispectral imaging system. This solution consists of a quadcopter which is equipped with multiple cameras, which are triggered by an on-board SBC. In this system, drone control is accomplished by the usage of the Ardupilot flight controller firmware, running on a Pixhawk

variant flight controller. The quadcopter can either be controlled manually using a common remote control or, alternatively, through the usage of a mission planner which enables a user to select a polygon covering the area to be monitored. This solution provides no support for multi-drone control.

2.5.6 MedizDroids

MedizDroids is an ongoing project with the goal of researching the affordable use of drones for mosquito vector control and suppression [88]. This project aims to develop multi-rotors that can be used to reduce the transmission of infectious diseases such as malaria and dengue fever through insecticide spraying. Although the authors state that a service oriented architecture is being created for the purpose, implementation details are not provided. Drone system details are also not provided, but a price under 500$ is suggested for a ready-to-fly drone.

2.6 Discussion

By analyzing the referred research efforts and commercially available systems for drone control, we can observe that an increasing amount of solutions are becoming available. These solutions, however, are often highly specialized in unique tasks, thus lacking support for generic drone control and mission planning. An example case would be the ROLFER project, which focuses on payload dropping in search and rescue scenarios. It is also evident that multi-drone control capabilities are still uncommon among existing commercial or community-based solutions, with support for multi-drone collaboration being particularly rare. Although multiple platforms support the execution of real-time complex tasks involving large numbers of commands, most are severely limited either on drone and flight controller compatibility or on their communication range. Within this scope, we aim at developing one single drone control solution which simultaneously features long communication range, real-time high-level control, complex mission planning, multi-drone support and multi-drone task collaboration.

CHAPTER 3

Architecture

This chapter describes the proposed architecture, including components of both drone and ground systems. As an introduction, sample usage scenarios are provided along with their requirements. This chapter also covers the communication methods of said architecture, as well as the automation and multi-drone capabilities that are enabled by it. A global overview is also presented, showing the interactions among all described components.

3.1 Scenarios and requirements

When aiming for a platform which supports the monitoring and control of an arbitrary number of drones in a decoupled and abstracted manner, a series of sample scenarios should be taken into consideration. These are scenarios that, when fully supported by the platform, will show its ability to execute single navigation commands, as well as relatively more complex tasks, which require the combined use of one or more components designed for such scenarios.

3.1.1 Internal telemetry acquisition

Flight controllers generally incorporate an array of sensors. These sensors are commonly integrated in the flight controller board itself, dispensing the need for external connections, and include the combination of a gyroscope, an accelerometer and a magnetometer. Integrated sensors are read periodically during flight, providing the drone with crucial data about its angular velocity, acceleration and heading. Based on these measurements, small adjustments are continuously and automatically performed to the drone, allowing for its automatic stabilization, which is critical for autonomous flight. Other flight-related information can also be provided by the flight controller, including current Central Processing Unit (CPU) and memory usage, currently activated flight mode, actual position and attitude estimation. The acquisition of this information must be possible in order to keep track of the actual state of the drone.

3.1.2 External sensor reading acquisition

A drone may carry aboard sensors that are not directly integrated in the flight controller. These sensors may be required for augmenting navigation capabilities or may simply be carried by the drone with the goal of obtaining, e.g., aerial imagery or environmental measurements. As an example of an external sensor which is required in autonomous flight scenarios, a GPS receiver keeps a permanent, physical connection to the flight controller and enables the implementation of navigation capabilities. A drone may also carry an environmental measurement device such as a CO_2 sensor module, to provide periodic measurements that are not necessarily relevant for real-time flight but may later be used as an important factor in navigation decisions.

3.1.3 Geographic coordinate based drone control

As a fundamental requirement of the proposed solution, a method for positional drone control must be provided, to allow the user to move a drone to a desired location without manual control of the drone using a radio controller. Instead of focusing on keeping the drone in a stable state and manually adjusting its attitude, the user should only be responsible for issuing high-level commands based on geographic coordinates. Since external sensors may be subject to variable response times, drones should also be able to hover at a specific location for a desired duration. Additionally, the solution should allow users to arm or disarm a drone and also to move it to a position that is relative to its actual location. An example of this functionality would be to request the drone to move 10 meters heading North or South, or to simply raise its altitude by 5 meters. When all of the described functionality is achieved, low-level control requirements are effectively abstracted from the user.

3.1.4 Mission execution

Although a control mechanism based on geographic coordinates can effectively abstract the details of low-level control from the user, this mechanism can be further extended to support more complex operations. Besides supporting single control commands, the solution must also allow the user to prepare large sequences of these commands. As an example of an elementary mission, a user should be able to, for example, instruct the drone to arm, take-off and then travel to a specific location. More complex missions could involve large sets of geographic coordinates to be visited sequentially. The control solution must allow these missions to be entirely executed in an autonomous manner upon request of the user. Keeping track of the actual mission completion status is also part of the core functionality, by providing a way of obtaining the current position of any drone as well as the actual mission step which a drone is currently executing.

3.1.5 Collaboration

In situations where the complete execution of a mission is not possible (for example, when the remaining battery capacity is not enough to make any additional progress), the architecture

must be capable of automatically replacing the drone that is currently in flight. In these cases, a new drone should be quickly assigned to complete the remaining steps of the mission. Prior to resuming the remainder of this mission, the newly assigned drone must first be placed in the position of the drone that is being replaced. In addition to autonomous replacement functionality, the platform should provide support for the simultaneous collaboration of multiple drones when a mission is of increased complexity or is composed by a number of steps too large for a single drone to execute. Drones connected to the system should be able to request collaboration at any given time during the execution of a mission and, upon these requests, one or more new drones should be selected and assigned for collaboration by the platform.

3.1.6 Logging

It is crucial that modules which are developed for the architecture provide logs describing details regarding their behavior. In the scenarios described in section 3.1.1 and section 3.1.2, the logging of telemetry and sensor readings allows one to further analyze the gathered data, either for debugging purposes or in the event of a crash. Logging, however, should be present not only in telemetry and sensor reading acquisition scenarios but also when control mechanisms are employed. For the scenario described earlier in section 3.1.3, every high-level command issued by a user must be effectively logged. In the mission execution scenario from section 3.1.4, a wide range of interactions should also be logged. These include mission decoding and message processing timing measurements, details of each executed mission step, along with geographic coordinates and step execution duration.

3.2 Overall architecture

Taking into consideration both the drone side and ground side architectures which will be described, an illustration of the overall architecture is presented in Figure 3.1, featuring multiple drone systems which are connected to the ground side through broker message relaying. The usage of message brokers allows component communication to be simplified, since each component only requires a single connection through which messages are published and subscribed. This approach also enables components to limit message reception to specific topics which are required for their functionality by subscribing only to these topics. This process is further described in section 3.5.

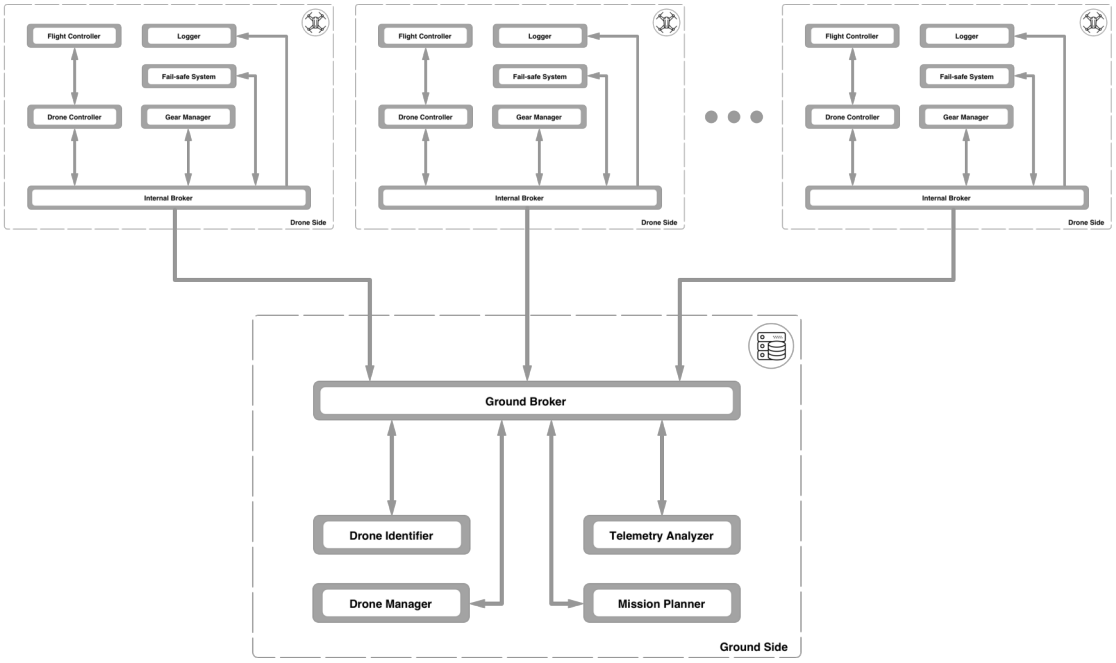

Figure 3.1: Overall platform architecture

3.3 Drone side

This section describes the platform architecture on the drone side, containing all components which must exist in every drone connected to the platform. Initially, an overview of these components is presented, followed by a brief description of each of them, along with details regarding their connections.

3.3.1 Overview

Taking into consideration the scenarios described in section 3.1, the drone-side architecture depicted in Figure 3.2 is established. The purpose and description of each architecture component is given in section 3.3.2, while details regarding component connections are described in section 3.3.3.

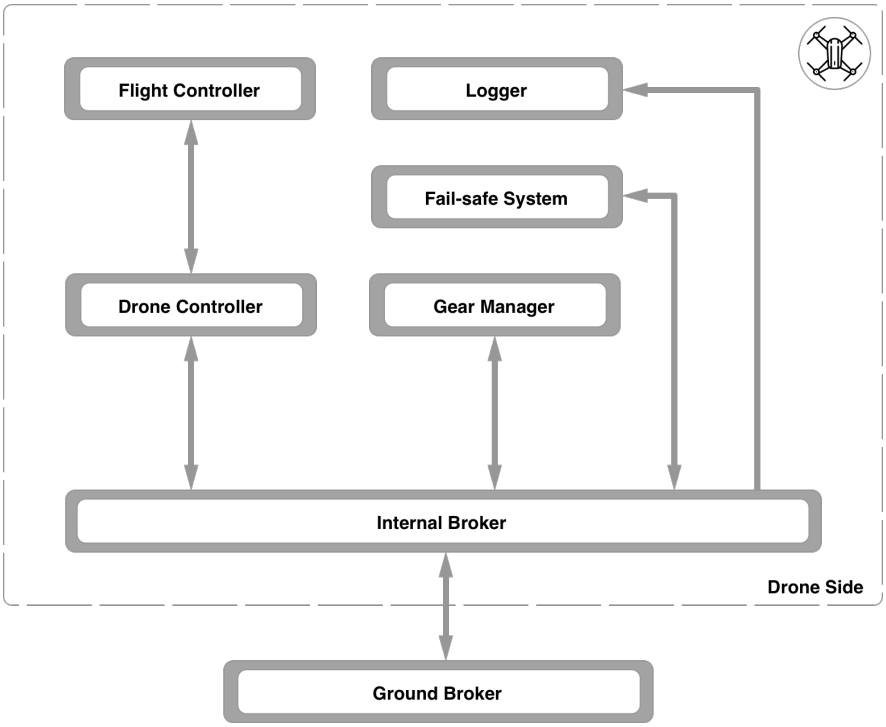

Figure 3.2: Overview of the drone side architecture

3.3.2 Component description

This section describes the motivation for developing each of the proposed architecture components, along with their functionality.

3.3.2.1 Flight Controller

Representing an essential piece for drone control, the **Flight Controller** holds a crucial place in the drone side architecture of the platform.

With reference to the first scenario, described in section 3.1.1, the architecture should be designed taking into consideration that it must be possible for internal telemetry information to be easily exported, providing architecture components which are external to the flight controller with a means of obtaining this information. If this requirement is fulfilled, it is possible to afterwards either display the information to the user in a dashboard or autonomously analyze it to trigger events or alarms.

Sensors such as the GPS receiver, external magnetometers or even battery voltage and current sensors are typically supported out-of-the-box by flight controllers and – since they can provide information that is necessary or relevant for flight operations – can be directly connected to them using commonly accepted communication interfaces such as UART, Inter-Integrated Circuit (I2C) or even simple Analog-to-Digital Converters (ADCs) in situations where a sensor provides an analog output. For all these cases, the external flight data connection displayed in Figure 3.3 can be used to additionally export data regarding these externally connected sensors.

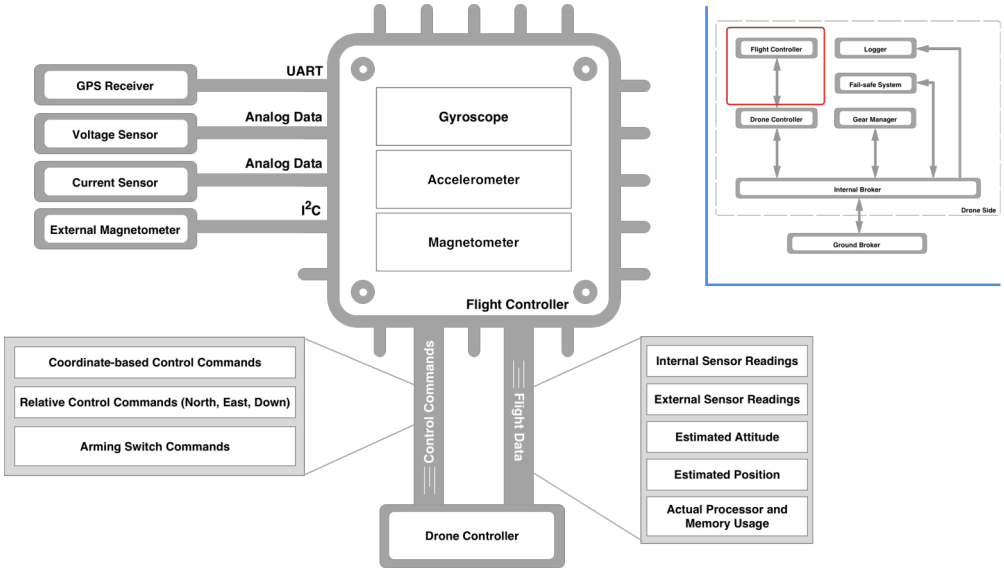

Figure 3.3: Flight Controller interaction supporting telemetry acquisition and sending control commands.

In order to implement the high-level control mechanisms described in section 3.1.3, section 3.1.4 and section 3.1.5, the **Flight Controller** will receive control commands that may either be relative to the actual position of the drone, based on geographic coordinates or even arming switch commands. These originate from the **Drone Controller**, as depicted in Figure 3.3. The **Flight Controller** also puts the burden of receiving and processing flight data on the **Drone Controller**.

3.3.2.2 Drone Controller

Another essential piece for drone control, although now consisting in a completely software-based component, is the **Drone Controller**. This component keeps a permanent connection to the **Flight Controller** and is responsible for making drone control functionality available in a high-level basis.

It is composed of several submodules which enable the functionality requested in the described scenarios. These are shown in Figure 3.4.

- **Flight Controller Interface**
  Since **Flight Controllers** typically rely on hardware implementations using diversified or even proprietary communication protocols, the **Drone Controller** must feature a layer which enables the abstraction of communication details that are specific to these implementations. This task is accomplished by the **Flight Controller Interface**, which is responsible for keeping direct, bi-directional communication with the **Flight Controller**, handling incoming flight data and crafting control commands.
- **Navigation Processor**
  As specified in the scenarios detailed in section 3.1.3, section 3.1.4 and section 3.1.5, low-level drone control requirements must be abstracted into general, high-level commands

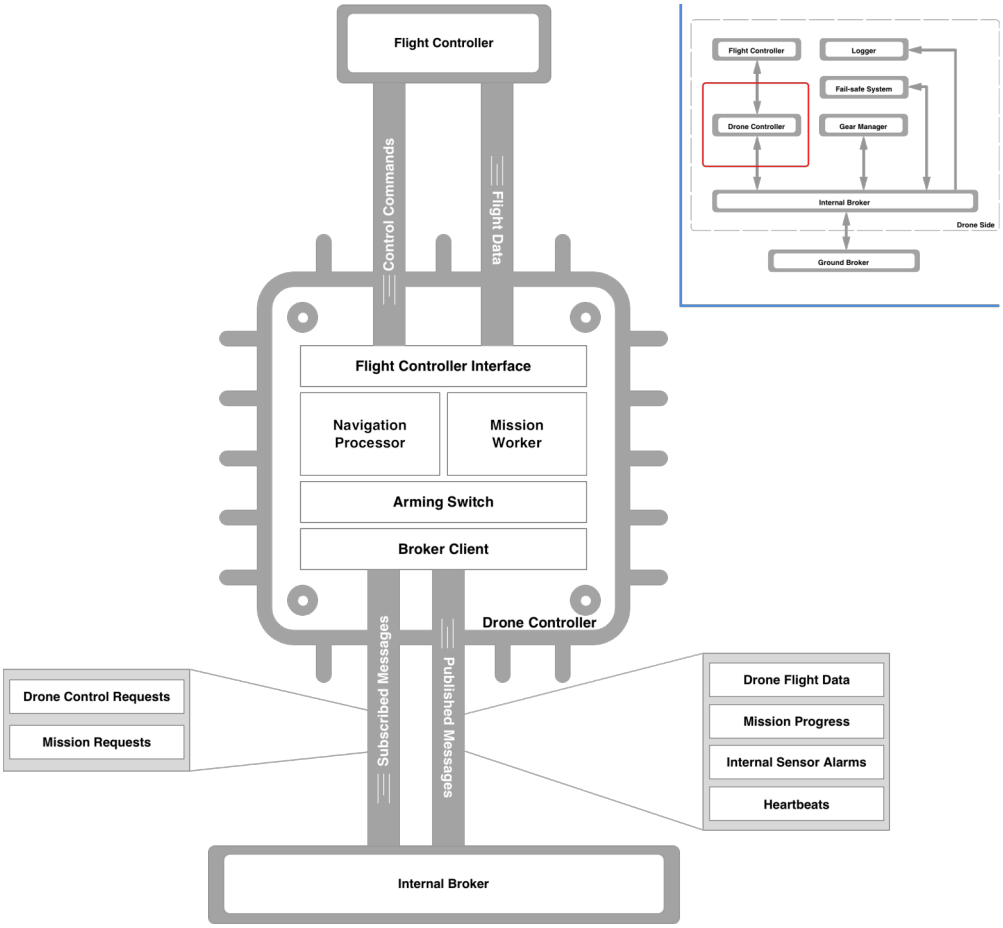

Figure 3.4: Drone Controller with submodules and internal broker interaction.

which may be based on geographic coordinates or relative to the current position of a drone. In order to achieve this functionality, the **Drone Controller** features the **Navigation Processor**, a submodule which is responsible for processing the desired control actions into commands that can be understood by the **Flight Controller**.

- **Mission Worker**

  Scenarios detailed in section 3.1.4 and section 3.1.5 imply that the platform must support the execution of missions, which are chains of control commands that are performed sequentially. The **Mission Worker** is a submodule which is responsible for parsing received mission requests, storing each of the mission steps in memory and sequentially feed each of them to the **Flight Controller**, while also keeping track of the progress of the current mission.

- **Arming Switch**

  Before initiating flight, drones must be placed in an activation state which is usually named *arming state*, the value for this state can either be *armed* or *disarmed*. A submodule of the **Drone Controller**, here named **Arming Switch**, is responsible for keeping track of the *arming state* of the drone and correctly issuing the low-level commands needed to either arm or disarm the drone.

- **Broker Client**

Every component which keeps a permanent connection to the **Internal Broker**, requires a **Broker Client**, a submodule which handles all broker communication, including the publishment of messages when requested by other submodules, along with the subscription and parsing of received messages.

In addition to the described functionality, the **Drone Controller** will periodically transmit heartbeats in order to announce its availability. This way, the **Drone Identifier**, which is a component specifically designed for this purpose on the ground side, can keep track of which drones are connected to the platform at a given time.

3.3.2.3 Gear Manager

With respect to the scenario described in section 3.1.2, the acquisition of external sensor data requires an additional component for interfacing with sensors that provide no compatibility with the flight controller.

Sensors such as the CO_2 module shown as an example in section 3.1.2 or a video camera may also use the common interfaces exemplified in section 3.3.2.1 or even Camera Serial Interface (CSI), but provide no integration compatibility with the flight controller. In these cases, an external component is required to obtain readings from the sensors. Just like internal telemetry, this information must also be made accessible to architecture components which are external to the flight controller, since it may contain relevant information for triggering events or alarms or for further logging purposes.

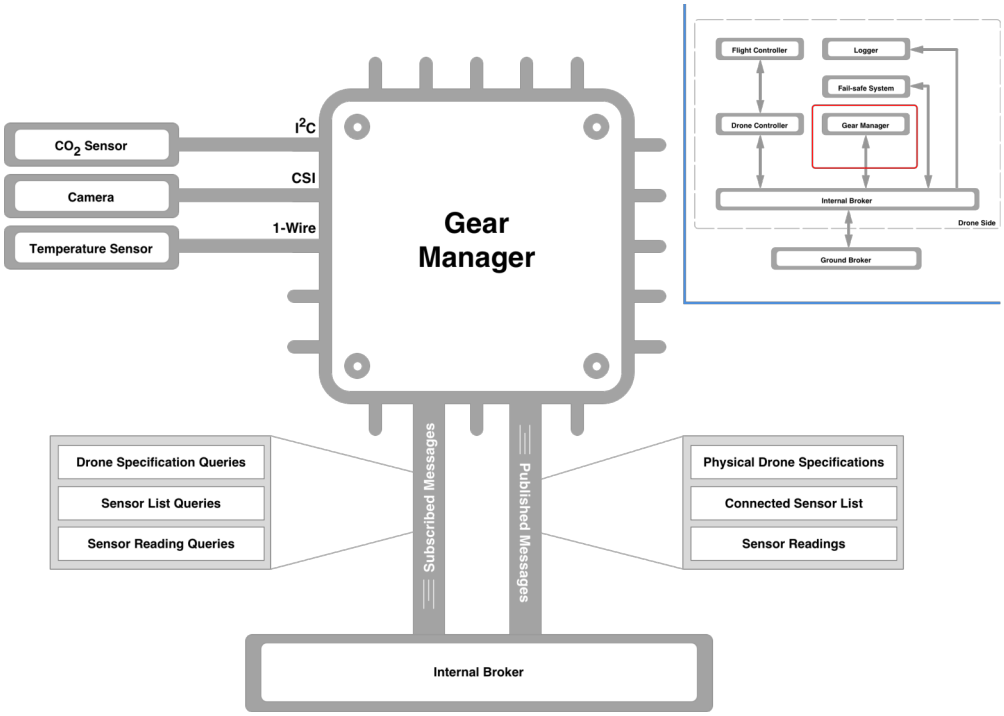

Figure 3.5: Gear Manager featuring sample sensors with commonly used interfaces.

The **Gear Manager**, depicted in Figure 3.5, is also responsible for keeping track of which sensors are connected to the drone at any given time, and must be able to provide

this information whenever it is requested. Along with the information regarding currently connected sensors, the **Gear Manager** must also possess the knowledge about physical characteristics of the drone, such as its type (e.g. quadcopter, hexacopter), frame size, battery capacity and cell count.

3.3.2.4 Fail-safe System

In order to implement the functionality of autonomous drone replacement, the architecture must feature a component that continuously analyzes drone telemetry.

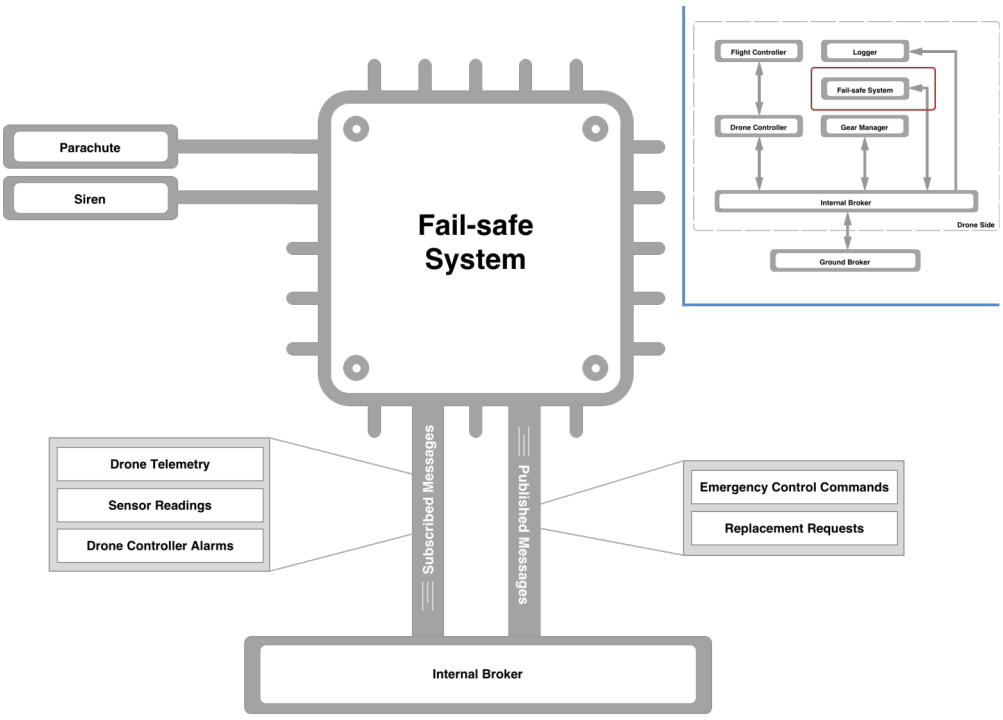

Figure 3.6: Fail-safe System featuring sample fail-safe mechanisms.

The **Fail-safe System** is a component that carries out this analysis with the goal of detecting behavior anomalies or sensor readings that indicate dangerous situations, such as a low remaining battery capacity. Detecting this kind of events gives in-flight drones the ability to automatically request a replacement or, upon critical situations such as an imminent crash due to a physical failure, to deploy on-board fail-safe mechanisms such as a parachute to slow down the descent of the drone and a loud audible alarm to alert nearby people.

3.3.2.5 Logger

The **Logger** is a trivial architecture component which keeps a connection to the internal broker and records every telemetry reading, control command or component interaction. With this in mind, the **Logger** does not publish any message back to the internal broker or perform any direct interaction with other architecture components.

3.3.3 Component connection

As shown in Figure 3.2, drone side architecture components accomplish their specified interactions by means of the **Internal Broker**, with the **Flight Controller** being the only exception to this generalization. The reason for this difference is explained in the **Flight Controller Interface** submodule description in section 3.3.2.2. In contrast to the **Flight Controller**, all the other described components rely on software implementations which can be deployed on an Single-Board Computer (SBC). With this in mind, the **Internal Broker** can consist of any lightweight implementation of a message broker which can be deployed on an SBC with diminished resources, to which each drone side component connects.

3.4 Ground side

In contrast to section 3.3, this section describes the platform architecture on the ground side, containing all components which must exist in a ground-located datacenter which is part of the platform. Initially, an overview of these components is presented, followed by a brief description of each of them, along with details regarding their connections.

3.4.1 Overview

Taking into consideration the scenarios described in section 3.1, the ground side architecture depicted in Figure 3.7 is established.

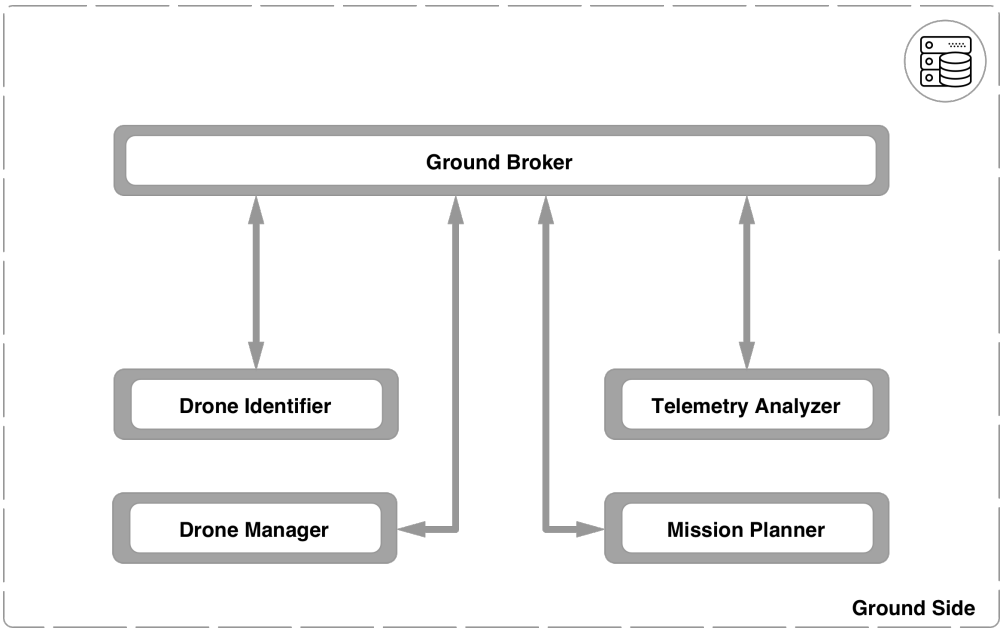

Figure 3.7: Overview of the ground side architecture

The purpose and description of each component is thoroughly described in section 3.4.2, while details regarding component connections are described in section 3.4.3.

3.4.2 Component description

This section describes the motivation for developing each of the proposed drone side architecture components, along with their functionality.

3.4.2.1 Drone Identifier

With the purpose of keeping track of which drones are currently connected to the platform, the **Drone Identifier** was developed. This component keeps a connection with the message broker deployed on the ground side and listens for special messages named heartbeats, which originate from each connected drone. These special messages contain information which can be useful for either the user or other components of the architecture, including a unique drone identifier, current geographic coordinates and a timestamp of when a drone was last seen. The **Drone Identifier** also keeps detailed information about the physical specifications and gear that is on-board every drone, which is transmitted by the **Gear Manager** of each drone.

3.4.2.2 Drone Manager

Although the abstraction of drone control is effectively accomplished in the drone side, a method for issuing high-level commands must be made available to the user. With this in mind, the **Drone Manager** was included on the ground side of the architecture. The **Drone Manager** is a component which serves a REST endpoint for high-level drone control. The existence of this endpoint allows for a high degree of extensibility since it enables any device with Internet access to achieve drone control through text-based commands. The **Drone Manager** additionally features a simple web application which provides the user with a graphical interface for controlling a drone.

3.4.2.3 Telemetry Analyzer

Telemetry acquisition scenarios described in section 3.1.1 and section 3.1.2 require architecture components which are capable of storing telemetry data. The storage of this data enables a user to later have access to flight details which may be important, for example, for debugging purposes or in case of a crash.

In order to be able to store this data for further access, however, this component requires a higher level of complexity, since it depends on submodules which must be able to both provide time series data storage and search over this type of data.

These modules are depicted in Figure 3.8 and are further explained.

- **Time Series Database**
  A database which is optimized for handling time series data. This database stores incoming telemetry from an arbitrary number of drones.
- **Broker-to-database Connector**
  In order to store telemetry in the database, a connector is developed with the purpose

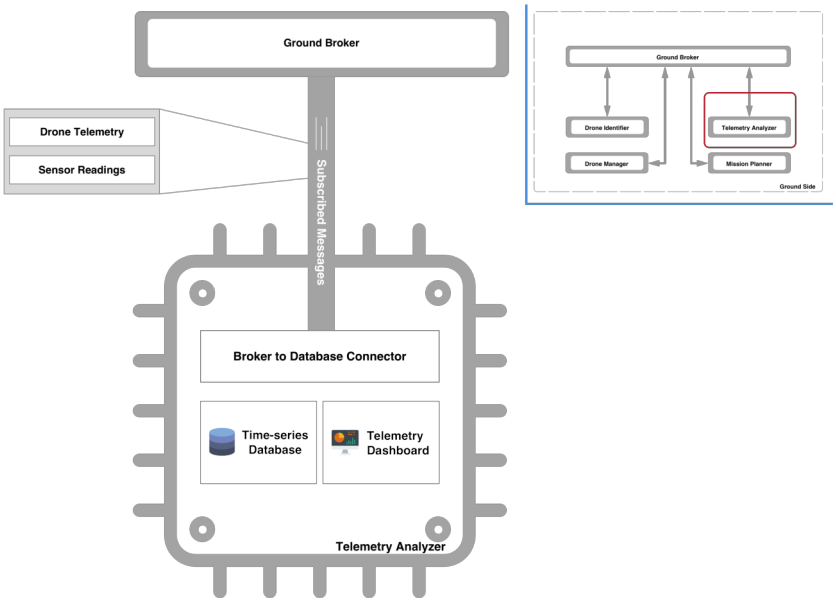

Figure 3.8: Telemetry Analyzer depiction

of continuously receiving drone telemetry from the **Ground Broker** and writing it to the **Time Series Database** on the fly.

- **Telemetry Dashboard**

  In order to attain the functionality requested in the scenarios described in section 3.1.1 and section 3.1.2, the **Telemetry Dashboard** provides the user with a graphical means of viewing the obtained drone telemetry. This should include not only real-time telemetry, but must also enable the user to access the obtained telemetry data of any requested time frame.

3.4.2.4 Mission Planner

Although the **Mission Worker**, located in the **Drone Controller** of each drone, is capable of parsing mission requests and executing mission steps, mission requests are normally generated on the ground side and are planned by a user. Replacement and collaboration requests are exceptions to this behavior, since these happen with no user interaction at all. The main purpose of the **Mission Planner** is the generation of mission requests that can be effectively parsed and executed by the **Mission Worker**. The **Mission Planner**, however, interacts with the **Drone Identifier** in order to obtain the list of drones which are connected to the platform, and is also in charge of keeping track of the mission progress of each drone. Knowing which drones are connected to the platform at a given time proves useful when a drone needs to be replaced or a collaborative mission should occur. In order to simplify the process of planning a mission, the **Mission Planner** also serves a web application which enables users to graphically prepare and send the chain of commands.

3.4.3 Component connection

In a similar way to the drone side, ground side architecture components achieve their interactions by the use of a message broker. This broker is referenced in the architecture description as the **Ground Broker**. Components that belong on this side of the architecture are to be deployed on a ground-located datacenter, which means resource availability should not be as limited as when deploying components in SBCs. Taking this into account, a wider range of message brokers which support advanced features such as clustering, plugin development and web management user interfaces can be considered for deployment on the ground side.

3.5 Communication and interaction

In order to enable communication between drones and ground architecture components, the **Internal Broker** of each drone and the **Ground Broker** share a connection through which broker messages can be relayed. In such manner, drones may use the **Internal Broker** not only for communication among internal components, but also when interaction with the ground is required. Likewise, this connection between the **Internal Broker** on board each drone and the **Ground Broker** allows ground side architecture components to interact, not only among themselves, but also with drones connected to the platform. In order to make this possible, message brokers should be configured so that messages which are published to specific topics can be automatically bridged to a message broker that is external to the current system. This represents, for drones, the relaying of messages to the ground side. For the ground side, this consists in relaying messages to one or more drones.

3.6 Multi-drone capabilities

Although the proposed platform enables the monitoring and control of an arbitrary number of drones in various scenarios (e.g. geographic coordinate based control, mission execution), one of its main ambitions is to enable the execution of missions in which more that one drone take part simultaneously. The proposed platform implements this type of functionality when adding support for handling self-replacement and mission collaboration situations, already described in the **Collaboration** scenario present in section 3.1.5. Drone autonomy, however, can further be enhanced by implementing drone-to-drone communication mechanisms which do not necessarily require a connection with the ground. This potential for improvement allows room for further research and is later proposed for future work in chapter 6.

3.7 Summary

In this chapter, we proposed an architecture for monitoring and controlling multiple drones, focused on covering scenarios such as telemetry and sensor data acquisition, geographic coordinate based drone control, mission execution and multi-drone collaboration.
This solution consists of multiple software and hardware modules which can either be deployed

on the drone side or on the ground side. These are loosely coupled and achieve their interaction through the usage of message brokers. Upon implementation, the proposed architecture aims to effectively abstract drone control from the user, while providing a set of graphical tools which enable a user to monitor and interact with connected drones.

CHAPTER 4

Implementation

This chapter describes the implementation of the proposed control platform. Initially, the implemented drone system is detailed, including both physical components of the drone and components which rely on software implementations. A description of each ground side component is also provided. In the final section, the implemented communication mechanisms are detailed, covering the interaction between drone and ground components, as well as the interfaces which are provided to platform users.

4.1 Drone side component description

This section contains the description of every physical or software component used in the implementation of each drone system.

4.1.1 Physical components

This subsection details the chosen drone type, along with the physical constituents of the drone system.

4.1.1.1 Drone type

Multi-rotors are proposed as the drone type to be used in the platform implementation. This drone type is of particular interest since, unlike fixed-wing drones, hovering still above a desired location is possible. This functionality is required as described in the **Geographic coordinate based drone control** requirement in section 3.1.3. Also, in order to allow for carrying external equipment such as sensors or an SBC, hexacopters have been chosen as the preferred type of multi-rotor, since they represent an adequate compromise between price and performance, allowing for higher stability and payload capacity when compared to quadcopters, while keeping part costs inferior to those of octocopters. A brief payload capacity comparison between a quadcopter and a hexacopter of the same manufacturer series

	Frame model	DJI F450 (quadcopter)	DJI F550 (hexacopter)
	Maximum recommended weight	1600g	2400g
Baseline drone system	**Frame**	282g	478g
	Electronic Speed Controllers	172g	258g
	Motors	240g	360g
	Propellers	52g	78g
	Battery	488g	488g
	Flight Controller	9g	9g
	GPS Module	32g	32g
	Radio Receiver	15g	15g
	Battery Eliminator Circuit	32g	32g
Payload	**Camera (Go Pro 3)**	77g	77g
	Single-board Computer (Raspberry Pi)	60g	60g
	3G Modem	40g	40g
	Arming Switch Emulator (Arduino Nano)	7g	7g
	Total weight	1506g	1934g
	Remaining payload capacity	94g	466g

Table 4.1: Payload capacity comparison between a quadcopter and a hexacopter of the DJI Flamewheel series.

is shown in Table 4.1. This, however, is not an imposed limitation to the supported multi-rotor type. Any type of multi-rotor can be used in the platform, as long as its **Flight Controller** provides support to it.

4.1.1.2 Component description

In order to implement a multi-rotor with the previously described characteristics that can be integrated in the proposed platform, the following physical components are used.

- **Flight Controller**

  The OpenPilot Revolution was selected as the **Flight Controller** to use, its representation in shown in Figure 4.1.

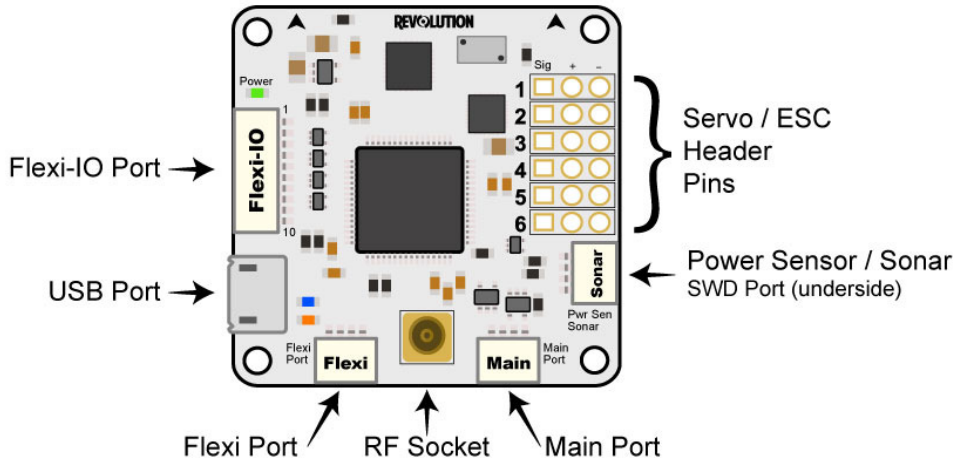

Figure 4.1: OpenPilot Revolution representation with connectivity highlights [56].

This **Flight Controller** contains all sensors required for autonomous flight, including

support for external GPS and magnetometer connection via the Main Port and the Flexi Port, respectively. This board also includes support for connecting voltage and current sensors and for external control using a Virtual COM Port (VCP) emulated using the on-board USB port, while keeping an affordable price of approximately 50€. The **Flight Controller** is flashed with the latest version of the dRonin [70] firmware, which is codenamed *Wired*, at the time of writing. This open-source flight controller firmware is of particular interest since it fully supports OpenPilot Revolution, implements basic navigation capabilities and provides a Python API for interfacing with it using a known protocol for interfacing with flight controllers, UAVTalk [71].

- **Frame**

  For the main structure of the drone, the DJI Flamewheel F550 frame kit [89] is used. This frame kit consists of six arms of equal length and two central pieces which act as a power distribution board, providing power connections for the drone battery and all six ESCs. A depiction of this frame kit before its assembly is shown in Figure 4.2.

Figure 4.2: DJI Flamewheel F550 frame kit before assembly [89].

- **Electronic Speed Controllers**

  The proposed drone system includes six DJI 420S ESCs [90], depicted in Figure 4.3, which are part of the DJI E310 Tuned Propulsion System [91].

- **Motors and Propellers**

  The proposed drone system includes six DJI 2312 motors [93] and six DJI 9450 self-tightening propellers [94], depicted in Figure 4.4 and Figure 4.5, respectively, which are part of the DJI E310 Tuned Propulsion System.

- **GPS and External Magnetometer**

  Drones feature an external module which contains both a u-blox NEO-M8N GPS receiver

Figure 4.3: Single DJI 420S electronic speed controller [92].

Figure 4.4: Six DJI 2312 motors [95].

Figure 4.5: A pair of clockwise and counter-clockwise rotation DJI 9450 propellers [96].

[97] and a Honeywell HMC5883L triple-axis magnetometer [98], which is depicted in Figure 4.6.

Figure 4.6: GPS and magnetometer module [99].

■ **Radio Receiver and Transmitter**

In order to enable manual control of the drone, a 2.4GHz radio control system is used, which is composed of the FLYSKY FS-i6 transmitter [100], shown in Figure 4.7, and the FLYSKY FS-iA6B receiver [101], shown in Figure 4.8. During the autonomous flight operations supported by this platform, the usage of the radio controller is not required, but it is always present during every platform test, enabling one to take over the drone if it exhibits an unexpected behavior.

Figure 4.7: FLYSKY FS-i6 radio transmitter [100].

Figure 4.8: FLYSKY FS-iA6B radio receiver [101].

- **Battery**

  For powering the electronic speed controllers of the drones, four-cell Lithium Polymer (LiPo) batteries with a capacity of 5500mAh are used. Following the specification sheet [102] of the used ESCs, these can be driven with up to a maximum voltage of 17.4V, meaning that they can be directly connected to the battery. Other components, such as the **Flight Controller**, **Radio Receiver**, **SBC**, **GPS and External Magnetometer**, however, must be powered using relatively low voltages (5V). Therefore, these devices are powered by a simple voltage regulator named Battery Eliminator Circuit (BEC), which steps down the voltage at the battery terminals to 5V.

- **Single-board Computer**

  Being the most widely used SBC, the Raspberry Pi 2 [103], depicted in Figure 4.9, was chosen for deploying the software components of the architecture. It runs Raspbian [104], the operating system which is officially supported by the Raspberry Pi Foundation. When on-board the drone, the Raspberry Pi is connected to a 3G modem that provides Internet access.

Figure 4.9: Raspberry Pi 2 Model B [103].

- **Arming Switch Emulator**

  In order to place the drone in the *armed* or *disarmed* state without the need of using the **Radio Transmitter**, a device is required to send the arming or disarming sequence to the **Flight Controller**. An Arduino Nano [105], shown in Figure 4.10, is used for this purpose. This process is described with greater detail in section 4.1.2.4.

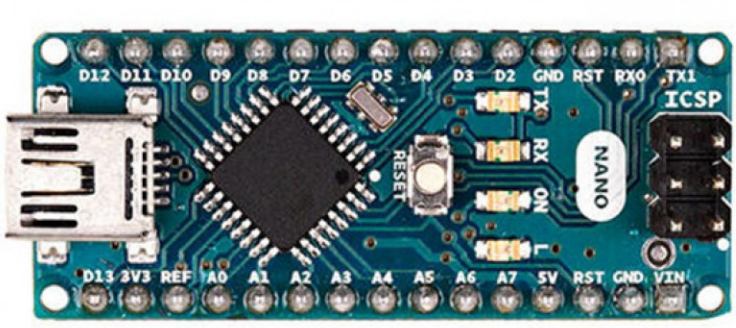

Figure 4.10: Arduino Nano 3 [105].

4.1.1.3 Component connections

This section depicts the physical component connections, shown in Figure 4.11, along with the key for the component connection types, shown in Figure 4.12.

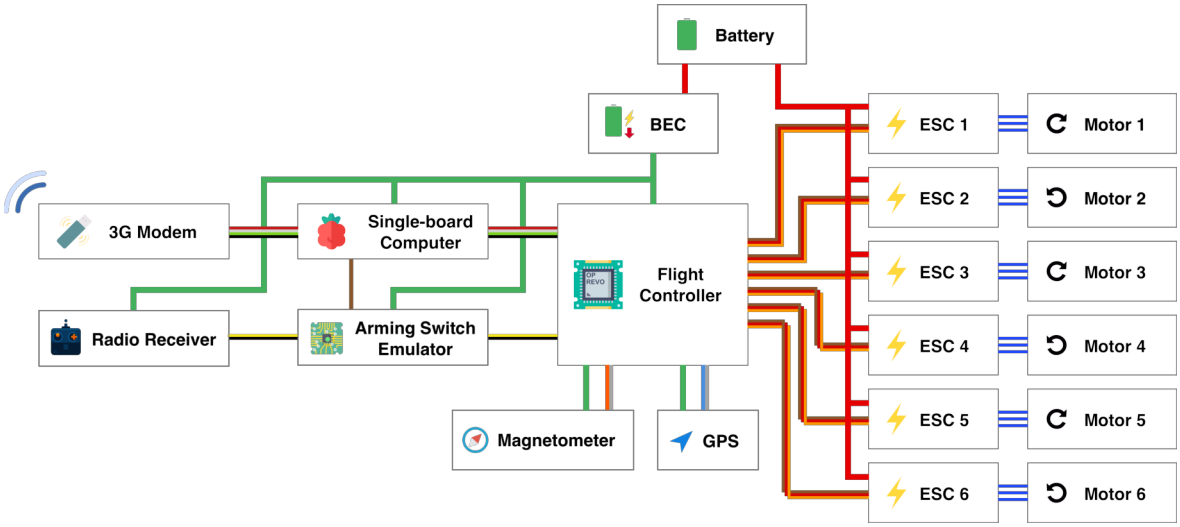

Figure 4.11: Drone physical component connection diagram.

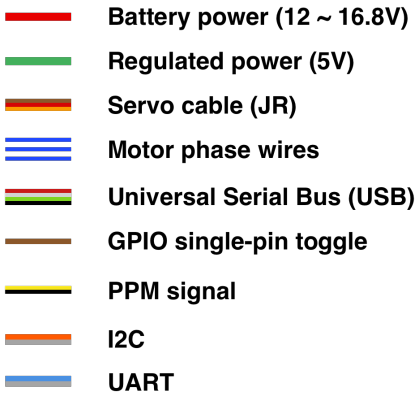

Figure 4.12: Key for physical component connection types.

4.1.2 Drone Controller

Developed as a Python 3 application, the **Drone Controller** is deployed on the Raspberry Pi. It is the most complex software component located on the drone side, and is comprised of various submodules. The **Drone Controller** is automatically launched upon powering on the Raspberry Pi, when the battery is connected. When this component is launched, it automatically establishes a connection with the **Flight Controller** using the USB port. Subsequently, a connection with the **Internal Broker** is also established. Once all connections are established, the **Drone Controller** will begin transmitting heartbeats every 5 seconds. These heartbeats contain a unique drone identifier which can be assigned by the user. If no identifier is assigned by the user, one will automatically be generated using the hexadecimal representation of the last 3 bytes of the Media Access Control (MAC) address of the network adapter embedded in the Raspberry Pi. Heartbeats also contain a timestamp and a GPS coordinate of the actual geographic location of the drone.

4.1.2.1 Flight Controller Interface

The **Flight Controller Interface** is responsible for establishing direct, bi-directional communication with the **Flight Controller**, handling incoming flight data and crafting control commands. This is the first submodule that is launched upon the startup of **Drone Controller**. If a physical **Flight Controller** connection is detected, this submodule will establish a session with it and ensure that no other session is created. Methods for acquiring and sending data objects to the **Flight Controller** are then made available by this submodule. As mentioned earlier, communication with the **Flight Controller** is achieved using UAVTalk, an open binary protocol used among various flight controller firmware such as OpenPilot [55], LibrePilot [106], TauLabs [107] and dRonin [70]. This protocol allows sending and receiving control and telemetry data to and from the **Flight Controller** using data containers known as **UAVObjects**. These objects can contain a variety of data types and fields which depend on the object name. For example, the **UAVO_Accels** object will contain three float values, each one with a reading for each of the accelerometer axes. Another object of interest is the **UAVO_Waypoint** object, to which can be written a set of NED coordinates that the drone will try to achieve. Names and definitions for each object are loaded at boot time, using an eXtensible Markup Language (XML) file for each object [108]. The **Flight Controller** automatically updates the values contained in each object periodically, with a frequency that depends on the name of the object. For example, the **UAVO_GPSPosition** object, which contains the actual GPS coordinate of the drone, is updated once every two seconds, while the **UAVO_AttitudeActual** object, which contains the attitude of the drone expressed in roll, pitch and yaw degree values, is updated ten times per second.

4.1.2.2 Navigation Processor

The **Navigation Processor** is a submodule which is essential for coordinate-based flight. It is responsible for parsing navigation requests that either reach the **Internal Broker**

when a command is sent from the ground side, or are part of a mission that is currently in progress. Upon parsing the navigation request, the **Navigation Processor** generates **UAVO_Waypoint** objects that are then sent to the **Flight Controller** by means of the **Flight Controller Interface**. Due to a firmware limitation, waypoints sent to dRonin must always be first converted to the NED coordinate system. In order to obtain coordinates in this system from an LLA coordinate, a method for coordinate conversion is adapted from the dRonin GCS source code [109]. It is shown in Figure 4.13.

DegreeToRadian = 0.017453293
EarthRadius = 6378137 meters

$$N = (\text{HomeAltitude} + \text{EarthRadius}) * (\text{TargetLatitude} - \text{HomeLatitude}) * \text{DegreeToRadian}$$

$$E = cos(\text{HomeLatitude} * \text{DegreeToRadian}) * (\text{HomeAltitude} + \text{EarthRadius}) * (\text{TargetLongitude} - \text{HomeLongitude}) * \text{DegreeToRadian}$$

$$D = \text{HomeLatitude} - \text{TargetAltitude}$$

Figure 4.13: LLA to NED coordinate conversion

The **Navigation Processor** is also capable of issuing landing and take-off requests to the **Flight Controller**, which are essential when performing autonomous missions.

4.1.2.3 Mission Worker

When missions are placed in the **Internal Broker**, the **Mission Worker** fetches these messages and proceeds to extract its mission steps. A mission is transmitted in a single message which contains all steps, and is encoded using the base64 encoding scheme by the **Mission Planner** on the ground side. In order to interpret the mission, the **Mission Worker** must first decode the received message. Then, an internal task queue, shown in Figure 4.14, is filled with the extracted mission steps. Mission steps are executed individually, following

Figure 4.14: Internal task queue used for mission execution.

the order in which they enter the queue. The **Mission Worker** is responsible for fetching the next step and handing it over to the component which is responsible for its execution. When the **Mission Worker** acknowledges the execution of a mission step, this information is

communicated to the **Mission Planner** on the ground side, which is responsible for keeping track of the progress of every mission. Upon executing a navigation request, the user may want the following step to only be executed once the drone reaches its goal. For this purpose, the **Mission Worker** features a **WAIT_FOR_FLIGHT** command. To detect whether a drone has already reached its target, the position of the drone is periodically acquired and compared to the goal position.

Algorithm 1: Checking if a drone has reached its goal

```
VERTICAL_TOLERANCE = 2;
HORIZONTAL_TOLERANCE = 2;
read actual;
read goal;
vertical_error = goal.Down - actual.Down;
horizontal_diff.x = goal.North - actual.North;
horizontal_diff.y = goal.East - actual.East;
horizontal_error = magnitude(horizontal_diff.x, horizontal_diff.y);
vertical_criterion = abs(vertical_error) < VERTICAL_TOLERANCE;
horizontal_criterion = horizontal_error < HORIZONTAL_TOLERANCE;
return vertical_criterion and horizontal_criterion;
```

Using algorithm 1, the **WAIT_FOR_FLIGHT** command causes the **Mission Worker** to wait until the drone has reached the goal with a customizable vertical and horizontal tolerance. These tolerances can be represented as a cylinder with its center on the exact desired position, as shown in Figure 4.15. Once the drone is located inside the cylinder,

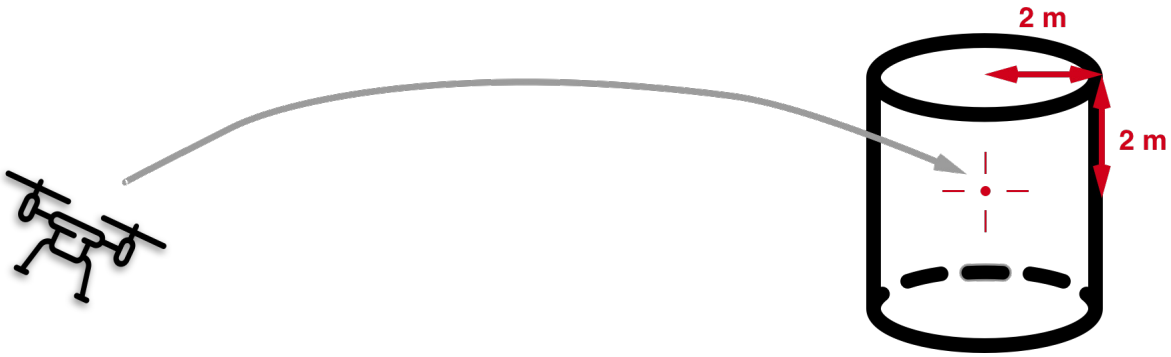

Figure 4.15: Waypoint tolerance representation.

the **WAIT_FOR_FLIGHT** command is marked as completed and the next command is fetched.

4.1.2.4 Arming Switch

The **Arming Switch** is responsible for keeping track of the arming state of the drone and is capable of changing it to either *armed* or *disarmed* upon request. As a firmware limitation, dRonin does not allow changing the arming state of the drone using requests sent by the

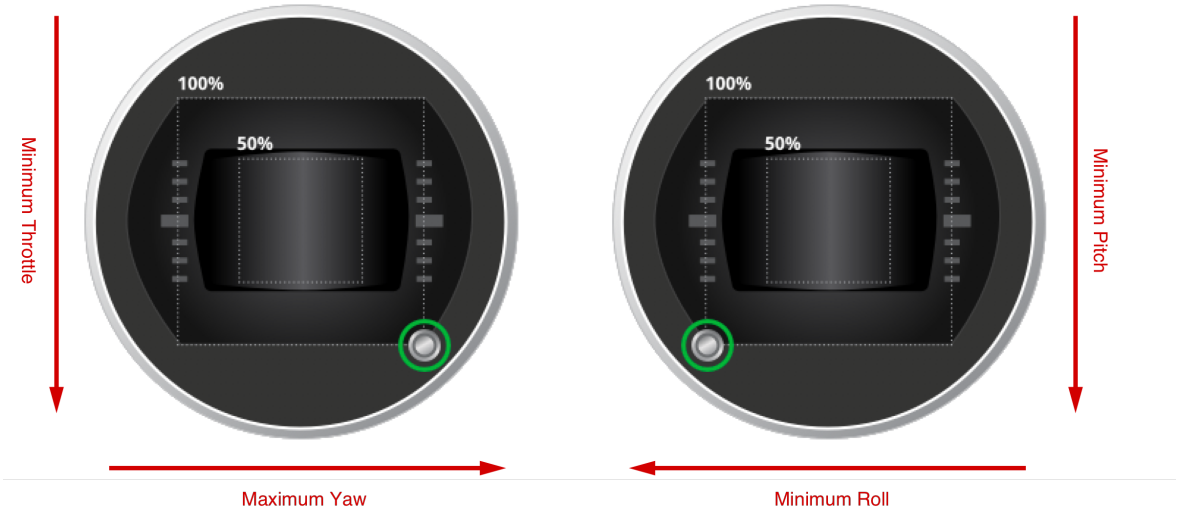

Figure 4.16: Sticks arming position representation.

USB port. In the typical, manual operation of the drone, this task is normally accomplished by placing the radio transmitter sticks in the arming position, as shown in Figure 4.16. For autonomous flight, however, this is not an acceptable method for performing arming state operations, since user interaction is required before and after every mission or flight in order to arm and disarm the drone. With this limitation in mind, the communication protocol used by the **Radio Receiver** to transmit stick position information to the **Flight Controller** was analyzed. The **Radio Receiver** transmits this information to the **Flight Controller** using a single pin output. When sticks are placed in their middle positions with zero throttle (shown in Figure 4.7), the flight controller receives the signal shown in Figure 4.17. Stick position information is separated in six channels – pitch, roll, throttle, yaw, one three-position switch and one two-position switch. This information is transmitted in this order, using inverted Pulse Period Modulation (PPM). Here, 20 millisecond frames are transmitted which contain the value of every channel.

As shown in Figure 4.18, the value for each channel is transmitted by keeping the signal line high for a specific duration. This duration may range from 500 microseconds, corresponding to the minimum value, up to 1500 microseconds, corresponding to the maximum value. Between the transmission of each of the channels, the line is kept low for a duration of 500 microseconds. After all channels are transmitted, the line is kept high for a variable duration, since the transmission of a new frame begins when 20 milliseconds have elapsed. In order to arm or disarm the drone, however, the **Flight Controller** must receive a frame which corresponds to the stick positions shown in Figure 4.16. This frame is shown in Figure 4.19.

With this in mind, the **Arming Switch** is complemented with the **Arming Switch Emulator**, shown in Figure 4.20, which consists of an Arduino Nano that is programmed to either forward the frame which is output by the **Radio Receiver** to the **Flight Controller** or, when requested using a General Purpose Input Output (GPIO) pin from the Raspberry Pi, send the frame required for arming and disarming instead. This GPIO pin is set to high or

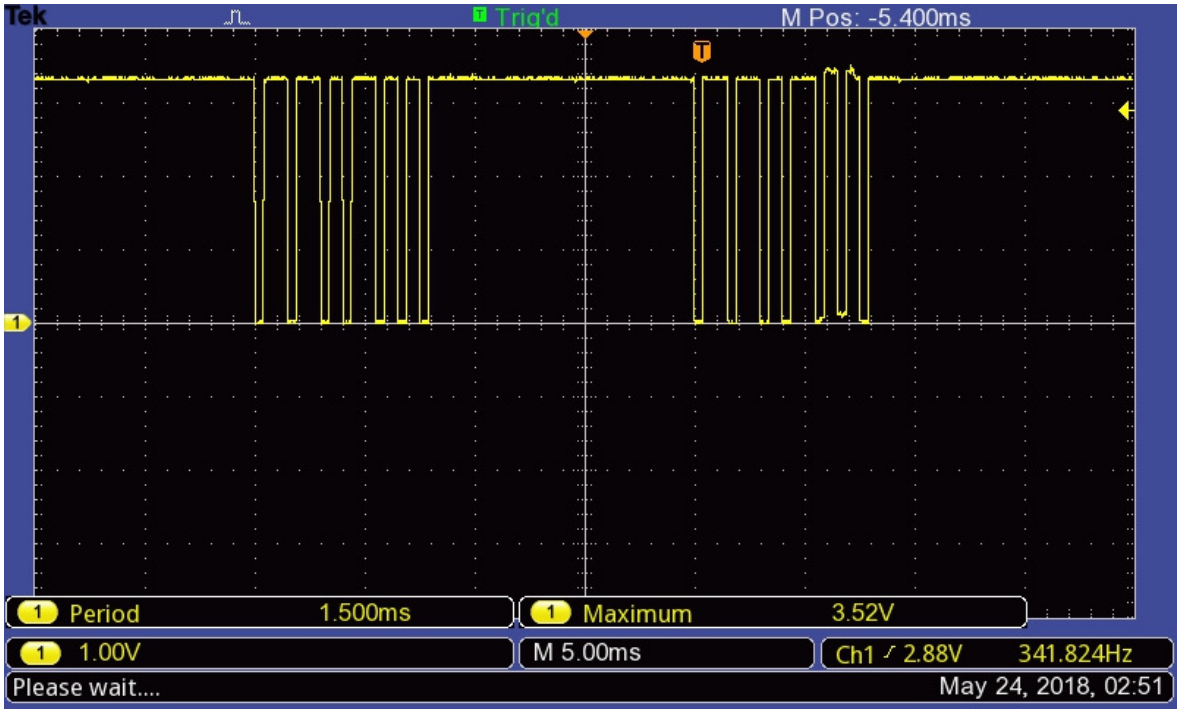

Figure 4.17: Signal received by the Flight Controller with middle pitch, middle roll, zero throttle and middle yaw.

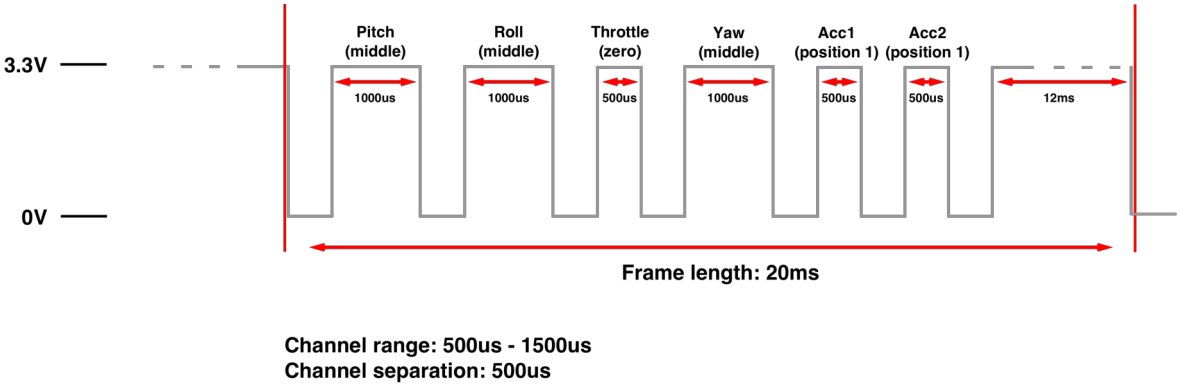

Figure 4.18: Radio Receiver signal diagram.

low using software, upon request of the **Arming Switch** submodule. Using this method, it is possible for the platform to arm or disarm a drone without the need of a **Radio Transmitter**, while also enabling one to be used during flight, in case a drone is not showing the expected behavior.

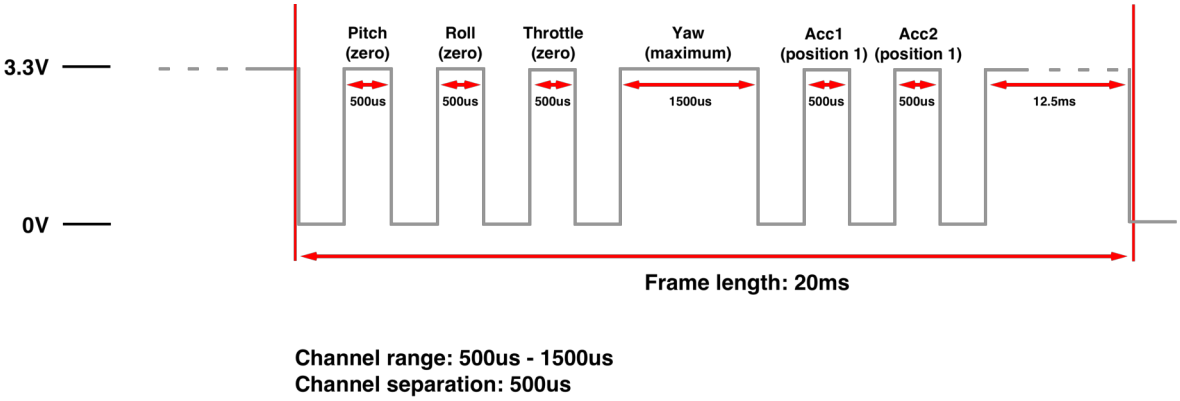

Figure 4.19: Frame required for arming and disarming.

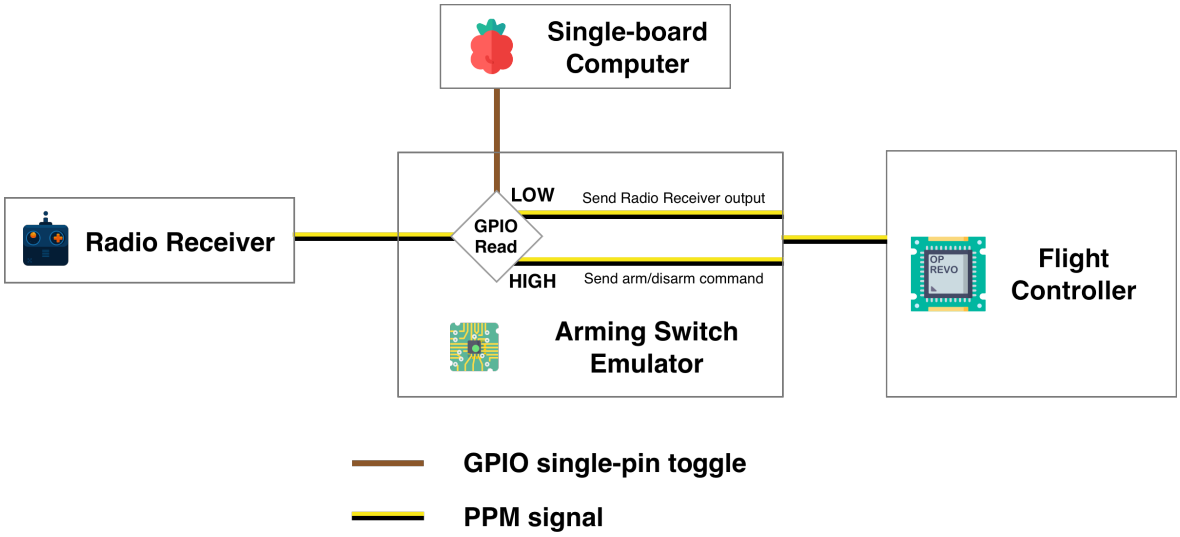

Figure 4.20: Arming Switch Emulator diagram.

4.1.3 Logger

Every **UAVObject**, when updated by the **Flight Controller**, is placed as telemetry in the **Internal Broker** by the **Drone Controller**. The **Logger**, also consisting of a Python 3 application, stores these objects in a text file for further examination, in the event of an unexpected behavior or a crash. The **Logger** also records events that are relative to the **Mission Worker**, including message decoding and processing timing measurements, details of each executed mission step, along with geographic coordinates and step execution duration. The acquisition of this data is critical to evaluate the procedure of each experiment documented in chapter 5. **Arming Switch** interactions are also recorded by the **Logger**, such as the exact time a drone was commanded to arm or disarm.

4.1.4 Gear Manager

Also relying in a Python 3 implementation, the **Gear Manager** is essential for providing access to sensors which are not directly compatible with the **Flight Controller**, as previously described in section 3.3.2.3. It interacts closely with the **Drone Controller** and is able

to provide readings from these sensors when requested. The **Gear Manager**, for example, automatically detects when a GoPro camera is available and handles each required step for establishing a connection to it, enabling users to capture frames, either manually or as a step in a mission. The **Gear Manager** is also responsible for keeping track of physical drone specifications and announcing what sensors are connected to it. Periodically, the **Gear Manager** parses JavaScript Object Notation (JSON) files from a customizable path, providing this information to ground-side components when requested.

Listing 4.1: Drone specifications file

```
{
  "TYPE" : "HEXACOPTER",
  "FRAME_SIZE" : "550",
  "BATTERY" : {
    "CELL_COUNT" : 4,
    "CAPACITY" : 5500
    }
}
```

Listing 4.2: Sensor specifications file

```
{
  "GAS" : {
    "CO2" : "Sensirion SCD30"
  },
  "CAMERA" : "GoPro Hero3"
}
```

4.2 Ground side component description

This section contains the description of every component used in the implementation of the ground system. All ground side components are deployed on Linux Containers (LXC) that run on a datacenter node. This node consists of a Dell PowerEdge R710 with the following specifications:

- **Processor:** Dual Intel Xeon X5670 @ 2.93GHz
- **Storage:** 6TB of hard disk storage
- **Memory:** 120GB DDR3
- **Networking:** Dual Gigabit Ethernet

4.2.1 Drone Identifier

The **Drone Identifier** is a simple component written in Python 3. It receives heartbeats, specifications and gear information of every drone that is connected to the platform and stores this information in an internal structure. Thus, this component always has the knowledge of which drones are available for usage, sharing it with other ground components whenever requested, via the **Ground Broker**. Additionally, the **Drone Identifier** makes this knowledge available to the user using the REST endpoint exposed via the **Drone Manager** by encoding the internal structure in a JSON object. An example of the internal data structure kept by the **Drone Identifier** is shown in Figure 4.21. Fields regarding drone specifications and connected sensors, however, may vary from drone to drone. In addition to sharing the connected drones list with other ground components, the **Drone Identifier** also features a web application which enables a user to monitor the connection status of a drone in real time.

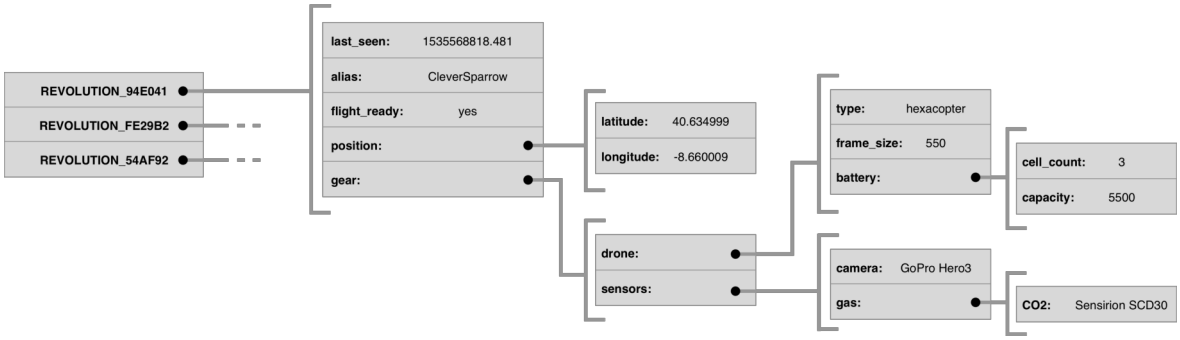

Figure 4.21: Example of the internal data structure kept by the **Drone Identifier**.

This application proves useful when performing field experiments with the drones. It is shown in Figure 4.22.

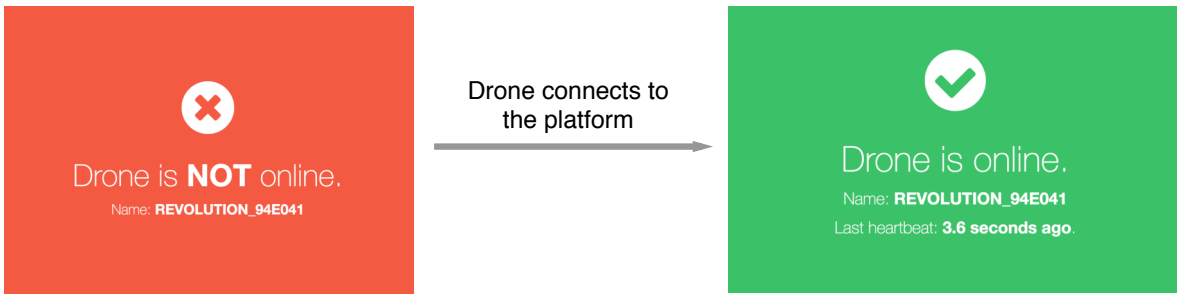

Figure 4.22: Front-end featured in the **Drone Identifier** for following the connection status of a drone.

4.2.2 Telemetry Analyzer

This component is responsible for storing and presenting telemetry data.

4.2.2.1 Time Series Database

In order to be able to store telemetry data, the **Telemetry Analyzer** features an installation of InfluxDB [110], an open-source time series database. Since received telemetry can always be indexed by time, the usage of a time series database can provide benefits over typical relational databases such as data summarization over long periods of time and data compression, in which older data is downsampled as its high precision becomes less relevant.

4.2.2.2 Broker-to-database Connector

To fill the **Time Series Database** with telemetry entries sent by the drone, a broker-to-database connector was developed. This connector, written in Python 3, subscribes to telemetry messages and, whenever one is available, it parses the message, extracts the **UAVObject** inside and stores its contents in the database.

4.2.2.3 Telemetry Dashboard

Once telemetry reaches the **Time Series Database**, a method must be provided to access it in a practical way. For this purpose, an instance of Grafana [111] is deployed as a submodule of the **Telemetry Analyzer**. Grafana is an open-source dashboard which allows the visualization of data stored in various supported databases. It supports using InfluxDB instances as data sources and allows users to select specific time frames for visualization. Stored data can be presented to the user in various forms such as graphs, tables, lists and heatmaps. In the **Telemetry Analyzer**, Grafana connects to the **Time Series Database**, where telemetry is stored, and makes it available to authorized users by means of a web page. Figure 4.23 shows the **Telemetry Dashboard** providing barometer, temperature, accelerometer, gyroscope, CPU load and remaining memory data.

Figure 4.23: Telemetry Dashboard screenshot.

The installation and setup of InfluxDB, Grafana and message brokers, along with the development of the **Broker-to-database connector**, was achieved in collaboration with Bruno Areias and is described in his work [112].

4.2.3 Drone Manager

The **Drone Manager** consists of a REST endpoint that enables high-level drone control available to users through Hypertext Transfer Protocol (HTTP) GET and POST requests. It also enables users to obtain a list of connected drones, their specifications and connected gear. Commands supported by the REST API are further detailed in Table 4.2. Example: a drone with the identifier **QUAD_REVO2** can be armed by executing a POST request to `https://<endpoint_address>/control` with parameters `**command**=arm&**droneID**=QUAD_REVO2`.

The **Drone Manager** also provides a responsive web interface to allow sending most high-level commands to a drone using a web browser on a computer or mobile device. This interface is shown in Figure 4.24. Further details regarding the communication method that

Method	Command	Additional Parameters	Description
GET	discover	None.	Routed to the **Drone Identifier**. Lists all connected drones.
GET	getgear	**droneID:** identifier of the target drone	Routed to the **Drone Identifier**. Fetches gear information of a drone.
GET	objrequest	**droneID:** identifier of the target drone **value:** name of the UAVObject to request	Routed to the **Drone Controller**. Requests a UAVObject from the drone.
POST	changeN	**droneID:** identifier of the target drone **value:** distance in meters	Routed to the **Drone Controller**. Commands the drone to move North by a specified number of meters.
POST	changeE	**droneID:** identifier of the target drone **value:** distance in meters	Routed to the **Drone Controller**. Commands the drone to move East by a specified number of meters.
POST	altitude	**droneID:** identifier of the target drone **value:** distance in meters	Routed to the **Drone Controller**. Commands the drone to climb a specified number of meters.
POST	navigate	**droneID:** identifier of the target drone **lat:** target latitude **lon:** target longitude **altitude_delta:** target altitude (optional)	Routed to the **Drone Controller**. Commands the drone to fly to a given coordinate.
POST	land	**droneID:** identifier of the target drone	Routed to the **Drone Controller**. Commands the drone to land.
POST	takeoff	**droneID:** identifier of the target drone	Routed to the **Drone Controller**. Commands the drone to take off.
POST	startchain	**droneID:** identifier of the target drone **mission:** mission steps encoded in base64	Routed to the **Mission Planner**. Commands the drone to execute a mission.
POST	stopchain	**droneID:** identifier of the target drone	Routed to the **Mission Planner**. Commands the drone to stop executing the current mission.
POST	arm	**droneID:** identifier of the target drone	Routed to the **Drone Controller**. Arms the drone.
POST	disarm	**droneID:** identifier of the target drone	Routed to the **Drone Controller**. Disarms the drone. Destructive during flight.
POST	takepic	**droneID:** identifier of the target drone	Routed to the **Drone Controller**. Commands the drone to take a picture at the current location, if a camera is present.
POST	resetfc	**droneID:** identifier of the target drone	Routed to the **Drone Controller**. Commands the drone to reboot its Flight Controller. Destructive during flight.
POST	rebuildtelemetry	**droneID:** identifier of the target drone	Routed to the **Drone Controller**. Commands the drone to re-establish the telemetry connection between the Flight Controller and the Drone Controller.

Table 4.2: Drone Manager REST API documentation.

this component uses to interact with other drone and ground components are provided in section 4.3.3. The **Drone Manager** features a back-end developed in NodeJS [113], chosen for its rapid prototyping potential and compatibility with the used message brokering solutions.

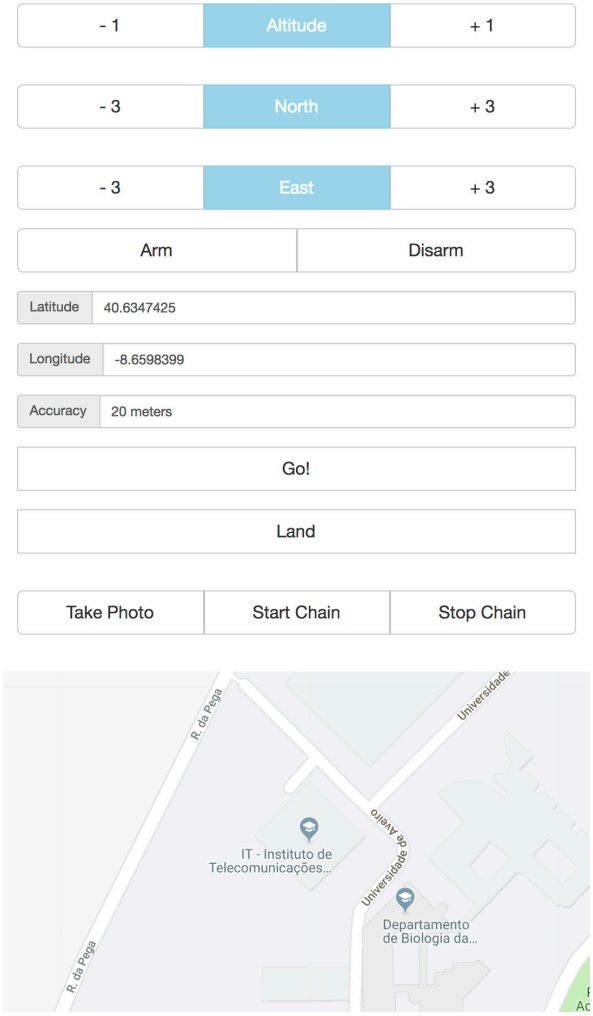

Figure 4.24: Drone Manager web interface screenshot.

4.2.4 Mission Planner

The **Mission Planner** allows the preparation of missions that can be parsed and executed by the **Mission Worker** of a drone. Like other previously described components, it is an application developed using Python 3. The **Mission Planner** keeps track of available drones through interaction with the **Drone Identifier**, along with the progress of every mission that is being executed, providing timing logs with this information. An example of the internal data structure in which mission data is kept is shown in Figure 4.25.

The **Mission Planner** features a front-end, developed using AngularJS [114], Bootstrap [115] and Leaflet [116], which allows a user to plan complex missions and choose their target drone. It is shown in Figure 4.26. Along with the commands described in Table 4.2, three additional commands can be used in mission environments. These are shown in Table 4.3. The **Mission Planner** provides a pragmatic mapping solution to automatically generate a path that covers a specified area. The execution of the mapping feature is shown in Figure 4.27. Missions with paths that are generated by this feature can be used in collaboration scenarios. The area coverage of these missions can be automatically extended when collaboration alarms

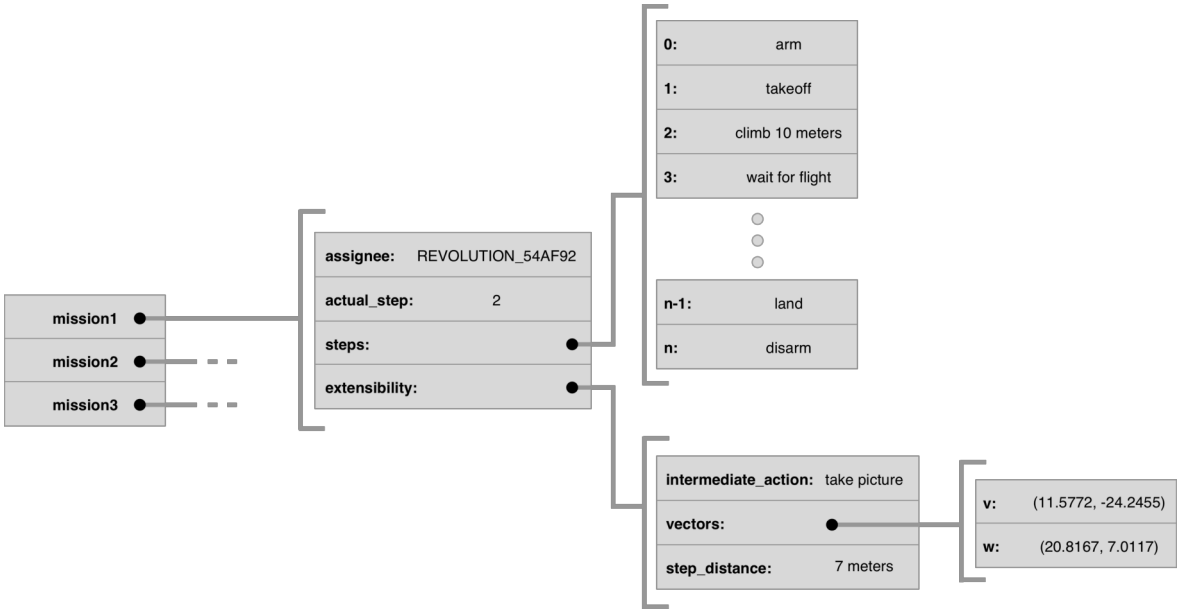

Figure 4.25: Example of the internal data structure kept by the **Mission Planner**.

Figure 4.26: Mission Planner web interface screenshot.

are launched. These missions include extensibility parameters which contain all the values required for the **Mission Planner** to rebuild the mission with the exact same shape, a new center and a higher area coverage, without the need for user intervention. As shown in Figure 4.25, three extensibility parameters are included: **intermediate_action**, **vectors** and **step_distance**.

- **intermediate_action** - Optional parameter. Specifies a mission step which is executed

Command	Additional Parameters	Description
wait_for_flight	None.	Causes the **Mission Worker** to wait until the drone has reached the previously requested position.
sleep	**value:** number of seconds to wait	Commands the drone to hover above the actual location for the specified number of seconds.
virtualalarm	**level:** alarm level (**CRITICAL** or **OUT_MEASURE**) **message:** text message which complements the alarm	Causes the drone to simulate an alarm. A **CRITICAL** alarm will cause an automatic replacement to occur. An **OUT_MEASURE** alarm will cause the mission to be collaboratively extended.

Table 4.3: Additional control commands supported in missions.

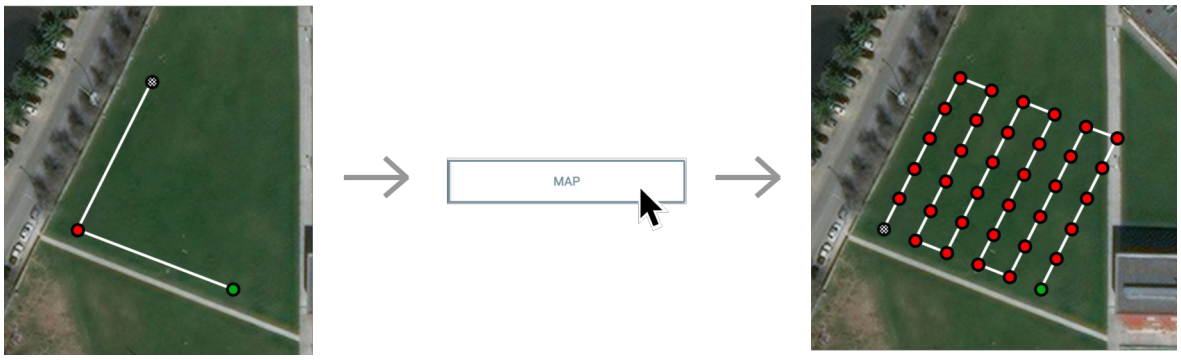

Figure 4.27: Mapping feature demonstration in the Mission Planner web interface.

upon reaching each waypoint.

- **vectors** - Contains vectors $\vec{v}$ and $\vec{w}$. Allows the **Mission Planner** back-end to recover the shape used in the mission and rebuild it with a higher area coverage and different center.
- **step_distance** - Specifies the distance between each waypoint.

A visual representation of **step_distance** and both vectors contained in **vectors** is provided in Figure 4.28. Upon extending a mission, the **Mission Planner** may also distribute equal parts of the new area among various drones. Additional drones are selected depending on their availability and distance to the center of the new mission, meaning that drones which are closer to the center will have a higher priority in the selection process. The same priority applies when selecting a drone for replacement, when a **CRITICAL** alarm is launched. When the last step of a mission is completed, the mission is marked as finished. At this moment, the **Mission Planner** generates a log file that enables a user to visualize the path taken by the drones that took part in such mission. To make this possible, a simple web application was developed, allowing the user to drag-and-drop the log file onto a web page. Once the user drops the log file on the page, the web application decodes it and parses the progress accomplished by the drones, displaying it in a map. An example of the usage of this tool after a mapping mission is completed is shown in Figure 4.29.

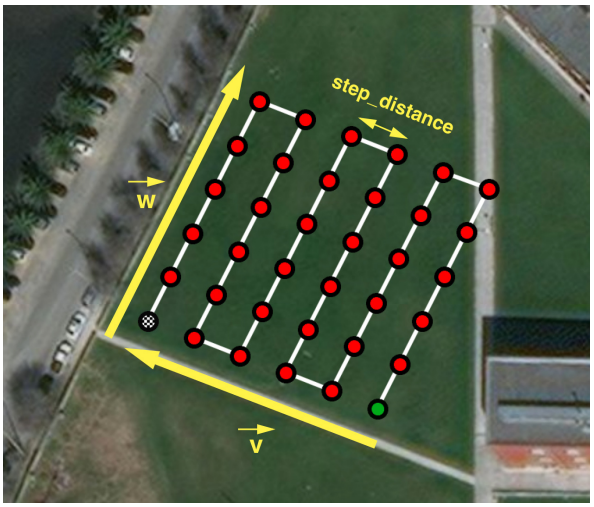

Figure 4.28: Extensibility parameter depiction.

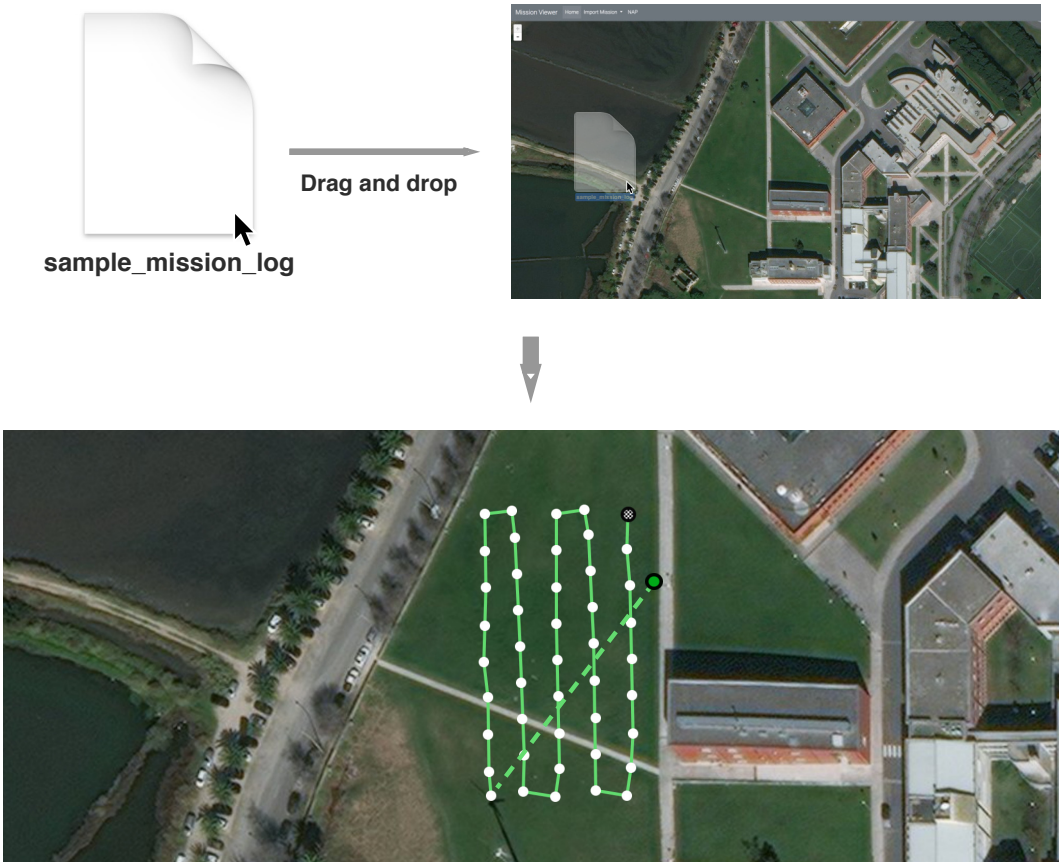

Figure 4.29: Mission viewing functionality demonstration.

4.3 Communication Mechanisms

This section describes the communication mechanisms used in the platform. It covers the brokering solutions used on both drone and ground sides, along with the message structure used in these interactions. It also describes the mechanisms that were implemented in order to allow a user to interact with drones using the web.

4.3.1 Message brokers

Message brokering solutions which were adopted for both the **Ground Broker** and the **Internal Broker** aboard every drone are described in this section.

4.3.1.1 Drone Internal Broker

Since the **Internal Broker** must be deployed on a Raspberry Pi, which consists of an SBC with limited resources, a lightweight message broker solution must be used. For this purpose, Mosquitto [117] was chosen. It is a project, created in 2014, which belongs to the Eclipse Foundation [118], which implements the Message Queuing Telemetry Transport (MQTT) [119] protocol. Mosquitto was chosen for being a very light, open-source message broker. In this platform, each drone contains an installation of Mosquitto in its Raspberry Pi. Thus, Mosquitto represents the **Internal Broker** of each drone. Drone side architecture components use the **Internal Broker** to achieve their interactions. Depending on the type of interaction, components publish and subscribe to different message topics. When using MQTT, a topic is represented by one or more topic levels separated by a forward slash (/). An example of this representation is shown in Figure 4.30. This example contains the topic to which the **Drone Controller** subscribes. Different topics are used among drone side components, as

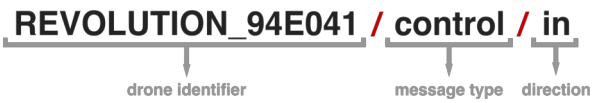

Figure 4.30: Topic representation when using the **Internal Broker**.

shown in Table 4.4. **REVOLUTION_94E041** is used as an example of a unique drone identifier. Whenever ground side components appear on this table, the bridging mechanisms described in section 4.3.2 are used. Since it subscribes to every topic and is only used for logging purposes, the **Logger** is omitted from this table.

Topic	Is Subscribed By	Receives Messages From	Message Content
REVOLUTION_94E041/control/in	**Drone Controller**	**Drone Manager**	Control commands
REVOLUTION_94E041/control/out	**Drone Manager**	**Drone Controller**	Responses to control commands
REVOLUTION_94E041/mission/in	**Drone Controller**	**Mission Planner**	Mission requests
REVOLUTION_94E041/mission/out	**Mission Planner**	**Drone Controller**	Mission progress information
REVOLUTION_94E041/gear/in	**Gear Manager**	**Drone Controller** **Drone Identifier**	Drone specification queries Sensor list queries Sensor reading queries
REVOLUTION_94E041/gear/out	**Drone Controller** **Drone Identifier**	**Gear Manager**	Drone specifications Connected sensors list Sensor readings
REVOLUTION_94E041/heartbeats/out	**Drone Identifier**	**Drone Controller**	Drone heartbeats
REVOLUTION_94E041/alarms/out	**Mission Planner**	**Drone Controller**	Replacement alarms Collaboration alarms
REVOLUTION_94E041/telemetry/out	**Telemetry Analyzer**	**Drone Controller**	Drone telemetry

Table 4.4: Topics used in interactions which make use of the **Internal Broker**.

4.3.1.2 Ground Broker

The **Ground Broker** is the communication medium used by components which are located on the ground side. Like other ground components, the **Ground Broker** is deployed on an LXC container which runs on the previously described datacenter node. Since the resource limitations which exist on the Raspberry Pi do not apply on the ground side, the usage of a lightweight message broker solution such as Mosquitto is not mandatory. With this in mind, RabbitMQ [120] was chosen. RabbitMQ is an open-source message broker which implements the Advanced Message Queuing Protocol (AMQP) [121]. Although not used in the actual development stage of the platform, RabbitMQ supports clustering, which enables for increased throughput and high availability by connecting multiple nodes together. RabbitMQ also supports the usage of plugins, such as a web accessible management interface which simplifies the configuration process and monitoring of the message broker. RabbitMQ provides compatibility with MQTT clients by means of an MQTT adapter – the usage of this mechanism is further described in section 4.3.2. In this platform, for interactions among ground components, each component that receives any type of message has its own dedicated exchange, and any other component that requires sending requests to it should use that exchange. Since RabbitMQ does not allow receiving messages directly from an exchange, a short lived queue which contains its messages is automatically generated for such component and is destroyed afterwards, once the connection is terminated. The process of a connected drones query, followed by its reply, is depicted in Figure 4.31 and Figure 4.32.

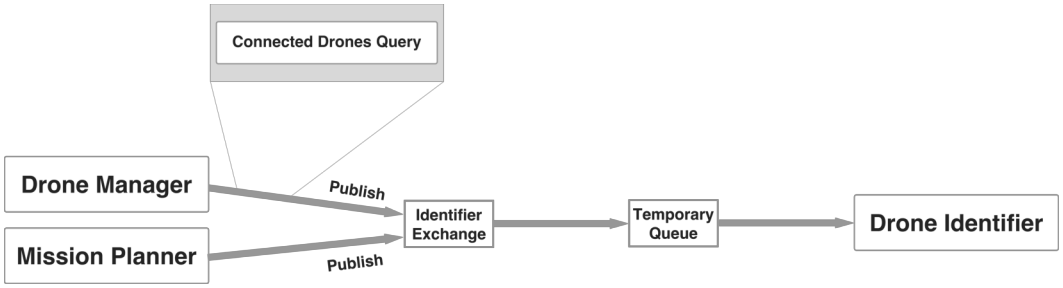

Figure 4.31: Example of a **Drone Manager** or **Mission Planner** query to the **Drone Identifier**.

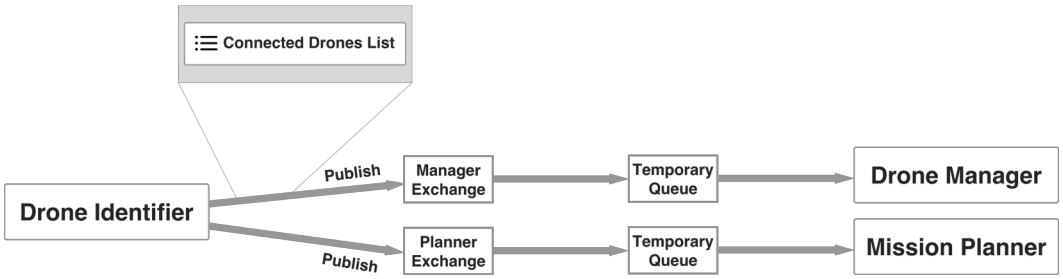

Figure 4.32: Example of a **Drone Identifier** response to the **Drone Manager** or **Mission Planner** query.

4.3.2 Communication between Drone and Ground

In order to allow interactions between drone and ground side components, the **Internal Brokers** and the **Ground Broker** share a connection that allows messages to be relayed from one broker to another. This is accomplished by using the bridging capabilities of Mosquitto and the MQTT adapter supported by RabbitMQ. Each **Internal Broker** instance in every drone is configured to connect to the **Ground Broker** once the Raspberry Pi boots. An example of the Mosquitto bridging configuration file is shown in Listing 4.3.

Listing 4.3: Mosquitto bridging configuration file example

```
connection rabbitRelay
address <GROUND_BROKER_ADDRESS>:1883

password dronePassword
username droneUsername
clientid droneMosquitto-REVOLUTION_94E041
topic REVOLUTION_94E041/control/in in "" ""
topic REVOLUTION_94E041/mission/in in "" ""
topic REVOLUTION_94E041/gear/in in "" ""
topic +/control/out out "" ""
topic +/mission/out out "" ""
topic +/gear/out out "" ""
topic +/heartbeats/out out "" ""
topic +/alarms/out out "" ""
topic +/telemetry/out out "" ""
```

In the presence of this configuration, an **Internal Broker** would request the **Ground Broker** to relay every message with the routing keys **REVOLUTION_94E041/control/in**, **REVOLUTION_94E041/mission/in** and **REVOLUTION_94E041/gear/in** to this drone. In turn, messages placed in the **Internal Broker** in topics which are marked **out** would be relayed to the **Ground Broker**. In these topics, one does not need to specify a drone identifier, since no messages with other drone identifiers would originate from this drone. The execution of a **UAVObject** request, followed by its response, is an interaction example which requires this bridging mechanism to take place. Its representation is shown in Figure 4.33.

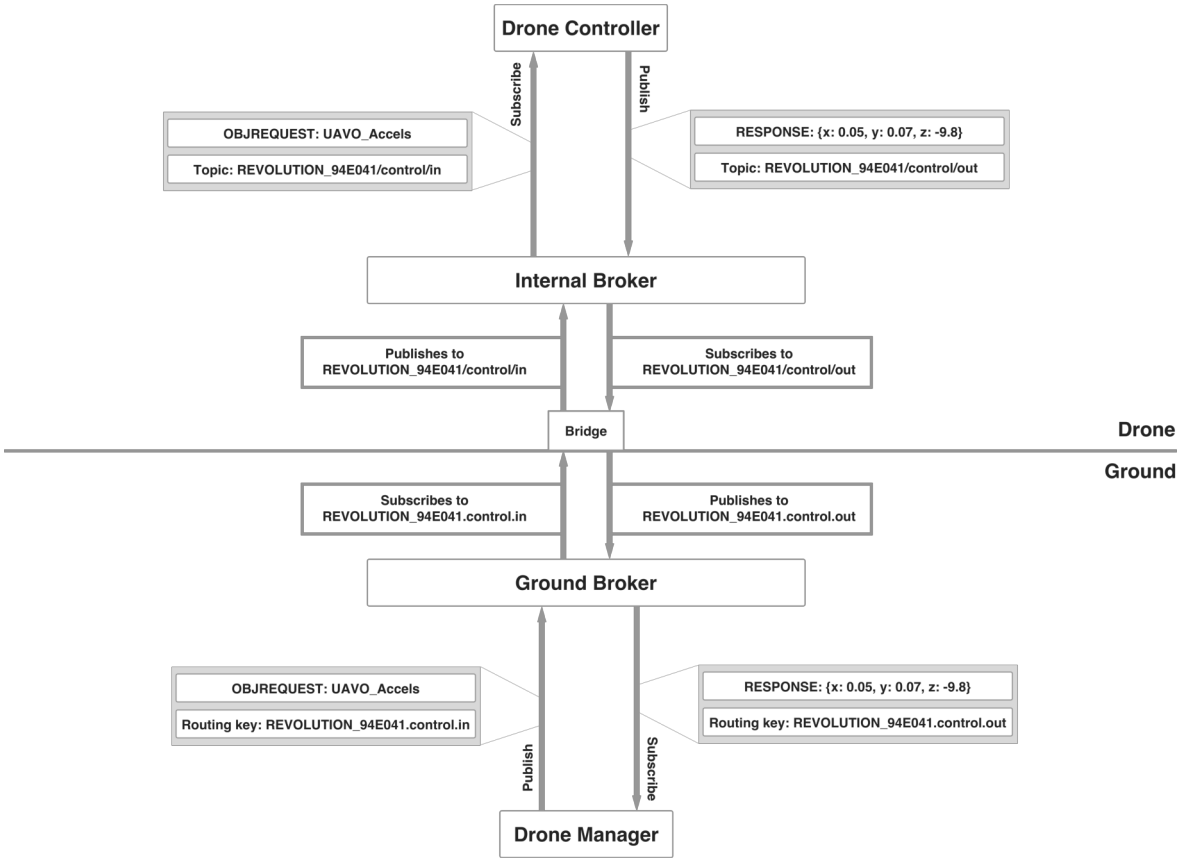

Figure 4.33: Interaction and bridging example of a **UAVObject** request initiated from the **Drone Manager** and followed by its response from the **Drone Controller**.

4.3.3 Communication between Ground and the user

The **Drone Manager** is the main component which allows web access to drones that are connected to the platform. In order to achieve this, the **Drone Manager** is able to translate HTTP requests into messages that are placed in the **Ground Broker**. When a user executes one of the commands described in Table 4.2 using the REST API, the **Drone Manager** generates a unique request identifier composed by 16 bytes of cryptographically strong pseudo-random data and includes it in the message that is placed in the **Ground Broker**. A response is only provided to the user once the component that is responsible for the requested command places a response message on the **Ground Broker** with the same identifier. An example of the full sequence of interactions in an arming request is provided in Figure 4.34. Components which feature web front-ends, such as the **Mission Planner**, also make use of the REST API to execute every request and thus also follow the interactions shown in Figure 4.34, with variations on the request and response message content.

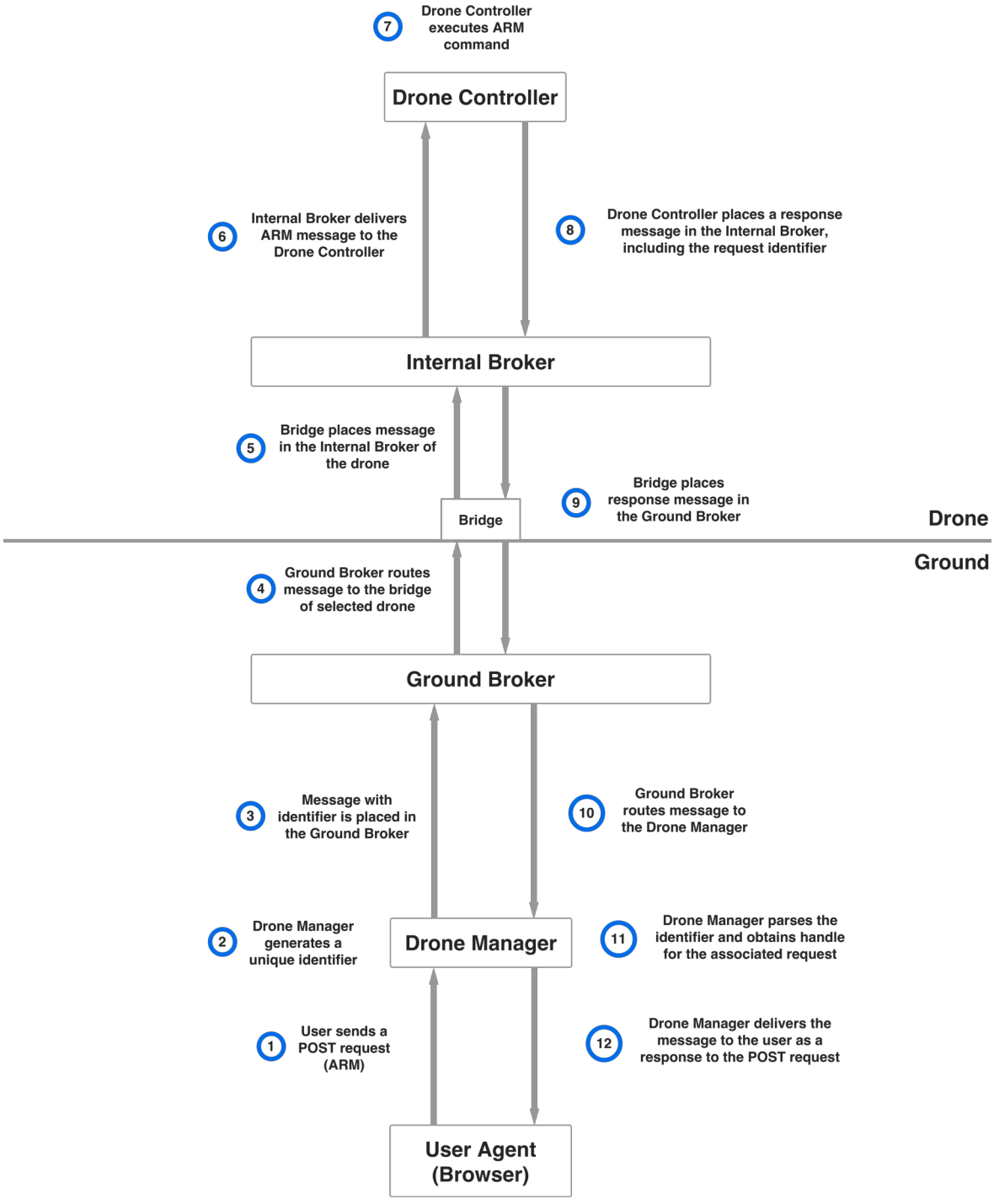

Figure 4.34: Interaction diagram of an arming request initiated by a user from the REST API and followed by its response from the **Drone Controller**.

4.3.4 Message structure

With exception to the messages used when interfacing directly with the **Flight Controller**, which must make use of the UAVTalk binary protocol, all exchanged messages are human-readable and transmitted in the JSON format. An example of the contents in the exchanged messages during an arming request initiated by a user is shown in Figure 4.35.

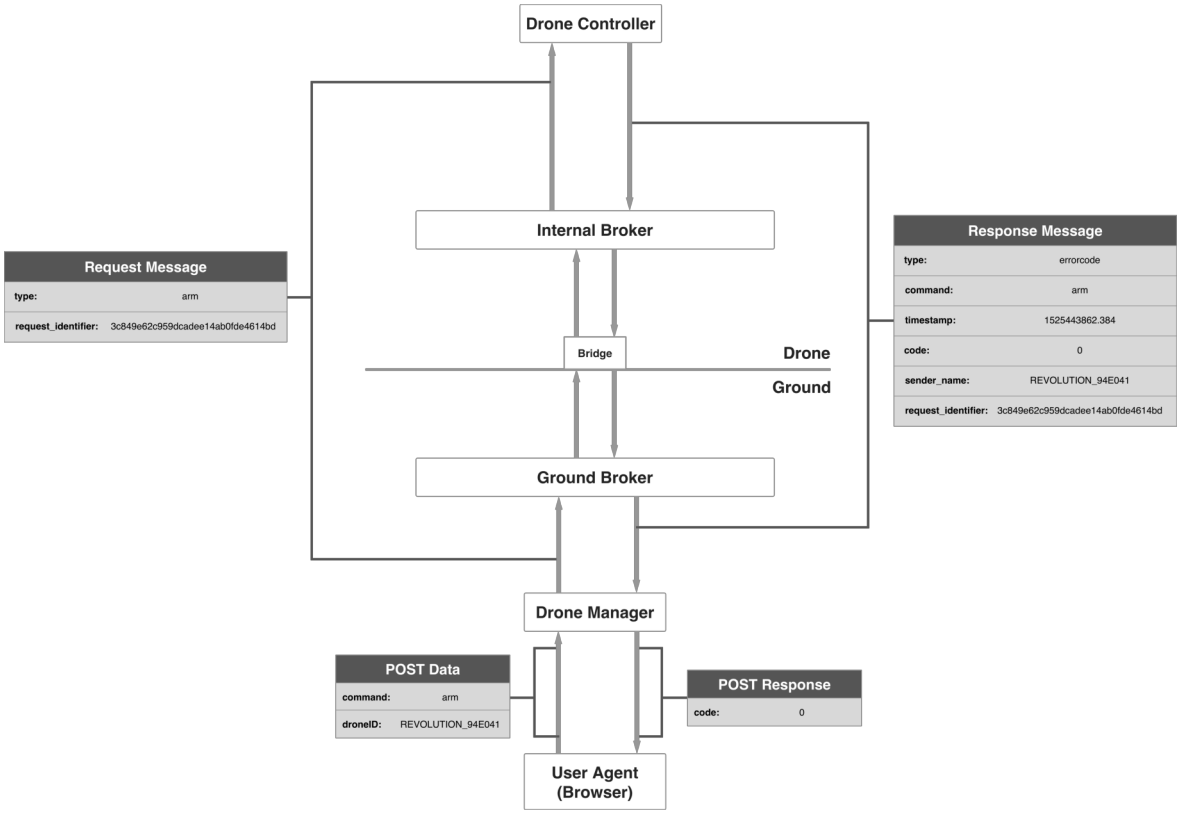

Figure 4.35: Example of the message contents inside exchanged messages during an arming request initiated by the user.

4.4 Summary

In this chapter, we described the implementation of a drone control platform architecture that fulfills the established requirements. This includes a proposal of the hardware components of a drone system along with their connections, from which resulted the assembly of three hexacopters. The end of this chapter also marks the successful development of every proposed software module that composes both the drone-side and the ground-side architecture, along with the communication mechanisms required for their interactions. The resulting platform is ready to support autonomous drone flight in tasks which may range from basic navigation commands to the automation of complex paths and collaboration of multiple drone systems. The developed REST API and web applications allow users to plan and execute these tasks through graphical interfaces, while also monitoring every drone through a telemetry dashboard.

CHAPTER 5

Experiments

This chapter describes the objectives, methodology, evaluation and results of four distinct experiments. The described experiments gradually increase in complexity, ranging from simple tests to the baseline capabilities of the infrastructure to more sophisticated tasks which require the automation of full paths and multiple drone collaborative missions. These experiments provide experimental validations of the implemented functionality.

5.1 Panic Button

5.1.1 Objectives

The implementation of a panic button represents the baseline experiment of the developed platform. Such an experiment allows testing the platform for the correct implementation of basic navigation capabilities. In this scenario, a user carries a device with geolocation capabilities which presents a simple button that, when pressed, calls an available drone to travel to the location of the user and hover above him at a fixed altitude. If successful, the experiment shows that not only both the drone-side control components and the server-side mission assignment components work as expected, but also that the communication mechanisms in place are also following the expected behavior.

5.1.2 Method

Although the panic button experiment seeks to be as simple and uncluttered as possible, it is of interest to make use of it to test the broadest range of functionalities in its context. With this in mind, the scenario was extended to support the automatic arming of the drone, along with its takeoff and climbing to a predefined altitude. This experiment makes use of a simple iOS application, developed for this purpose. This application presents the panic button to the user, who can use his smartphone as a geolocation-capable device which is able to communicate with the platform and provide location information.

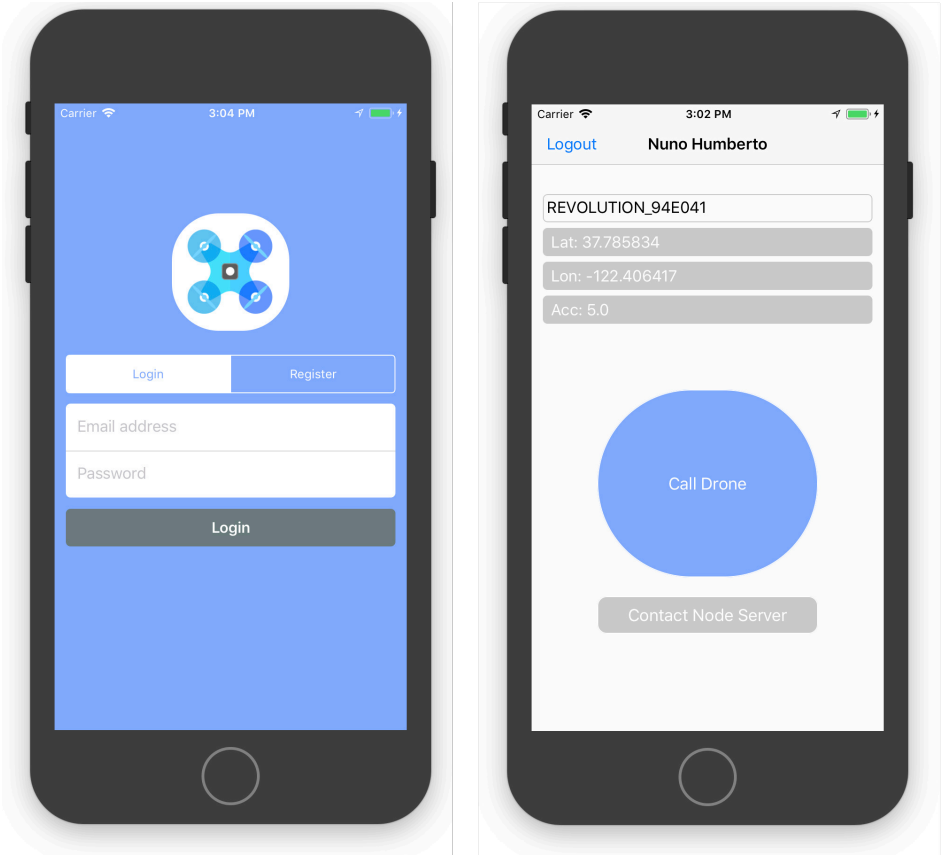

Figure 5.1: Mobile application for invoking connected drones

The **Call Drone** button, as shown in Figure 5.1, allows the user to initiate the panic button request with a single tap. When the user taps this button, the application will obtain the current geographic coordinate of the smartphone by using its integrated GPS receiver, and then include it in a request that is sent to the platform.

5.1.3 Evaluation Procedure

This scenario assumes the drone is at a fixed distance from the user. For this experiment, the drone is placed approximately 100 meters away from the user. The drone starts at the ground level, in a disarmed state, as shown in Figure 5.2.

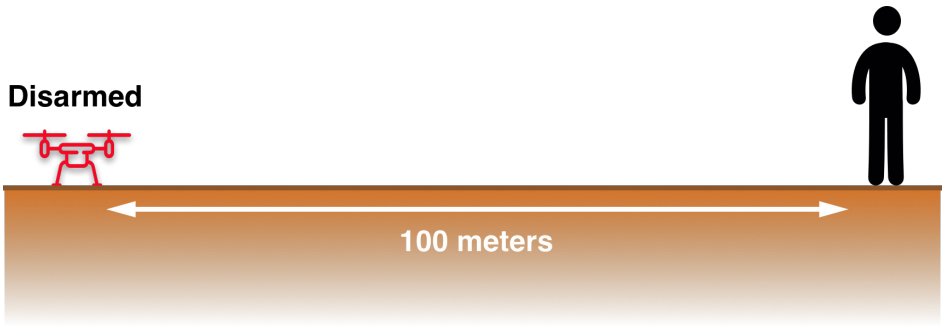

Figure 5.2: Initial status – drone is disarmed

At this point, the user makes use of the developed application to initiate the experiment. To accomplish this, the **Call Drone** button, shown in Figure 5.1, is tapped. Average round-trip times can be obtained and logged in this stage.

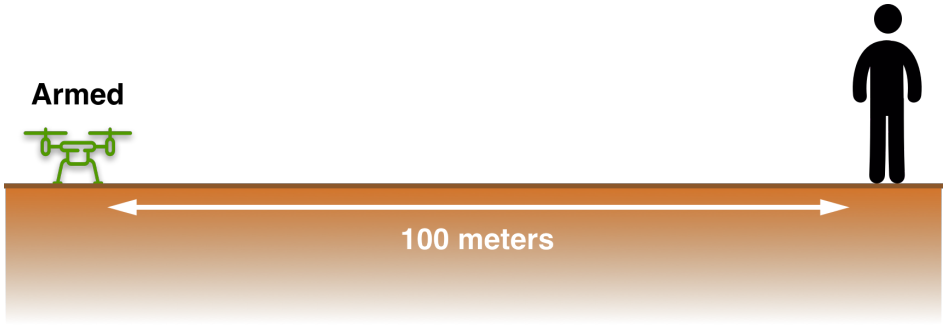

Figure 5.3: First stage – drone is armed

Once the user taps the **Call Drone** button, the procedure is initiated. The first stage consists in placing the drone in an armed state, as shown in Figure 5.3, allowing any following flight operations. Time spent in this stage can be logged, however, it will not represent a relevant metric for overall platform quality since arming time is a configurable value at the **Flight Controller** level. Upon achieving the armed state, the drone should climb to an altitude of 10 meters, as shown in Figure 5.4.

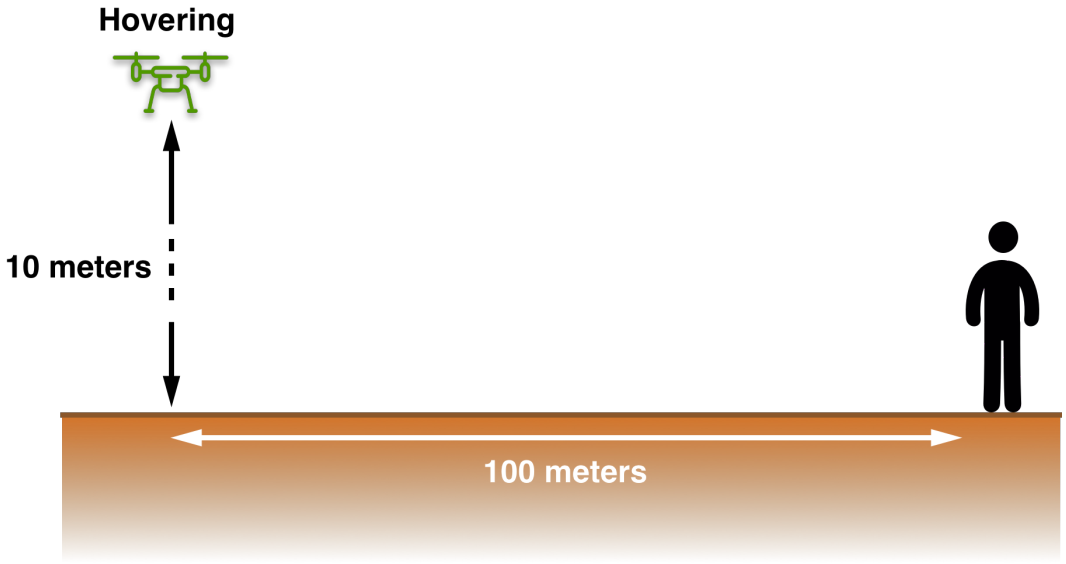

Figure 5.4: Second stage – drone is reaching desired altitude

The second stage finishes once the drone reaches the desired altitude. At this point, the drone can initiate the final stage, which consists in an entirely horizontal flight towards the position of the calling user, as shown in Figure 5.5.

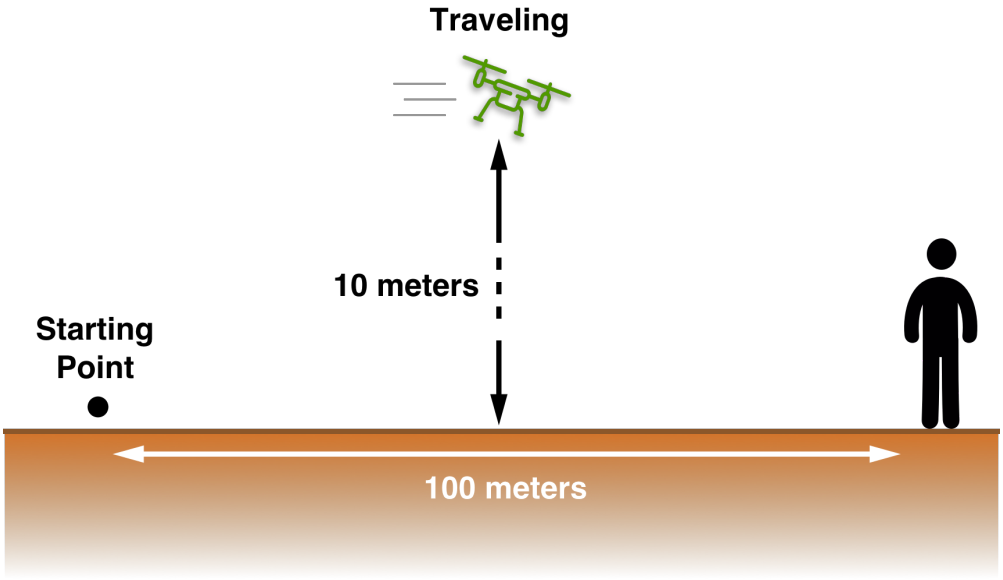

Figure 5.5: Final stage – drone attempts to reach the user

Final flight time metrics for this experiment can be obtained once the drone reaches the position of the user, as depicted in Figure 5.6. Here, all logging activities are stopped, the drone hovers above the location of the user and awaits further commands. At the end of the experiment, timing metrics for message reception and processing, arming, vertical and horizontal flight, as well as the total time elapsed since the initial request up to its completion, can be obtained.

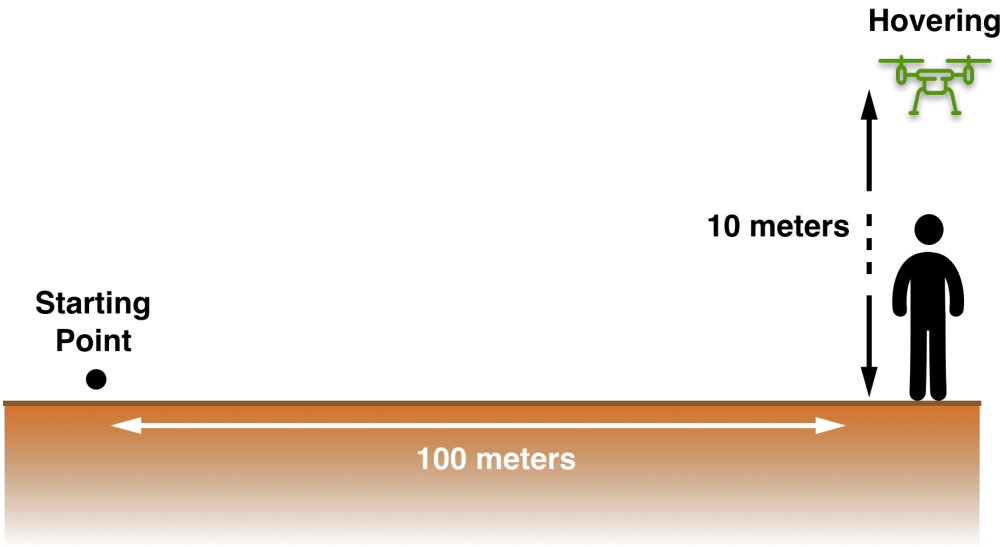

Figure 5.6: Terminating status – drone holds its position above the user

5.1.4 Results

After executing this experiment, the **Mission Planner** produced a mission log file with the path followed by the drone. This file was imported into the viewing front-end of the **Mission**

Planner, showing the path displayed in Figure 5.7. By observing the path, it is possible to

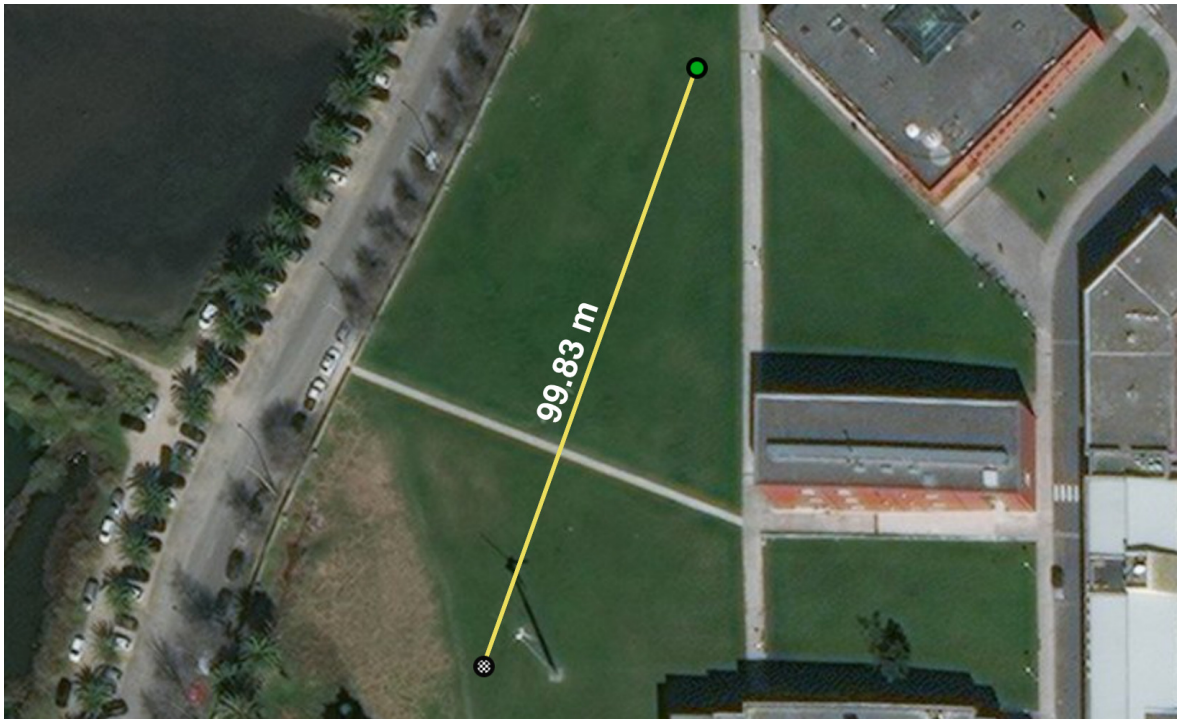

Figure 5.7: Path followed during the execution of the panic button experiment with distance information.

conclude that the drone followed the expected behavior, traveling along an horizontal line of approximately 100 meters. For each stage of the experiment, timing metrics were acquired. These are shown in Figure 5.8.

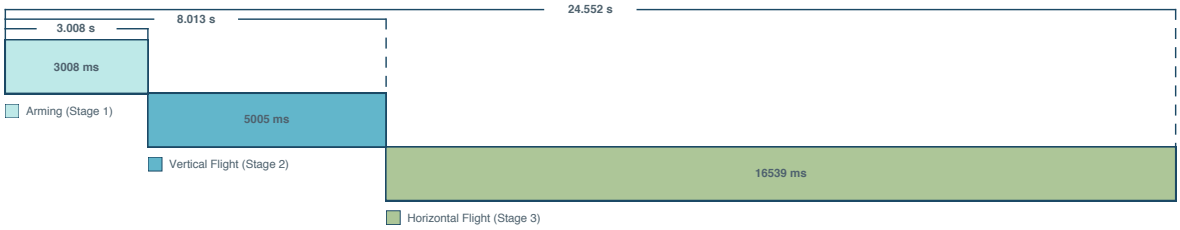

Figure 5.8: Timing representation of each stage of the panic button experiment.

As previously explained, the arming time is a configurable value at the **Flight Controller** level. This value defaults to 3 seconds and was not modified during the execution of every experiment. With this in mind, the time spent in the first stage represents an expected value. As a safety measure, the maximum climb rate is limited to $2m/s$. The maximum horizontal speed was kept at its default value, $10m/s$. Taking these limits into consideration, when the drone must climb 10 meters during the vertical flight stage, it should never complete this task in less than 5 seconds. However, flight speed during vertical or horizontal flights may be affected by external factors such as gusts of wind. The duration of stages 2 and 3 show that the drone executed its vertical flight with a speed of approximately $1.99m/s$ and its horizontal flight with a speed of approximately $6.04m/s$, which both correspond to acceptable values.

Finally, the obtained timing metrics for each stage show that the platform is able to place an initially disarmed drone above a user which is 100 meters away in under 25 seconds, even when including the negligible delay of 3.5 ms required to decode and process the steps of the mission. During the execution of this experiment, additional timing metrics were obtained. These are shown in Figure 5.9 and include the average round-trip time, measured between the drone and the user device which initiates the mission, along with the average time required for the drone to communicate its progress to the **Mission Planner**. These values are agnostic to

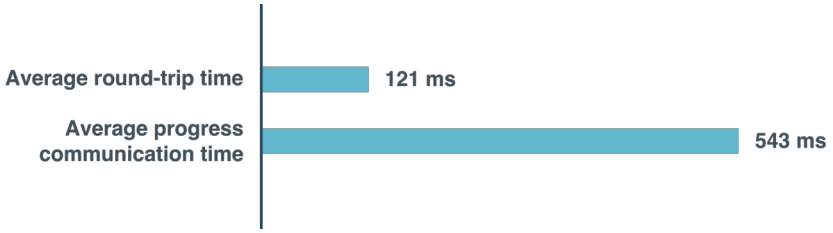

Figure 5.9: Average network round-trip time between the drone and the user device and average time taken for the drone to communicate its progress to the ground.

the complexity of the mission that is executed and are identical in every experiment. They are, however, strongly correlated to the usage of 3G as the communication technology. Progress communication time is notably high, since it requires interfacing with the **Flight Controller** for requesting the actual GPS coordinate and including it in progress messages.

5.2 Automated Aerial Photography

5.2.1 Objectives

The automated aerial photography experiment aims to enable the autonomous acquisition of aerial images of a specified area with arbitrary dimensions. It serves as a sample scenario of an autonomous mission which, in contrast to the experiment described in section 5.1, is associated with a higher complexity and may comprise a large number of commands, depending on the dimensions of the selected area. The successful completion of this experiment effectively proves that, when using the proposed infrastructure, drones are not only capable of reaching automatically assigned waypoints, but also of performing specific tasks upon reaching them. In this case, such task is the acquisition of a picture at the current location.

Aerial imagery obtained by drones can be processed using external tools which are capable of generating useful products such as point clouds, surface models and orthorectified images. An example of such a toolkit is OpenDroneMap [122], which provides a free and open-source solution for generating such products. In this experiment, OpenDroneMap will be used to generate a digital surface model using the pictures acquired during the autonomous mission.

5.2.2 Method

In this scenario, a drone travels along an even distribution of waypoints contained inside a specified quadrilateral area. The placement of these waypoints is automatically calculated

once the user delimits the desired area using the **Mission Planner** front-end described in section 4.2.4. For delimiting the desired area, the user must select three vertexes of the area by clicking on the map element, as shown in Figure 5.10.

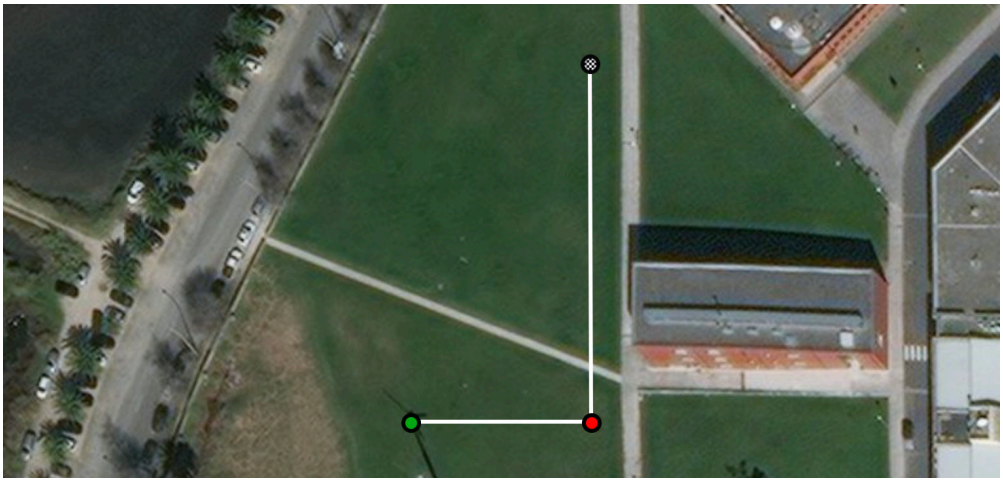

Figure 5.10: Selecting the desired area for photographing

Upon pressing the Map button, the distribution of waypoints is automatically calculated for the user. This sequence of waypoints is displayed in Figure 5.11 and shows the path that will be followed by the drone once the mission is initiated.

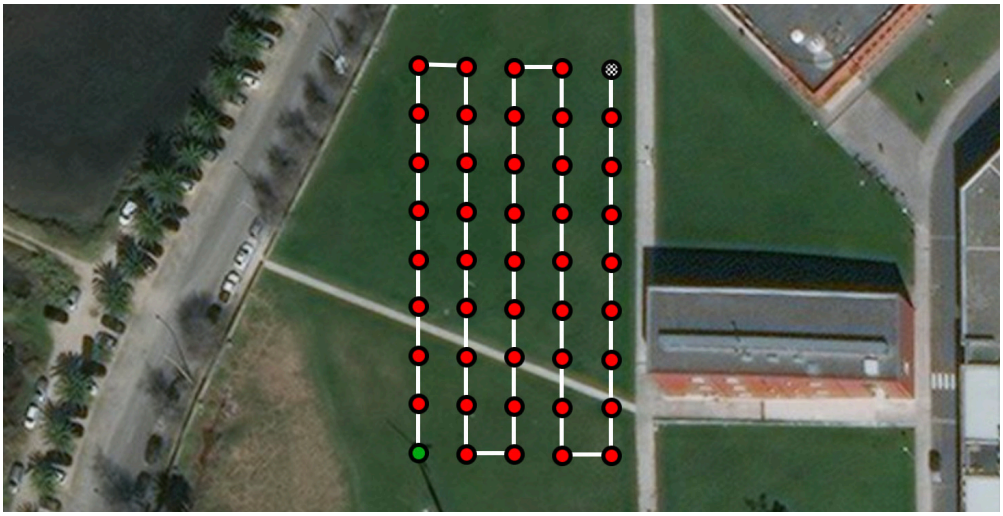

Figure 5.11: Automatically generated sequence of waypoints inside the delimited area

Upon reaching each of these waypoints, the vehicle must be able to communicate with a locally connected camera and issue commands for capturing pictures. For this functionality, an action camera is used. The available camera for this purpose is a GoPro Hero 3 [123]. This camera has the ability to create a wireless access point which can be especially useful to take advantage of its API to issue photographing and filming commands. This way, when reaching a new waypoint, a picture request is automatically executed, causing a photograph to be taken and saved to the internal storage of the camera.

```
[ CAMERA ]: Detected and connected to a GoPro.
[ CAMERA ]: Received a picture request.
[ CAMERA ]: Successfully took a picture.
```

Figure 5.12: Drone Controller debug console – processing a picture request

Figure 5.13: Sample picture taken with the GoPro

When the **Gear Manager** is launched, it will automatically look for the camera and attempt to connect to it. This way, no manual connection setup is needed. Once a connection is successfully established, various types of requests can be sent to the camera, such as photography taking requests or file downloads.

Pictures taken by the GoPro, however, will present a heavy distortion known as barrel effect, as shown in Figure 5.13. This distortion can be corrected using an image processing tool such as ImageMagick. [124]
For each pixel of the output image, ImageMagick calculates the radius (R_{dst}) from the center of the image and, to find which source pixel should be copied, it does the following calculation:

$$R_{src} = A \cdot {R_{dst}}^4 + B \cdot {R_{dst}}^3 + C \cdot {R_{dst}}^2 + D \cdot R_{dst} \tag{5.1}$$

Ideal values for parameters A, B, C and D vary greatly depending on the used lens, these values can be precisely calculated using the Hugin toolset [125] or by trial and error. For the GoPro model used, the following values provide generally good results [126].

$$A = 0.10, B = -0.32, C = 0, D = 1.22 \tag{5.2}$$

In order to use this tool in every captured image, a simple bash script was used, shown in Listing 5.1.

Listing 5.1: Batch Distortion Correction

```
#!/bin/bash
numfiles=$(ls -l *.JPG | wc -l | xargs)
echo "Processing $numfiles pictures..."
mkdir -p out
i=0
for f in *.JPG; do
let "i++"
echo "($i/$numfiles) - $f"
convert $f -distort barrel '0.1 -0.32 0' out/$f
done
echo "Wrote $numfiles files to 'out/'."
```

After running the script using the sample image in Figure 5.13 as input, the result is shown in Figure 5.14. Once every picture has passed through the distortion correction script,

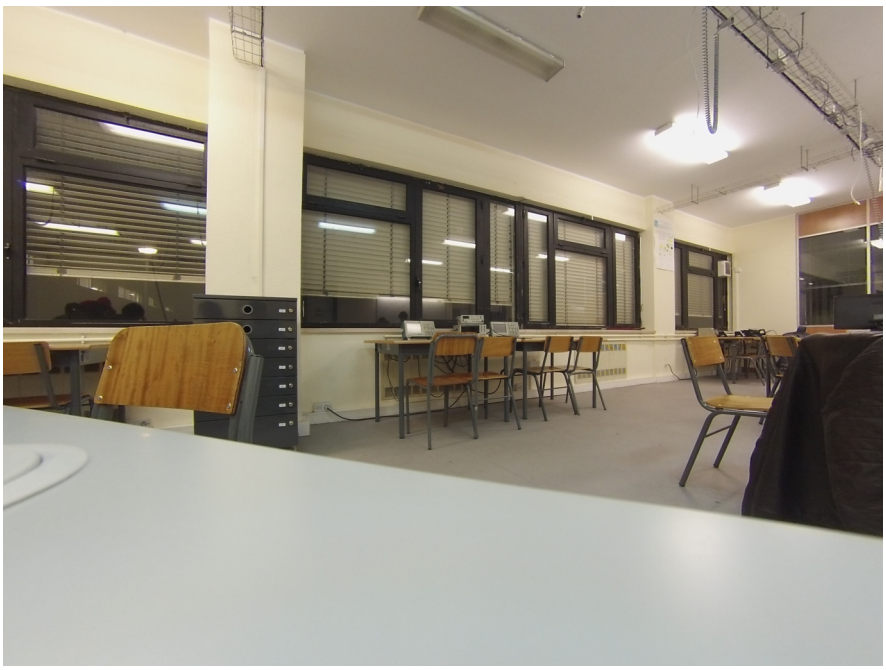

Figure 5.14: Sample picture after distortion correction

OpenDroneMap may be called. The syntax for running the toolkit is shown in Figure 5.15. The tool will then start the processing, requiring no additional user interaction.

```
nunohumberto@PortusLuna ~/rmi
❯ sudo docker run -it --rm -v $(pwd)/images:/code/images -v $(pwd)/odm_orthophoto:/code/odm_orthophoto \
-v $(pwd)/odm_texturing:/code/odm_texturing opendronemap/opendronemap --force-ccd 7.81
[INFO]    Initializing OpenDroneMap app - Wed Feb 07 08:28:26 2018
[INFO]    Running ODM Load Dataset Cell
[DEBUG]   Loading dataset from: /code/images
[DEBUG]   /code/images
```

Figure 5.15: Running OpenDroneMap

5.2.3 Evaluation Procedure

A single drone is needed for the experiment. It starts with the drone hovering nearby, awaiting further orders from the user. The mapping mission, including its area delimitation, is prepared using the front-end of the **Mission Planner**. For this experiment, a rectangular, open area located inside the campus of the University of Aveiro will be used. The dimensions of this area will be of approximately 32 meters per 64 meters and will correspond to the area selected in Figure 5.10 and Figure 5.11, with each waypoint being placed 8 meters apart from the previous. Once the mission is prepared, the user is able to execute the mission by pressing the Send button in the page. Here, the drone should immediately start the photographing process and timing measurements for the mission may be initiated. The user then waits for the autonomous mission to finish. Once the drone completes the path, the image acquisition stage is finished and the user can download the acquired pictures from the camera. Processing time measurements can now be taken for both the distortion removal script and the OpenDroneMap execution. After the execution of OpenDroneMap, a digital surface model is ready to be observed.

5.2.4 Results

Executing this experiment produced a mission log file which contains the path traveled by the drone. In a similar way to the first experiment, the mission log file was imported into the viewing front-end of the **Mission Planner**. The result shown in Figure 5.16 was obtained. As shown, the drone successfully achieved the approximate position of the 45 different waypoints, meaning that the expected behavior was accomplished. In the followed path representation, the green marker shows the starting waypoint. This waypoint is included in the representation with the purpose of showing the initial position of the drone before reaching the first waypoint of the planned mission. From this initial position to the first waypoint of the planned mission, approximately 60 meters must be traveled. Thus, reaching the first waypoint should take more time in comparison to the following waypoints. This difference is shown in Figure 5.17. In a similar way to the first experiment, a timing representation is also provided for the image acquisition stage of this experiment. For readability purposes, the representation shown in Figure 5.19 displays the time elapsed in each set of 9 waypoints, representing the 5 columns of waypoints which were generated for this path. As expected, due to the higher execution time of the first waypoint, the first column of waypoints takes significantly longer to execute in comparison to the other columns, which complete in as low as $60.351s$. The timing representation, however, also shows that the drone is able to map the entire area in approximately 5 minutes and 31 seconds, with the mission decoding and step processing still requiring a negligible $6.29ms$ to take place before mission execution. Once this image acquisition stage is completed, the shell script for barrel distortion removal shown in Listing 5.1 is executed, followed by OpenDroneMap, using the command show in Figure 5.15. The execution time of these stages is shown in Figure 5.18.

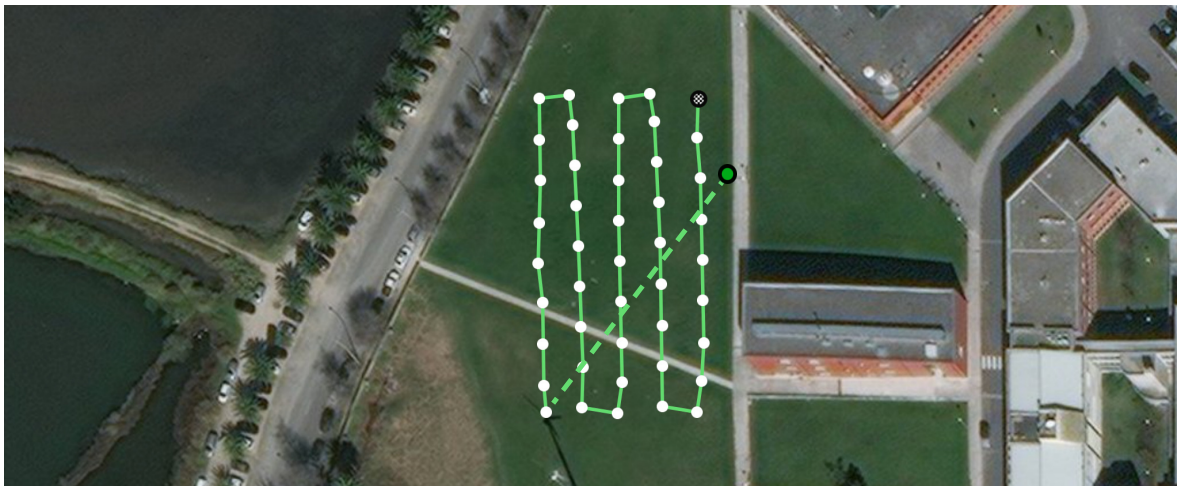

Figure 5.16: Path followed during the execution of the aerial photography experiment.

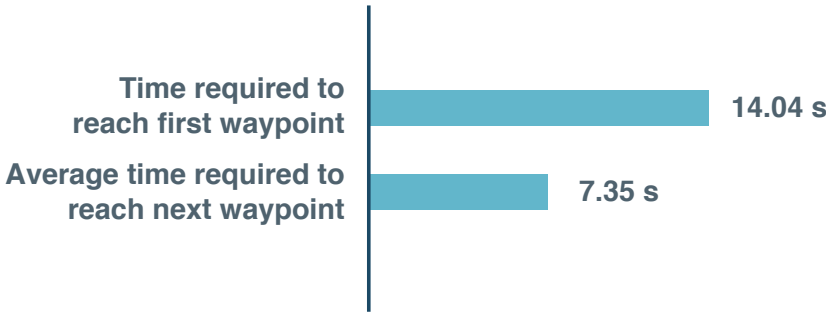

Figure 5.17: Time required for the execution of the first waypoint in comparison to the average time required for the execution of waypoints.

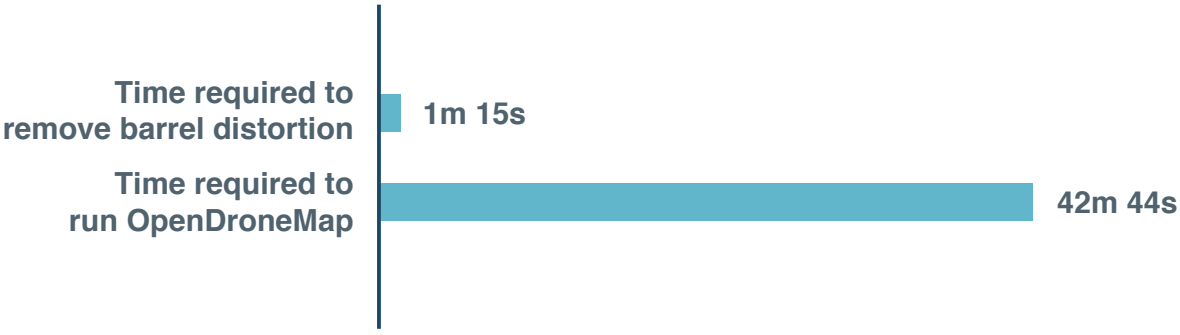

Figure 5.18: Time required for the execution of the barrel distortion removal shell script and OpenDroneMap.

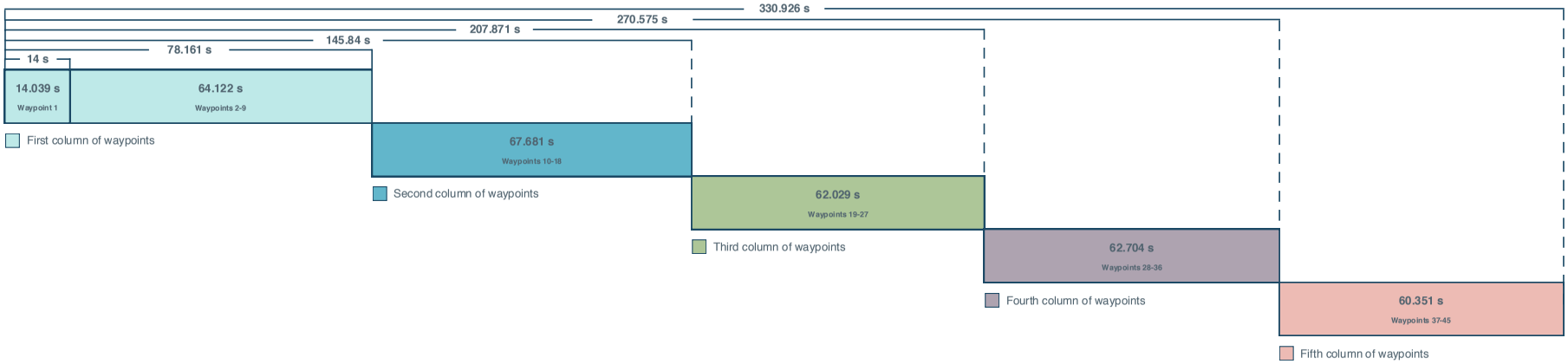

Figure 5.19: Timing representation of the five waypoint columns of the aerial photography experiment.

Upon finishing its execution, OpenDroneMap produced a surface model. Using MeshLab [127], an open-source 3D mesh processing software, the surface model can be rendered. The result is shown in Figure 5.20. The obtained results show that the platform is able to

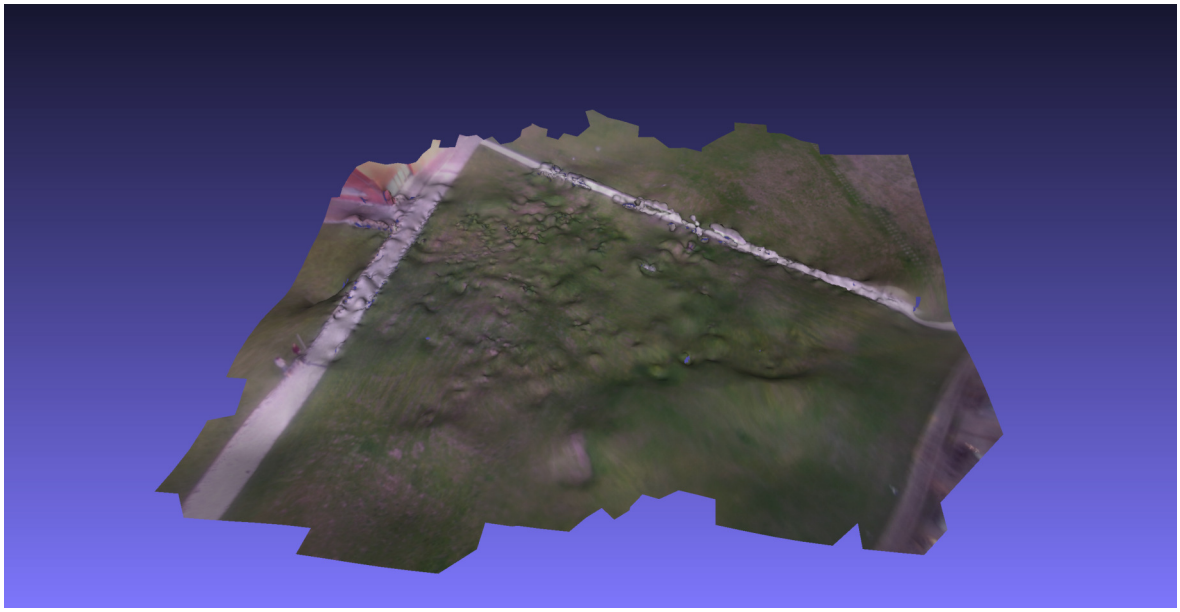

Figure 5.20: Rendering the obtained surface model using MeshLab.

autonomously handle single-drone missions which comprise large number of automatically generated waypoints, while also being able to execute tasks, such as taking pictures, upon reaching each waypoint. In this experiment, a total of 45 waypoints covering an area of approximately $2048m^2$ were completed in 5 minutes and 31 seconds.

5.3 Drone Self-Replacement

5.3.1 Objectives

In this experiment, the most basic collaboration functionality of the platform is evaluated. Here, a single drone participates in a mission, during which self-replacement is triggered. When this happens, the remaining steps of the mission are assigned automatically to a second drone, which executes them sequentially until reaching the end of the mission. Besides having only one drone performing the execution of the mission at a single time, the goal of this experiment is to test if the platform is able to quickly replace an in-flight drone and have a second drone resume its execution.

5.3.2 Method

In a similar way to what is done in the aerial photography experiment in section 5.2, the **Mission Planner** front-end is used to generate a sequence of waypoints that cover a quadrilateral area. However, in this experiment, the drone which is executing the mission will simulate a **CRITICAL** alarm after executing approximately half of the steps of the mission, causing

the **Mission Planner** to replace the in-flight drone with the closest available drone which is connected to the platform at that time. Figure 5.21 displays the path that is sent to the drone, along with the waypoint in which the alarm will be triggered.

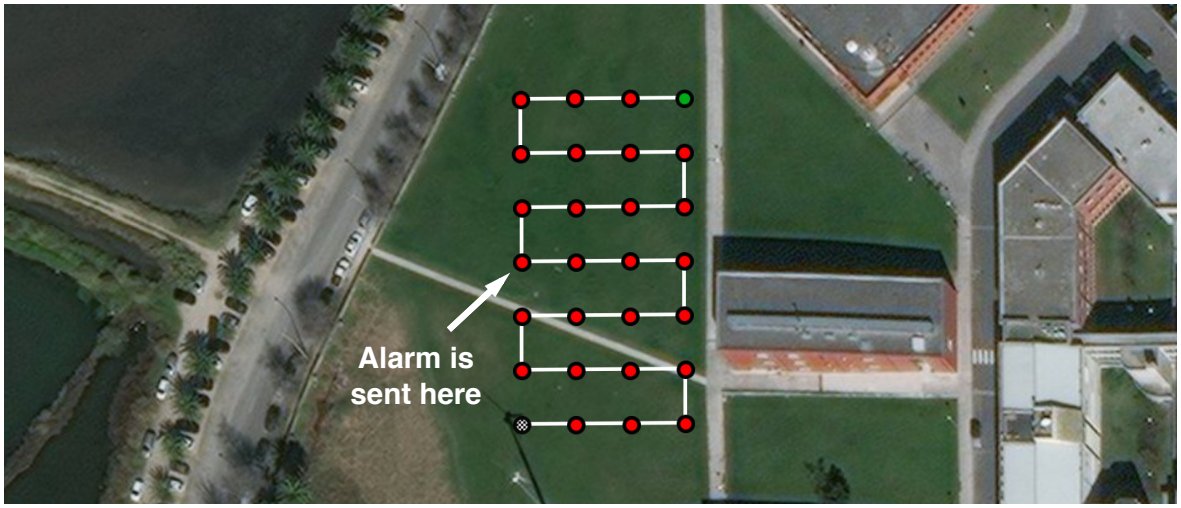

Figure 5.21: Path to be followed in the drone self-replacement experiment.

5.3.3 Evaluation Procedure

Two drones are required for this experiment. Before executing the experiment, the mission path is selected using the front-end of the **Mission Planner**, covering an area which is similar to the one used in the aerial photography experiment in section 5.2, but using waypoints that are 10 meters apart instead of 8 meters since, in contrast to the aerial photography experiment where the number of taken pictures may influence the outcome, waypoint density is not relevant for evaluating the self-replacement performance. A total of 28 waypoints are generated. The drone which is initially assigned to execute the mission is already hovering nearby, awaiting orders from the user, while the drone which will be used to replace the first drone is located at ground level, in a disarmed state. Upon initiating the mission, the first drone is expected to travel to the first waypoint of the delimited area, successively flying towards the following waypoints until it reaches the waypoint in which the alarm is triggered. This waypoint is shown in Figure 5.21. When this alarm is triggered, the **Mission Planner** should instruct the in-flight drone to cancel its current mission and perform a controlled descent until it reaches the ground. While this happens, the **Mission Planner** is expected to assign a new mission to the second drone, which contains only the remaining steps of the mission which was being executed by the first drone. Since the second drone is in a disarmed state when the mission is executed, preparation steps which include arming and vertical ascension are required before resuming the mission of the first drone. Once the second drone reaches the final step of the mission, the viewing interface of the **Mission Planner** can be used to evaluate the path executed by both drones. Timing metrics can also be acquired for both mission execution and mission handover.

5.3.4 Results

As with previous experiments, executing this mission produced a log file which contains the path traveled by both drones. Importing this file to the viewing interface of the **Mission Planner** produced the results shown in Figure 5.22.

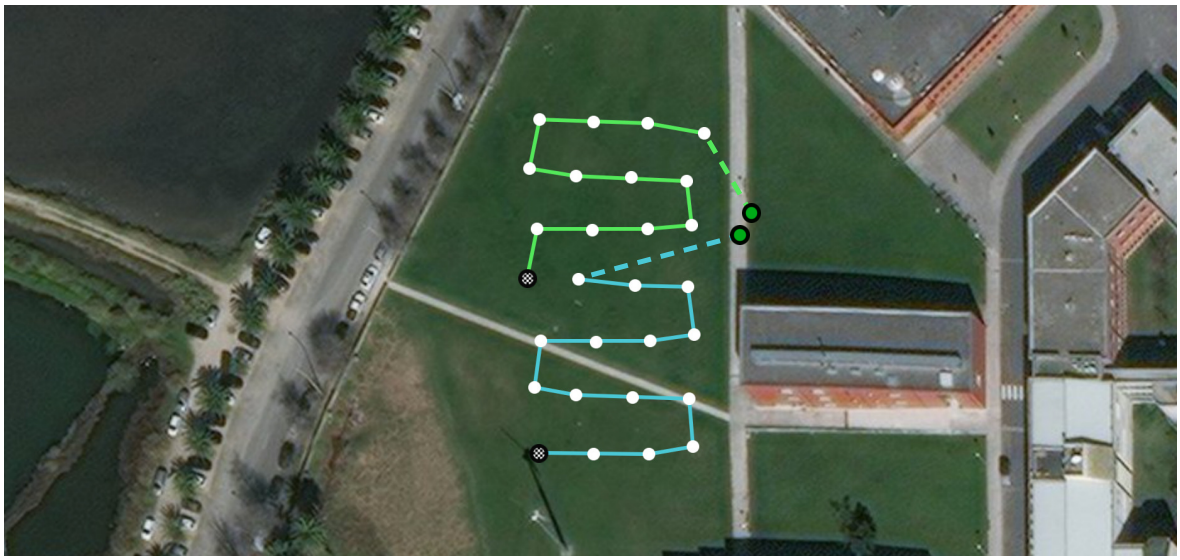

Figure 5.22: Path followed by both drones during the execution of the self-replacement experiment.

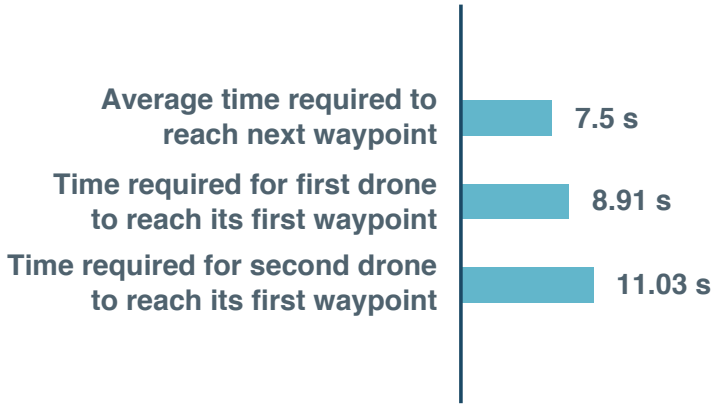

Figure 5.23: Time required for the execution of the first waypoint of each drone in comparison to the average time required for the execution of waypoints in the self-replacement experiment.

The green line represents the path taken by the first drone, while the blue line represents the path taken by the second drone. By observing Figure 5.22, it is possible to conclude that the two drones have collaborated successfully in the experiment. The first drone achieved 13 mission waypoints, while the second drone completed 15 waypoints. Adding together the progress achieved by both drones, every one of the 28 generated waypoints was completed. The timing representation shown in Figure 5.24 contains the time required for each stage of this experiment, including the waypoint flight and landing of the first drone, along with the preparation (arming and vertical flight) and waypoint flight of the second drone. Since both the first and the second drones are, respectively, 17 meters and 40 meters away from

their first waypoint when executing their part of the mission, its execution time is expected to be greater than the remaining waypoints, in a similar way to what happens in the aerial photography experiment in section 5.2. Figure 5.23 shows the difference between the time taken to execute the first waypoint of each drone and the average time required to execute a waypoint.

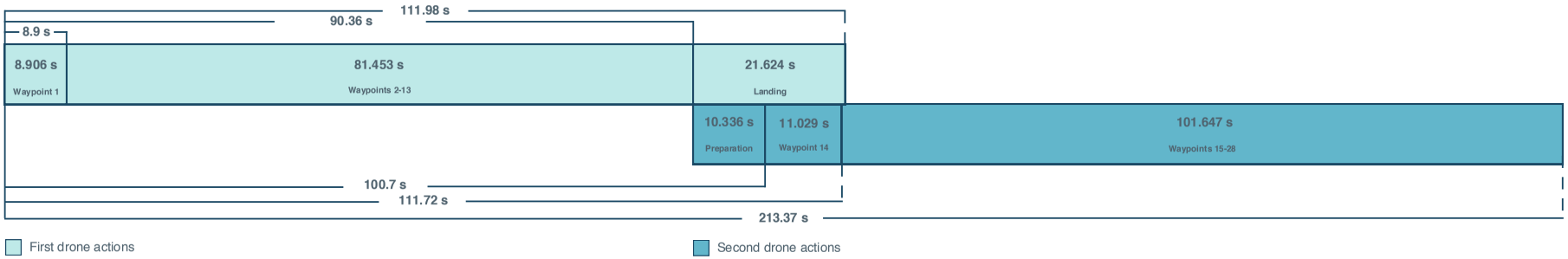

Figure 5.24: Timing representation of the self-replacement experiment.

5.4 Collaborative Sensing

5.4.1 Objectives

This experiment seeks to demonstrate and evaluate the accomplishment of a task which involves the collaboration of multiple drones that are simultaneously connected to the platform. In contrast to the previously described experiment in section 5.3, which aimed to quickly replace an in-flight drone and carry on with its active mission, here the goal is to have more than one drone working on the same task at a given time while each of them provides relevant progress to the mission.

5.4.2 Method

As an example of the specified interaction, a collaborative sensing task is proposed. In this scenario, a drone is actively acquiring data from a set of environmental sensors carried aboard. In order to accomplish this task, the Mission Planner calculates a series of evenly distributed waypoints inside the area of interest and subsequently travels along them, obtaining data from the connected sensors upon reaching each waypoint. This scenario is shown in Figure 5.25.

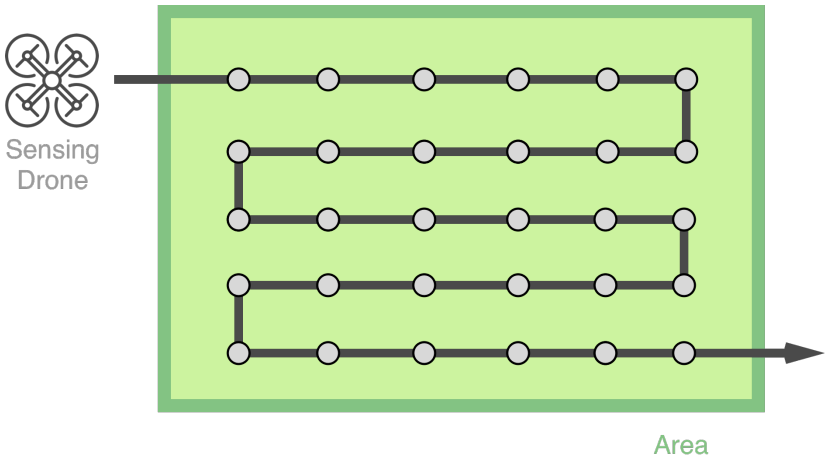

Figure 5.25: Sensing scenario with one single drone

The flight time of a drone, however, is a very limited resource, thus restricting the possible area coverage that can be accomplished with the usage of a single drone. With this limitation in mind, the usage of multiple drones becomes particularly helpful. In a situation where a drone is continuously sensing a delimited area, a request for collaboration has been implemented. For this request to be triggered, the user can set a higher or lower threshold value for each sensor. This way, when the drone acquires an environmental measure which is above or below the expected values, a request for collaboration can be automatically issued. As shown in Figure 5.26, upon receiving this request, the Mission Planner automatically selects available drones from the ones that are currently connected to the platform. For this situation, the nearest drones with high remaining battery life are prioritized. A new, wider area centered on the location in which the alarm was triggered is then calculated along with its set of interior

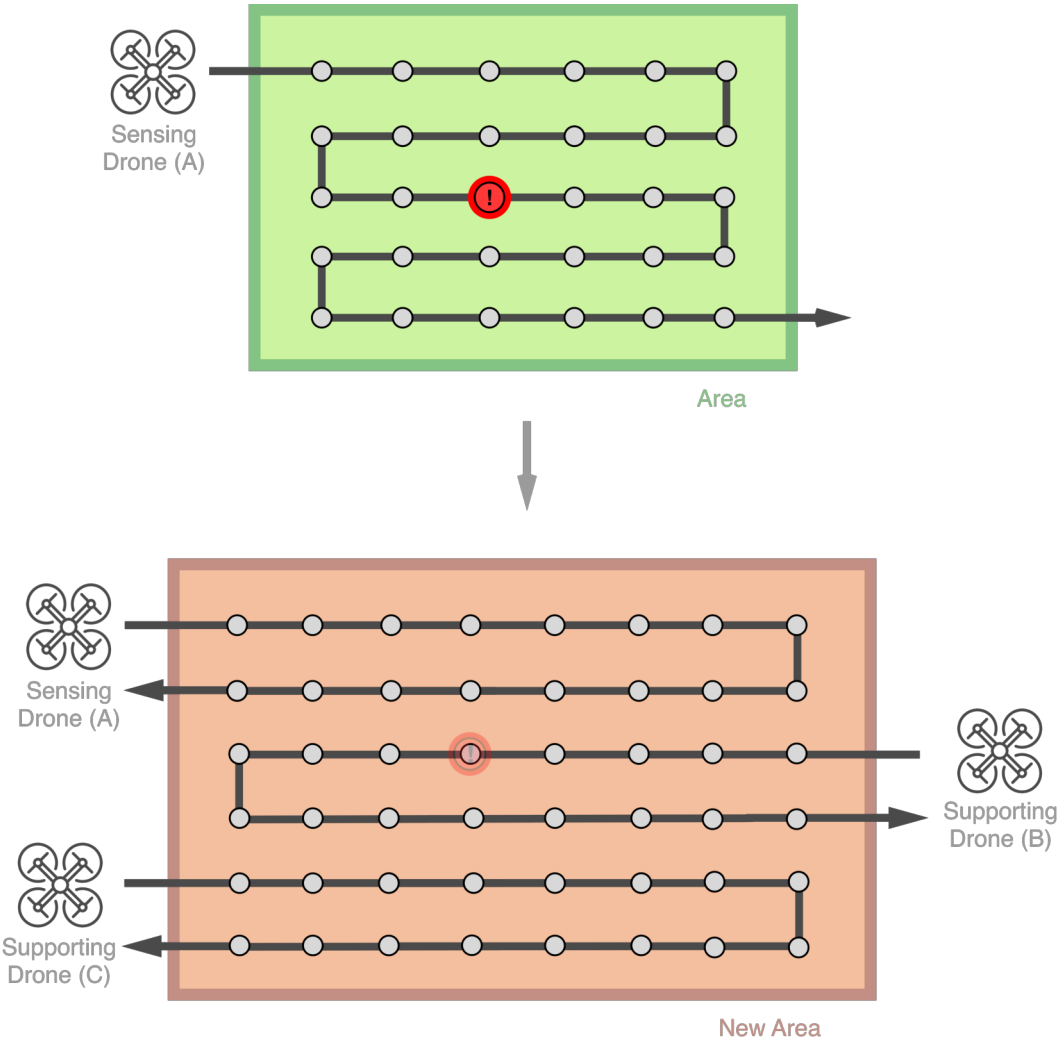

Figure 5.26: Depiction of the collaborative sensing scenario with three drones after automatic area reconstruction, showing the location where the alarm occurs, which becomes the center of the new area.

waypoints. For each new drone that joins a mission for collaboration, its area is expected to increase by 50%. Finally, these waypoints are evenly distributed among the participating drones, and the new sensing task is initiated.

5.4.3 Evaluation Procedure

For this experiment, a single drone is commanded to monitor a delimited area. This will be an open area with a rectangular shape and will be located inside the campus of the University of Aveiro. The dimensions of this area will be of approximately 28 meters per 35 meters, meaning that the drone is initially instructed to cover an area of $980m^2$.

Since the description of this particular scenario uses the acquisition of data from environmental sensors as a task example, the experiment will be carried out providing attention to the properties and limitations of a real environmental sensor. For this purpose, the specifications of a modern sensor module - the SprintIR-6S, will be taken into consideration. The SprintIR-6S,

Figure 5.27: SprintIR-6S CO2 sensor module

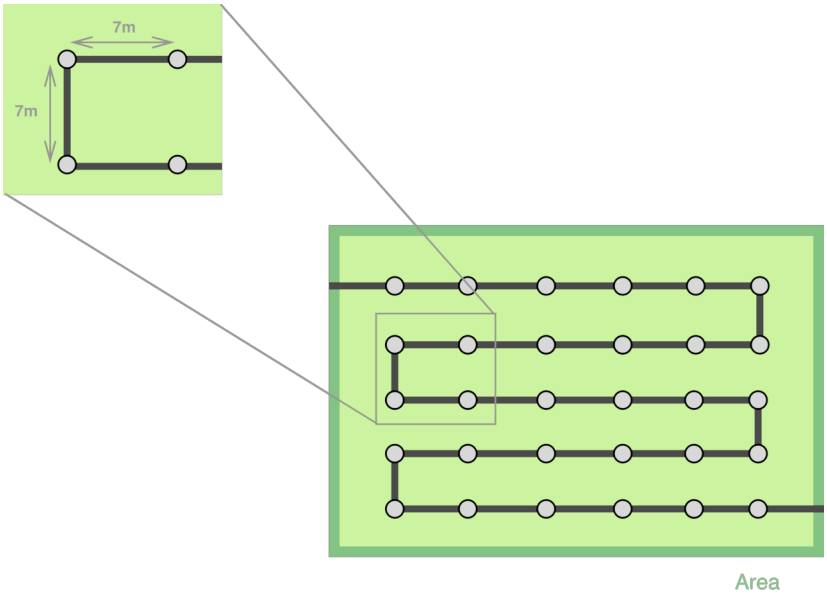

Figure 5.28: Waypoint distribution across the proposed area

shown in Figure 5.27, is a high-speed Nondispersive Infrared (NDIR) CO_2 sensor which also provides humidity and temperature sensing capabilities. The datasheet of the sensor module [128] specifies that CO_2 measures are provided with a response time of up to 2 seconds, varying with the gas flow rate. Regarding this information, a drone must hold its position for, at least, 2 seconds in each waypoint while acquiring a CO_2 measure, before proceeding to the next waypoint. In the proposed area, the waypoints will be distributed 7 meters apart from each other. This way, the entire area can be covered by a total of 30 waypoints, as shown in Figure 5.28.

The request for collaboration is launched by issuing an **OUT_MEASURE** alarm once the initial drone reaches approximately half of the originally assigned waypoints. This simulates the occurrence of an environmental reading which is out of the expected range of values.

During this experiment, three drones will be connected to the platform, all of them on the ground and disarmed. The mission, including the command to simulate abnormal sensor readings, will be prepared and executed using the **Mission Planner** front-end, resembling the initiation mechanism used in section 5.2 and section 5.3.

5.4.4 Results

By feeding the generated logs to the **Mission Planner** viewing interface, it is possible to review the path followed by the first drone before a collaboration request occurred, along with the paths taken by all drones when simultaneously collaborating in the mission. These paths are shown in Figure 5.29.

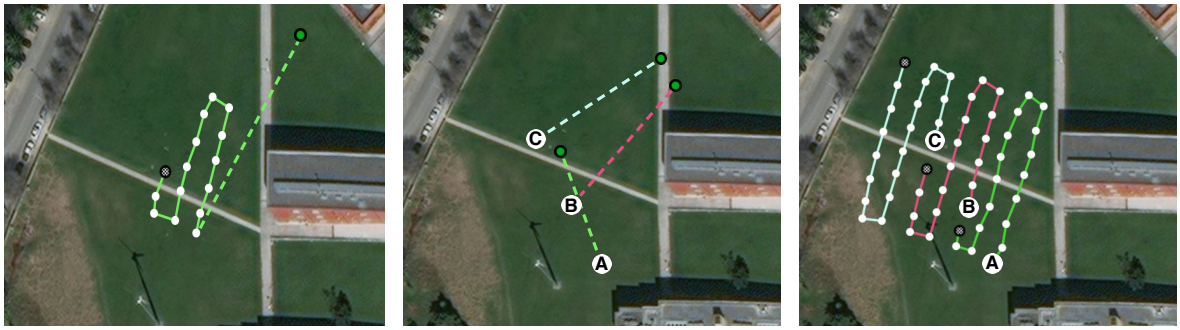

Figure 5.29: Path followed by all drones during the execution of the collaborative sensing experiment. In step (a), drone A executes the initially planned mission until the alarm waypoint is reached. Next, in step (b), a new mission is generated and drones A, B and C simultaneously reach the first waypoint of their part of the mission. Finally, in step (c), all drones collaboratively execute their parts of the mission.

As can be seen in the step c) of Figure 5.29, the mission now includes 56 waypoints instead of 30, covering an area of $2058m^2$ which corresponds to an area increase of 110%. Since two drones have joined the mission for collaboration and each one is expected to provide an area increase of 50%, the new generated area should ideally only be 100% larger than the initial area. However, since the distance between waypoints remains unchanged when missions are extended, if the new generated mission was one row or column shorter, its area would only be, respectively, 80% or 75% larger than the initial mission. Thus, the new mission generated by the **Mission Planner** is the alternative which is closer to the expected area increase of 100%. It is also possible to observe that the new mission was successfully and collaboratively executed by the three drones, since every generated waypoint was achieved. The timing representation shown in Figure 5.30 contains the time required for each stage of this experiment. The collaboration alarm was launched by drone A once it reached waypoint 15 of the initial mission, after approximately 2 minutes had passed. Once the alarm was launched, drones B and C initiated their preparation stages, which include arming, take-off and vertical flight to an altitude of 10 meters. Since drone A was already in flight, no preparation stage was required when the alarm was launched, and thus it proceeded to execute the first waypoint of its part of the mission, waypoint 1. When it finished its preparation stage, drone

B proceeded to execute waypoint 19, while drone C proceeded to execute waypoint 37. After approximately 2 minutes and 34 seconds had passed since the mission expansion, all drones successfully had completed their parts of the mission, meaning that an area of $2058m^2$ with waypoints placed 7 meters apart from each other can be collaboratively covered in this period of time when using three drones.

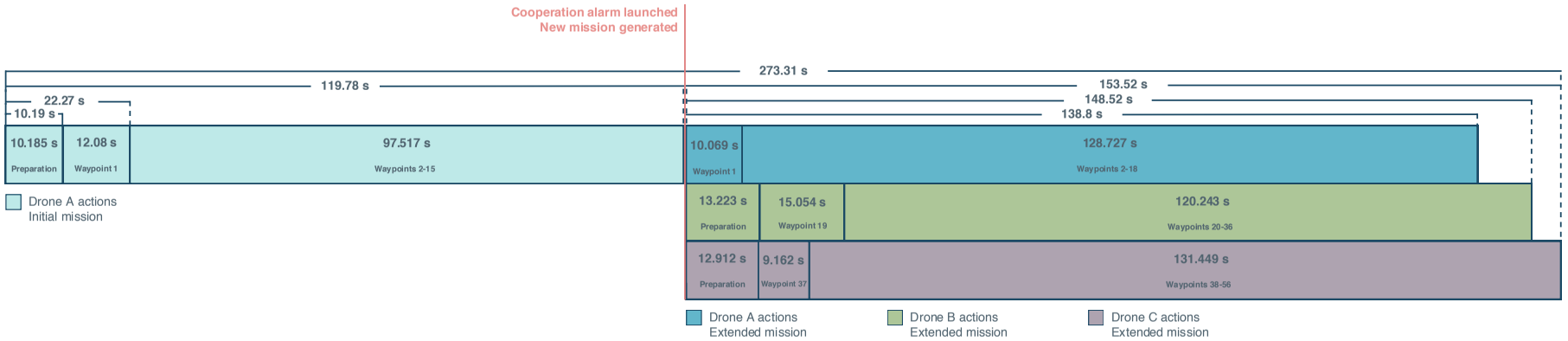

Figure 5.30: Timing representation of the collaborative sensing experiment.

5.5 Conclusions

The execution of the described experiments allows us to evaluate the functionalities of the platform, ranging from the most basic tasks to its most complex capabilities. The results obtained in each of the experiments are able to validate the platform in respect to different levels of functionality, as follows:

- **First experiment** - Confirms that the platform is able to move a drone to the location of a caller who is using the button of a mobile application. This evaluates the geographic coordinate based navigation capabilities of the platform, along with the autonomous take-off of a drone.
- **Second experiment** - Demonstrates that the platform is able to have a drone autonomously survey an area selected by the user, while taking pictures upon reaching each waypoint. This evaluates the mission planning and execution capabilities of the platform, covering the automatic generation of waypoints based in an area selected by the user and the autonomous execution of a mission composed of several waypoints and actions.
- **Third experiment** - Displays the most basic drone collaboration functionality which was implemented, showing that the platform is able to autonomously replace an in-flight drone. This evaluates the self-replacement capabilities of the platform, including the autonomous landing of the in-flight drone and the assignment of the remaining workload to a new drone.
- **Fourth experiment** - Shows that the platform is able to support multiple drones collaborating in a mapping mission simultaneously, allowing larger dimension areas to be autonomously surveyed by a set of drones. This evaluates the multi-drone simultaneous collaboration capabilities of the platform, including the autonomous assignment of similar workloads among multiple connected drones.

Obtaining results for each of the experiments also required the usage of the developed reviewing tools such as the **Logger**, for timing measurements, and the viewing interface of the **Mission Planner**, for displaying a representation of the followed paths, thus also effectively evaluating the mission reviewing capabilities of the platform.

CHAPTER 6

Conclusion and Future Work

This final chapter completes the dissertation providing general conclusions and suggestions for future work.

6.1 Conclusion

This dissertation described the design, implementation and validation of a modular platform for autonomously controlling aerial drones. This platform enables control details to be completely abstracted, enabling an inexperienced user to plan, execute and monitor complex missions with one or more participating drones, while also enabling their collaboration in the execution of these missions. Through a comprehensive set of real-life experiments, it is possible to conclude that the proposed platform is able to meet with every objective that was put forward initially. Additionally, the modular nature of the developed platform allows its expandability for new scenarios, for increased drone and flight controller compatibility and for its scalability. A brief summary of the achieved objectives is provided below.

6.1.1 Building a drone management platform

The core of this dissertation relies on the development of a drone management platform. This platform is composed of a set of software and hardware modules that make up the drone-side and ground-side architecture, along with message broker based communication mechanisms required for their interactions. The modularity of the platform is justified with the loose coupling of each software module, allowing new modules to be developed and implemented with little knowledge of existing modules. The resulting platform is ready to support drone monitoring and autonomous drone flight in tasks ranging from basic navigation commands to the automation of complex paths and collaboration of multiple drone systems.

6.1.2 Building a drone system

A crucial part of this work involved building a drone system which could easily be integrated in the control platform. The result is a reliable drone system, shown in Figure 6.1, which can easily be replicated if a higher number of mission participants is required. The usage of a hexacopter allows for an increased take-off weight, providing a high flexibility in the usage of different sensor payloads.

Figure 6.1: Two drones in a disarmed state before execution of the self-replacement experiment.

6.1.3 Control abstraction

As one of the main objectives of this work, the resulting platform provides mechanisms for high-level control, abstracting flight details from the user. The best example of this functionality is described in the panic button experiment in section 5.1, in which the platform was successfully able to command a drone to arm, take-off, and fly towards a user with a single tap of a button.

6.1.4 Mission execution

The successful completion of the proposed experiments shows that the platform is able to correctly handle the execution of missions containing large sequences of commands and waypoints. This enables to conclude that, not only the mechanisms developed to implement mission execution work as expected, but also the set of tools that enable planning missions and evaluating mission results.

6.1.5 Drone collaboration

Executing the final two experiments involves the participation of more than one drone in the same mission. The obtained results in the self-replacement experiment described in section 5.3 show that the platform is able to autonomously replace an in-flight drone and hand-off its work load to a newly assigned drone. The successful execution of the collaborative sensing

scenario described in section 5.4 shows that the platform supports three drones collaborating in a mission simultaneously, covering an area of $2058m^2$ in approximately 2 minutes and 34 seconds.

6.2 Future Work

Given that the possible usage scenarios of this platform are vast and apply to a broad range of sectors, it leaves wide room for improvement and future work. Some suggestions for future work are provided as follows.

6.2.1 Drone-to-drone communication

In the actual state of the platform, drones make use of ground side components for every interaction. Implementing collaboration mechanisms which do not constantly rely on a connection to ground but instead take place using drone-to-drone communications could be interesting, for instance, to avoid collisions or in the scenario of ground connection loss. This could also promote drone differentiation, where drones in a swarm that do not carry a 3G modem aboard relay ground communication to a drone with such capability.

6.2.2 Complex mission planning

Mission planning and execution could be further improved by adding the possibility of creating conditional steps and fallbacks. A suggestion would be to represent each mission as a finite state machine, in which states represent mission steps and state transitions depend on the result of the execution of each step. This would enable missions to execute differently according to obtained sensor measurements or in case of a failure.

6.2.3 Fail-safe mechanisms

As a safety measure, devices such as parachutes or sirens could be added to drones. Further improvements in telemetry analysis could enable a drone to recognize its own anomalous behavior and automatically deploy these devices in case of an imminent crash.

6.2.4 Broaden flight controller compatibility

At the present time, only flight controllers which implement the UAVTalk protocol are supported by the platform. Extending this support, for example, to the MAVLink protocol or to DJI flight controllers would make the platform compatible with a new set of devices. In order to accomplish this, additional capabilities should be added to the **Flight Controller Interface** submodule of the **Drone Controller**.

References

[1] S. Chaumette, "Collaboration Between Autonomous Drones and Swarming", in *UAV Networks and Communications*, Cambridge University Press, 2017, pp. 177–193. DOI: 10.1017/9781316335765.009. [Online]. Available: https://www.cambridge.org/core/product/identifier/CBO9781316335765A060/type/book%7B%5C_%7Dpart.

[2] I. Colomina and P. Molina, "Unmanned aerial systems for photogrammetry and remote sensing: A review", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 92, pp. 79–97, Jun. 2014, ISSN: 0924-2716. DOI: 10.1016/J.ISPRSJPRS.2014.02.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271614000501.

[3] D. Gonzalez-Aguilera and P. Rodriguez-Gonzalvez, "Drones-An Open Access Journal", *Drones*, 2017. DOI: 10.3390/drones1010001. [Online]. Available: www.mdpi.com/journal/drones.

[4] N. Mohd Noor, A. Abdullah, and M. Hashim, "Remote sensing UAV/drones and its applications for urban areas: a review", *IOP Conference Series: Earth and Environmental Science*, vol. 169, no. 1, p. 012 003, Jul. 2018, ISSN: 1755-1315. DOI: 10.1088/1755-1315/169/1/012003. [Online]. Available: http://stacks.iop.org/1755-1315/169/i=1/a=012003?key=crossref.ff09c78b4233f15a0e590f58ca43f99d.

[5] A. Chapman, "Types of Drones: Multi-Rotor vs Fixed-Wing vs Single Rotor vs Hybrid VTOL", *DRONE magazine, issue 3*, Jun. 2016. [Online]. Available: https://www.auav.com.au/articles/drone-types/.

[6] Q. Feng, J. Liu, J. Gong, Q. Feng, J. Liu, and J. Gong, "UAV Remote Sensing for Urban Vegetation Mapping Using Random Forest and Texture Analysis", *Remote Sensing*, vol. 7, no. 1, pp. 1074–1094, Jan. 2015, ISSN: 2072-4292. DOI: 10.3390/rs70101074. [Online]. Available: http://www.mdpi.com/2072-4292/7/1/1074.

[7] G. Salvo, L. Caruso, and A. Scordo, "Urban Traffic Analysis through an UAV", *Procedia - Social and Behavioral Sciences*, vol. 111, pp. 1083–1091, Feb. 2014, ISSN: 1877-0428. DOI: 10.1016/J.SBSPRO.2014.01.143. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187704281400144X?via%7B%5C%%7D3Dihub.

[8] F. Mohammed, A. Idries, N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "UAVs for smart cities: Opportunities and challenges", in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, May 2014, pp. 267–273, ISBN: 978-1-4799-2376-2. DOI: 10.1109/ICUAS.2014.6842265. [Online]. Available: http://ieeexplore.ieee.org/document/6842265/.

[9] R. Clarke and L. Bennett Moses, "The regulation of civilian drones' impacts on public safety", *Computer Law & Security Review*, vol. 30, no. 3, pp. 263–285, Jun. 2014, ISSN: 0267-3649. DOI: 10.1016/J.CLSR.2014.03.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0267364914000594.

[10] D. Mardiansyah and A. H. S. Budi, "UAV Vision System for Rescue Payload Delivery", *IOP Conference Series: Materials Science and Engineering*, vol. 384, p. 012 005, Jul. 2018, ISSN: 1757-8981. DOI: 10.1088/1757-899X/384/1/012005. [Online]. Available: http://stacks.iop.org/1757-899X/384/i=1/a=012005?key=crossref.11cf7898e5e2258a325c953ebddb7894.

[11] E. Lygouras, A. Gasteratos, K. Tarchanidis, and A. Mitropoulos, "ROLFER: A fully autonomous aerial rescue support system", *Microprocessors and Microsystems*, vol. 61, pp. 32–42, Sep. 2018, ISSN: 0141-9331. DOI: 10.1016/J.MICPRO.2018.05.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933118301224?via%7B%5C%%7D3Dihub.

[12] H. Kayan, R. Eslampanah, F. Yeganli, and M. Askar, "Heat leakage detection and surveiallance using aerial thermography drone", in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, IEEE, May 2018, pp. 1–4, ISBN: 978-1-5386-1501-0. DOI: 10.1109/SIU.2018.8404366. [Online]. Available: https://ieeexplore.ieee.org/document/8404366/.

[13] M. Abarca, C. Saito, A. Angulo, J. A. Paredes, and F. Cuellar, "Design and development of an hexacopter for air quality monitoring at high altitudes", in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, IEEE, Aug. 2017, pp. 1457–1462, ISBN: 978-1-5090-6781-7. DOI: 10.1109/COASE.2017.8256309. [Online]. Available: http://ieeexplore.ieee.org/document/8256309/.

[14] I. Gomes, L. Peteiro, J. Bueno-Pardo, R. Albuquerque, S. Pérez-Jorge, E. R. Oliveira, F. L. Alves, and H. Queiroga, "What's a picture really worth? On the use of drone aerial imagery to estimate intertidal rocky shore mussel demographic parameters", *Estuarine, Coastal and Shelf Science*, vol. 213, pp. 185–198, Nov. 2018, ISSN: 0272-7714. DOI: 10.1016/J.ECSS.2018.08.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0272771418303214?via%7B%5C%%7D3Dihub.

[15] P. Gray, J. Ridge, S. Poulin, A. Seymour, A. Schwantes, J. Swenson, D. Johnston, P. C. Gray, J. T. Ridge, S. K. Poulin, A. C. Seymour, A. M. Schwantes, J. J. Swenson, and D. W. Johnston, "Integrating Drone Imagery into High Resolution Satellite Remote Sensing Assessments of Estuarine Environments", *Remote Sensing*, vol. 10, no. 8, p. 1257, Aug. 2018, ISSN: 2072-4292. DOI: 10.3390/rs10081257. [Online]. Available: http://www.mdpi.com/2072-4292/10/8/1257.

[16] J. Huuskonen and T. Oksanen, "Soil sampling with drones and augmented reality in precision agriculture", *Computers and Electronics in Agriculture*, vol. 154, pp. 25–35, Nov. 2018, ISSN: 0168-1699. DOI: 10.1016/J.COMPAG.2018.08.039. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169918301650?via%7B%5C%%7D3Dihub.

[17] P. Surový, N. Almeida Ribeiro, and D. Panagiotidis, "Estimation of positions and heights from UAV-sensed imagery in tree plantations in agrosilvopastoral systems", *International Journal of Remote Sensing*, vol. 39, no. 14, pp. 4786–4800, Aug. 2018, ISSN: 0143-1161. DOI: 10.1080/01431161.2018.1434329. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/01431161.2018.1434329.

[18] T. R. Goodbody, N. C. Coops, T. Hermosilla, P. Tompalski, and P. Crawford, "Assessing the status of forest regeneration using digital aerial photogrammetry and unmanned aerial systems", *International Journal of Remote Sensing*, vol. 39, no. 15-16, pp. 5246–5264, Aug. 2018, ISSN: 0143-1161. DOI: 10.1080/01431161.2017.1402387. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/01431161.2017.1402387.

[19] M. Schirrmann, A. Giebel, F. Gleiniger, M. Pflanz, J. Lentschke, K.-H. Dammer, M. Schirrmann, A. Giebel, F. Gleiniger, M. Pflanz, J. Lentschke, and K.-H. Dammer, "Monitoring Agronomic Parameters of Winter Wheat Crops with Low-Cost UAV Imagery", *Remote Sensing*, vol. 8, no. 9, p. 706, Aug. 2016, ISSN: 2072-4292. DOI: 10.3390/rs8090706. [Online]. Available: http://www.mdpi.com/2072-4292/8/9/706.

[20] D. Mirk and H. Hlavacs, "Virtual Tourism with Drones", in *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use - DroNet '15*, New York, New York, USA: ACM Press, 2015, pp. 45–50, ISBN: 9781450335010. DOI: 10.1145/2750675.2750681. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2750675.2750681.

[21] *DJI - The Future Of Possible*. [Online]. Available: https://www.dji.com/ (visited on 09/17/2018).

[22] D. Joshi, *Top Drone Manufacturers & Companies To Watch & Invest In 2017*, Jul. 2017. [Online]. Available: https://www.businessinsider.com/top-drone-manufacturers-companies-invest-stocks-2017-07.

[23] A. Glaser, "DJI is running away with the drone market", *Recode*, p. 5, 2017. [Online]. Available: https://www.recode.net/2017/4/14/14690576/drone-market-share-growth-charts-dji-forecast.

[24] P. Akademia Baru, S. Sabikan, and S. W. Nawawi, "Open-Source Project (OSPs) Platform for Outdoor Quadcopter", Tech. Rep. 1, 2016, pp. 13–27. [Online]. Available: http://www.akademiabaru.com/doc/ARDV24%7B%5C_%7DN1%7B%5C_%7DP13%7B%5C_%7D27.pdf.

[25] H. Chao, Y. Cao, and Y. Chen, "Autopilots for small unmanned aerial vehicles: A survey", *International Journal of Control, Automation and Systems*, vol. 8, no. 1, pp. 36–44, Feb. 2010, ISSN: 1598-6446. DOI:

10.1007/s12555-010-0105-z. [Online]. Available: http://link.springer.com/10.1007/s12555-010-0105-z.

[26] *What's difference of NAZA-M Lite/NAZA-M V1/NAZA-M V2*. [Online]. Available: https://www.dji.com/newsroom/news/whats-difference-of-naza-m-litenaza-m-v1naza-m-v2 (visited on 09/18/2018).

[27] *DJI - The World Leader in Camera Drones/Quadcopters for Aerial Photography*. [Online]. Available: https://www.dji.com/2-4g-bluetooth-datalink (visited on 09/18/2018).

[28] *DJI Naza-M V2 Multirotor Gyro System w/ GPS - HeliPal*. [Online]. Available: http://www.helipal.com/dji-naza-m-multirotor-gyro-system.html (visited on 09/18/2018).

[29] *DJI N3 Naza Flight Controller and GPS | professional-multirotors*. [Online]. Available: https://www.professional-multirotors.com/product/dji-n3-naza-flight-controller-gps/ (visited on 09/18/2018).

[30] *N3 - Specs, FAQ, Tutorials, Downloads and DJI GO - DJI*. [Online]. Available: https://www.dji.com/n3/info%7B%5C#%7Dspecs (visited on 09/18/2018).

[31] *A3*. [Online]. Available: https://www.dji.com/a3/info%7B%5C#%7Dspecs (visited on 09/18/2018).

[32] *Buy N3 - DJI Store*. [Online]. Available: https://store.dji.com/product/n3?from=menu%7B%5C_%7Dproducts (visited on 09/18/2018).

[33] *Buy A3 - DJI Store*. [Online]. Available: https://store.dji.com/product/a3?from=menu%7B%5C_%7Dproducts (visited on 09/18/2018).

[34] *DJI Developer*. [Online]. Available: https://developer.dji.com/onboard-sdk/ (visited on 09/18/2018).

[35] *Buy A3 Pro - DJI Store*. [Online]. Available: https://store.dji.com/product/a3-pro (visited on 09/18/2018).

[36] *Parrot C.H.U.C.K. | Official Parrot® site*. [Online]. Available: https://www.parrot.com/eu/business-solutions/parrot-chuck%7B%5C#%7Dparrot-chuck (visited on 09/18/2018).

[37] *Parrot C.H.U.C.K. | Official Parrot® site*. [Online]. Available: https://www.parrot.com/eu/business-solutions/parrot-chuck%7B%5C#%7Dtechnicals (visited on 09/18/2018).

[38] *Parrot PF070251 Disco C.H.U.C.K. Unit Replacement autopilot module for Parrot Disco drone at Crutchfield.com*. [Online]. Available: https://www.crutchfield.com/S-x8MaU53RmBn/p%7B%5C_%7D333PF70251/Parrot-PF070251-Disco-C-H-U-C-K-Unit.html (visited on 09/18/2018).

[39] *Emlid Edge — Advanced drone controller with HDMI video input*. [Online]. Available: https://emlid.com/edge/ (visited on 09/19/2018).

[40] *Edge docs*. [Online]. Available: https://docs.emlid.com/edge/ (visited on 09/19/2018).

[41] E. Ebeid, M. Skriver, and J. Jin, "A Survey on Open-Source Flight Control Platforms of Unmanned Aerial Vehicle", in *2017 Euromicro Conference on Digital System Design (DSD)*, IEEE, Aug. 2017, pp. 396–402, ISBN: 978-1-5386-2146-2. DOI: 10.1109/DSD.2017.30. [Online]. Available: http://ieeexplore.ieee.org/document/8049816/.

[42] *NuttX Real-Time Operating System - NuttX Real-Time Operating System*. [Online]. Available: http://nuttx.org/ (visited on 09/19/2018).

[43] *Pixhawk 4 · PX4 User Guide*. [Online]. Available: https://docs.px4.io/en/flight%7B%5C_%7Dcontroller/pixhawk4.html (visited on 09/19/2018).

[44] *Hardware - Dronecode*. [Online]. Available: https://www.dronecode.org/hardware/ (visited on 09/19/2018).

[45] *Dronecode - The Open Source UAV Platform*. [Online]. Available: https://www.dronecode.org/ (visited on 09/19/2018).

[46] *The Linux Foundation – Supporting Open Source Ecosystems*. [Online]. Available: https://www.linuxfoundation.org/ (visited on 09/19/2018).

[47] *Buy Pixhawk 4*. [Online]. Available: https://shop.holybro.com/pixhawk-4%7B%5C_%7Dp1089.html (visited on 09/19/2018).

[48] S. Debnath and J. Nayak, "Visual odometry data fusion for indoor localization of an unmanned aerial vehicle", in *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, IEEE, Sep. 2017, pp. 279–284, ISBN: VO -. DOI: 10.1109/ICPCSI.2017.8392247. [Online]. Available: https://ieeexplore.ieee.org/document/8392247/.

[49] W. Liu, C. Yu, X. Wang, Y. Zhang, and Y. Yu, "The Altitude Hold Algorithm of UAV Based on Millimeter Wave Radar Sensors", in *Proceedings - 9th International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC 2017*, vol. 1, IEEE, Aug. 2017, pp. 436–439, ISBN: 9781538630228. DOI: 10.1109/IHMSC.2017.106. [Online]. Available: http://ieeexplore.ieee.org/document/8047665/.

[50] M. Beul, N. Krombach, M. Nieuwenhuisen, D. Droeschel, and S. Behnke, "Autonomous navigation in a warehouse with a cognitive micro aerial vehicle", in *Studies in Computational Intelligence*, vol. 707, 2017, pp. 487–524. DOI: 10.1007/978-3-319-54927-9_15. [Online]. Available: http://link.springer.com/10.1007/978-3-319-54927-9%7B%5C_%7D15.

[51] R. Opromolla, G. Fasano, G. Rufino, M. Grassi, and A. Savvaris, "LIDAR-inertial integration for UAV localization and mapping in complex environments", in *2016 International Conference on Unmanned Aircraft Systems, ICUAS 2016*, IEEE, Jun. 2016, pp. 649–656, ISBN: 9781467393331. DOI: 10.1109/ICUAS.2016.7502580. [Online]. Available: http://ieeexplore.ieee.org/document/7502580/.

[52] G. Crespo, G. Glez-de-Rivera, J. Garrido, and R. Ponticelli, "Setup of a communication and control systems of a quadrotor type Unmanned Aerial Vehicle", in *Proceedings of the 2014 29th Conference on Design of Circuits and Integrated Systems, DCIS 2014*, IEEE, Nov. 2014, pp. 1–6, ISBN: 9781479957439. DOI: 10.1109/DCIS.2014.7035590. [Online]. Available: http://ieeexplore.ieee.org/document/7035590/.

[53] *Pixhawk Series · PX4 User Guide*. [Online]. Available: https://docs.px4.io/en/flight%7B%5C_%7Dcontroller/pixhawk%7B%5C_%7Dseries.html (visited on 09/19/2018).

[54] *OpenPilot CC3D Revolution Revo 10DOF STM32F4 Flight Controller Staight Pin | Alexnld.com*. [Online]. Available: https://alexnld.com/product/openpilot-cc3d-revolution-revo-10dof-stm32f4-flight-controller-staight-pin/ (visited on 09/19/2018).

[55] *Welcome to LibrePilot/OpenPilot Wiki — LibrePilot/OpenPilot Wiki 0.1.4 documentation*. [Online]. Available: https://opwiki.readthedocs.io/en/latest/index.html (visited on 09/19/2018).

[56] *Revolution Board Setup — LibrePilot/OpenPilot Wiki 0.1.4 documentation*. [Online]. Available: https://opwiki.readthedocs.io/en/latest/user%7B%5C_%7Dmanual/revo/revo.html (visited on 09/19/2018).

[57] *OpenPilot CC3D Revolution (Revo) 32bit F4 Based Flight Controller w/Integrated 433Mhz OPLink*. [Online]. Available: https://hobbyking.com/en%7B%5C_%7Dus/openpilot-cc3d-revolution-revo-32bit-flight-controller-w-integrated-433mhz-oplink.html?%7B%5C_%7D%7B%5C_%7D%7B%5C_%7Dstore=en%7B%5C_%7Dus (visited on 09/19/2018).

[58] *CopterControl / CC3D / Atom Hardware Setup — LibrePilot/OpenPilot Wiki 0.1.4 documentation*. [Online]. Available: https://opwiki.readthedocs.io/en/latest/user%7B%5C_%7Dmanual/cc3d/cc3d.html (visited on 09/20/2018).

[59] *Buy OpenPilot CC3D Flight Controller*. [Online]. Available: https://www.banggood.com/OpenPilot-CC3D-Flight-Controller-STM32-32-bit-Flexiport-p-937044.html (visited on 09/20/2018).

[60] *OpenPilot CC3D Flight Controller (Right Angle Pins)*. [Online]. Available: https://www.getfpv.com/openpilot-cc3d-flight-controller-right-angle-pins-3876.html (visited on 09/20/2018).

[61] *ArduPilot :: About*. [Online]. Available: http://ardupilot.org/about (visited on 09/20/2018).

[62] *Optional Hardware — Copter documentation*. [Online]. Available: http://ardupilot.org/copter/docs/common-optional-hardware.html (visited on 09/20/2018).

[63] *Introduction · MAVLink Developer Guide*. [Online]. Available: https://mavlink.io/en/ (visited on 09/20/2018).

[64] *Trademark — Dev documentation*. [Online]. Available: http://ardupilot.org/dev/docs/trademark.html (visited on 09/20/2018).

[65] *Introduction · PX4 User Guide*. [Online]. Available: https://docs.px4.io/en/ (visited on 09/20/2018).

[66] *Advanced flight control - dRonin*. [Online]. Available: https://dronin.org/ (visited on 09/20/2018).

[67] L. Meier, T. Gubler, D. Agar, J. Oes, B. Küng, D. Sidrane, A. Babushkin, Px4dev, M. Charlebois, R. Bapst, A. D. Antener, J. Goppert, A. Tridgell, P. Riseborough, D. Mannhart, M. Whitehorn, M. Grob, S. Wilks, K. Mohammed, S. Smeets, P. Kirienko, ChristophTobler, JohanJansen, D. Gagne, B. Siesta, J. R. de Souza, L. D. Marchi, F. Achermann, J. Lecoeur, and S. Guscetti, "PX4/Firmware: v1.8.0 Stable Release", Jun. 2018. DOI: 10.5281/ZENODO.1292429. [Online]. Available: https://zenodo.org/record/1292429%7B%5C#%7D.W6KBNf5KiRc.

[68] *Technology - PX4 Open Source Autopilot*. [Online]. Available: http://px4.io/technology/ (visited on 09/20/2018).

[69] M. Lyle, *dRonin 2016-01-20 release ("renatus")*, 2016. [Online]. Available: https://github.com/dronin/dRonin/releases/tag/Release-20160120.3.

[70] *Advanced flight control - dRonin*. [Online]. Available: https://dronin.org/ (visited on 09/20/2018).

[71] TauLabs, *Development of the UAVTalk Protocol*, 2014. [Online]. Available: https://github.com/TauLabs/TauLabs/wiki/Development-UAVTalk-Protocol.

[72] *QGC - QGroundControl - Drone Control*. [Online]. Available: http://qgroundcontrol.com/ (visited on 09/21/2018).

[73] *Virtual Joystick (PX4) · QGroundControl User Guide*. [Online]. Available: https://docs.qgroundcontrol.com/en/SettingsView/VirtualJoystick.html (visited on 09/21/2018).

[74] *SiK Telemetry Radio — Copter documentation*. [Online]. Available: http://ardupilot.org/copter/docs/common-sik-telemetry-radio.html (visited on 09/21/2018).

[75] *Mobile App | DroneDeploy*. [Online]. Available: https://www.dronedeploy.com/product/mobile/ (visited on 09/21/2018).

[76] *Enterprise Drone Mapping Software | DroneDeploy*. [Online]. Available: https://www.dronedeploy.com/product/platform/ (visited on 09/21/2018).

[77] *Live Map | DroneDeploy*. [Online]. Available: https://www.dronedeploy.com/product/live-map/ (visited on 09/21/2018).

[78] *Supported Drones*. [Online]. Available: https://support.dronedeploy.com/docs/supported-drones (visited on 09/21/2018).

[79] *Industry Agnostic Drone Technology Applications*. [Online]. Available: https://www.precisionhawk.com/industries (visited on 09/21/2018).

[80] *PrecisionFlight | UAV & Drone Flight Planner*. [Online]. Available: https://www.precisionhawk.com/precisionflight (visited on 09/21/2018).

[81] *PrecisionFlight Pro | UAV Tracking & Flight Planner for Professionals*. [Online]. Available: https://www.precisionhawk.com/precisionflight-pro (visited on 09/21/2018).

[82] *FlytBase: Build Smart and Scalable Drone Applications*. [Online]. Available: https://flytbase.com/ (visited on 09/21/2018).

[83] *FlytOS: Operating System for Drones*. [Online]. Available: https://flytbase.com/flytos/ (visited on 09/21/2018).

[84] J. Besada, L. Bergesio, I. Campaña, D. Vaquero-Melchor, J. López-Araquistain, A. Bernardos, J. Casar, J. A. Besada, L. Bergesio, I. Campaña, D. Vaquero-Melchor, J. López-Araquistain, A. M. Bernardos, and J. R. Casar, "Drone Mission Definition and Implementation for Automated Infrastructure Inspection Using Airborne Sensors", *Sensors*, vol. 18, no. 4, p. 1170, Apr. 2018, ISSN: 1424-8220. DOI: 10.3390/s18041170. [Online]. Available: http://www.mdpi.com/1424-8220/18/4/1170.

[85] A. Williams and O. Yakimenko, "Persistent mobile aerial surveillance platform using intelligent battery health management and drone swapping", in *Proceedings - 2018 4th International Conference on Control, Automation and Robotics, ICCAR 2018*, IEEE, Apr. 2018, pp. 237–246, ISBN: 9781538663387. DOI: 10.1109/ICCAR.2018.8384677. [Online]. Available: https://ieeexplore.ieee.org/document/8384677/.

[86] N. Yamamoto and K. Naito, "Proposal of Continuous Remote Control Architecture for Drone Operations", in, Springer, Cham, Jun. 2019, pp. 64–73. DOI: 10.1007/978-3-319-92231-7_7. [Online]. Available: http://link.springer.com/10.1007/978-3-319-92231-7%7B%5C_%7D7.

[87] A. M. de Oca, L. Arreola, A. Flores, J. Sanchez, and G. Flores, "Low-cost multispectral imaging system for crop monitoring", in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, Jun. 2018, pp. 443–451, ISBN: 978-1-5386-1354-2. DOI: 10.1109/ICUAS.2018.8453426. [Online]. Available: https://ieeexplore.ieee.org/document/8453426/.

[88] J. T. Amenyo, D. Phelps, O. Oladipo, F. Sewovoe-Ekuoe, S. Jadoonanan, S. Jadoonanan, T. Tabassum, S. Gnabode, T. D. Sherpa, M. Falzone, A. Hossain, and A. Kublal, "MedizDroids Project: Ultra-low cost, low-altitude, affordable and sustainable UAV multicopter drones for mosquito vector control in malaria disease management", in *Proceedings of the 4th IEEE Global Humanitarian Technology Conference, GHTC 2014*, IEEE, Oct. 2014, pp. 590–596, ISBN: 9781479971930. DOI: 10.1109/GHTC.2014.6970343. [Online]. Available: http://ieeexplore.ieee.org/document/6970343/.

[89] *DJI - The World Leader in Camera Drones/Quadcopters for Aerial Photography*. [Online]. Available: https://www.dji.com/flame-wheel-arf (visited on 09/24/2018).

[90] *Buy E310 420S ESCs - DJI Store*. [Online]. Available: https://store.dji.com/product/e310-420s-esc (visited on 09/24/2018).

[91] DJI, *DJI - The World Leader in Camera Drones/Quadcopters for Aerial Photography*, 2018. [Online]. Available: https://www.dji.com/e310 (visited on 09/24/2018).

[92] *Original DJI E310 420S ESC para RC Multicopter - RcMoment.com*. [Online]. Available: https://www.rcmoment.com/pt/p-rm6052.html (visited on 09/24/2018).

[93] *Buy E310 2312 Motor (CCW&CW) - DJI Store*. [Online]. Available: https://store.dji.com/product/e310-motor (visited on 09/24/2018).

[94] *Buy 9450 Self-tightening Propellers (Composite Hub, Gray) - DJI Store*. [Online]. Available: https://store.dji.com/product/9450-plastic-hub-props-gray (visited on 09/24/2018).

[95] *DJI E310 Tuned Propulsion System Quadcopter Build Your Own Drone*. [Online]. Available: https://www.buildyourowndrone.co.uk/dji-e310-tuned-propulsion-system-quadcopter (visited on 09/24/2018).

[96] *DJI E300, E310 & E305 9450 Self Tightening Propellers Grey Build Your Own Drone*. [Online]. Available: https://www.buildyourowndrone.co.uk/dji-e310-e305-9450-self-tightening-grey-propellers (visited on 09/24/2018).

[97] *NEO-M8 series | u-blox*, 2017. [Online]. Available: https://www.u-blox.com/en/product/neo-m8-series (visited on 09/24/2018).

[98] *Magnetic Sensors and Transducers*. [Online]. Available: https://aerospace.honeywell.com/en/products/navigation-and-sensors/magnetic-sensors-and-transducers (visited on 09/24/2018).

[99] *Ublox Neo-M8N GPS with Compass*. [Online]. Available: https://store.nerokas.co.ke/index.php?route=product/product%7B%5C&%7Dproduct%7B%5C_%7Did=1297 (visited on 09/24/2018).

[100] *FlySky FS-i6*. [Online]. Available: http://www.flysky-cn.com/products%7B%5C_%7Ddetail/productId=36.html (visited on 09/24/2018).

[101] *FlySky FS-iA6B*. [Online]. Available: http://www.flysky-cn.com/products%7B%5C_%7Ddetail/productId=51.html (visited on 09/24/2018).

[102] *DJI - The World Leader in Camera Drones/Quadcopters for Aerial Photography*. [Online]. Available: https://www.dji.com/e310/spec (visited on 09/24/2018).

[103] *Raspberry Pi 2 Model B - Raspberry Pi*. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ (visited on 09/24/2018).

[104] *Download Raspbian for Raspberry Pi*. [Online]. Available: https://www.raspberrypi.org/downloads/raspbian/ (visited on 09/24/2018).

[105] *Arduino Nano*. [Online]. Available: https://store.arduino.cc/usa/arduino-nano (visited on 09/24/2018).

[106] *LibrePilot – Open – Collaborative – Free*. [Online]. Available: https://www.librepilot.org/site/index.html (visited on 09/24/2018).

[107] *Tau Labs*. [Online]. Available: http://taulabs.org/ (visited on 09/24/2018).

[108] DRonin, *UAV Object Definition files*. [Online]. Available: https://github.com/d-ronin/dRonin/tree/next/shared/uavobjectdefinition.

[109] ——, *Coordinate Conversion Methods*. [Online]. Available: https://github.com/d-ronin/dRonin/blob/3a866925d6148e33e0620c94c5940413574a7247/ground/gcs/src/libs/utils/coordinateconversions.cpp.

[110] *InfluxDB | The Time Series Database in the TICK Stack | InfluxData*. [Online]. Available: https://www.influxdata.com/time-series-platform/influxdb/ (visited on 09/24/2018).

[111] *Grafana - The open platform for analytics and monitoring*. [Online]. Available: https://grafana.com/ (visited on 09/24/2018).

[112] B. Areias and S. Sargento, "Modular Event-Driven Unmanned Aerial Vehicles Control Platform", PhD thesis, Universidade de Aveiro, 2017.

[113] *Node.js*. [Online]. Available: https://nodejs.org/en/ (visited on 09/24/2018).

[114] *AngularJS — Superheroic JavaScript MVW Framework*. [Online]. Available: https://angularjs.org/ (visited on 09/24/2018).

[115] *Bootstrap · The most popular HTML, CSS, and JS library in the world.* [Online]. Available: http://getbootstrap.com/ (visited on 09/24/2018).

[116] *Leaflet - a JavaScript library for interactive maps*. [Online]. Available: https://leafletjs.com/ (visited on 09/24/2018).

[117] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol", DOI: 10.21105/joss.00265. [Online]. Available: http://joss.theoj.org/papers/10.21105/joss.00265.

[118] *Eclipse Foundation | The Eclipse Foundation*. [Online]. Available: https://www.eclipse.org/org/foundation/ (visited on 09/24/2018).

[119] *MQTT*. [Online]. Available: https://mqtt.org/ (visited on 09/24/2018).

[120] *Messaging that just works — RabbitMQ*. [Online]. Available: https://www.rabbitmq.com/ (visited on 09/24/2018).

[121] *Home | AMQP*. [Online]. Available: https://www.amqp.org/ (visited on 09/24/2018).

[122] *Drone Mapping Software - OpenDroneMap*. [Online]. Available: https://www.opendronemap.org/ (visited on 09/24/2018).

[123] *GoPro Official Website - Capture + share your world - HERO3*. [Online]. Available: https://gopro.com/update/hero3 (visited on 09/24/2018).

[124] *Convert, Edit, Or Compose Bitmap Images @ ImageMagick*. [Online]. Available: https://www.imagemagick.org/script/index.php (visited on 09/24/2018).

[125] *Hugin Overview*. [Online]. Available: http://hugin.sourceforge.net/tutorials/overview/en.shtml (visited on 09/24/2018).

[126] A. Gibson, "De-barrel distortion", [Online]. Available: http://im.snibgo.com/debarrel.htm.

[127] *MeshLab*. [Online]. Available: http://www.meshlab.net/ (visited on 09/24/2018).

[128] "SprintIR 6S 5% to 100% CO2 Sensor Ultra-fast response Carbon Dioxide Sensor", Tech. Rep. [Online]. Available: http://www.co2meters.com/Documentation/Datasheets/DS-GC-0028-SprintIR6S.pdf.