Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2018

Bruno Miguel Sousa Areias

Plataforma Modular de Controlo de Veículos Aéreos não Tripulados Baseada em Eventos
Modular Event-Driven Unmanned Aerial Vehicles Control Platform

Universidade de Aveiro Departamento de
Electrónica, Telecomunicações e Informática
2018

Bruno Miguel
Sousa Areias

Plataforma Modular de Controlo de Veículos Aéreos não Tripulados Baseada em Eventos
Modular Event-Driven Unmanned Aerial Vehicles Control Platform

"It is not the critic who counts; not the man who points out how the strong man stumbles, or where the doer of deeds could have done them better. The credit belongs to the man who is actually in the arena, whose face is marred by dust and sweat and blood; who strives valiantly; who errs, who comes short again and again, because there is no effort without error and shortcoming; but who does actually strive to do the deeds; who knows great enthusiasms, the great devotions; who spends himself in a worthy cause; who at the best knows in the end the triumph of high achievement, and who at the worst, if he fails, at least fails while daring greatly, so that his place shall never be with those cold and timid souls who neither know victory nor defeat."

— Theodore Roosevelt

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2018

Bruno Miguel Sousa Areias

Plataforma Modular de Controlo de Veículos Aéreos não Tripulados Baseada em Eventos
Modular Event-Driven Unmanned Aerial Vehicles Control Platform

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica da Professora Doutora Susana Sargento, Professora Associada com Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e co-orientação científica do Professor Doutor José Maria Fernandes, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	**António José Ribeiro Neves** Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
	Rolando da Silva Martins Professor Auxiliar Convidado do Departamento de Ciência de Computadores da Faculdade de Ciências da Universidade do Porto
	José Maria Amaral Fernandes Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos / acknowledgements

O termino de uma dissertação é o marco final de mais uma etapa da vida académica, simboliza o culminar de todo um percurso de absorção de conhecimento. Percurso esse que para mim é com grande pesar que não poderei partilhar a alegria de mais um objetivo concluído com os meus pais, que em muito se esforçaram para que este dia viesse a ser possível.
Contudo quero agradecer a todos os que no dia a dia foram contribuindo para a conclusão desta etapa. Em particular, quero começar por agradecer à minha tia e namorada pelo apoio, motivação e paciência diária, sem elas terminar esta fase teria sido impossível!
A todos os meus amigos, que no dia-a-dia foram incentivando e acompanhando, em particular: André Rainho, Paulo Oliveira, Daniela Pinto, João Barbosa, Paulo Barbosa, Sara Oliveira, Jorge Pereira, Pedro Silva, Luis Catarino. A todos um muito obrigado!
Finalmente, aos que diretamente participaram no desenvolvimento deste trabalho, em particular: à Professora Doutora Susana Sargento pelo aconselhamento, apoio e pelas horas que dedicou à correção deste trabalho; ao Professor Doutor José Maria Fernandes pela contínua força para o término deste trabalho; ao Nuno Paulo pelas incessantes horas de trabalho dedicadas ao desenvolvimento desta plataforma; ao Doutor Lucas Guardalben pelo apoio e aconselhamento contínuo durante todo o desenvolvimento; ao Doutor André Reis pelo apoio na escrita deste documento; e finalmente ao grupo NAP em geral.
A todos, um muito sincero obrigado!

Resumo

Hoje em dia, os drones estão-se a tornar cada vez mais comuns nas nossas vidas diárias. Com a agilidade, acessibilidade e diversidade dos drones, eles são uma excelente plataforma para transportar dispositivos (p.ex., conjunto de sensores, câmeras, unidades computacionais de pequena dimensão). Assim sendo, são uma excelente ferramenta para tarefas como: explorar e estudar áreas perigosas, monitorizar campos de agricultura, ajudar na deteção e combate de incêndios ou vigiar multidões. Para realizar tais tarefas, ferramentas de automação e integração são essenciais, para que o desenvolvimento se concentre na própria aplicação e não nos problemas relacionados com a integração e automação do sistema do drone. Os drones atualmente disponíveis não são capazes de lidar com tais complexidades de forma tão transparente. Por exemplo, certos níveis de automação são já possíveis, mas requerem hardware e software específicos do fornecedor; no que toca a integração, alguns já supportam SDK ou API para interagir com o drone, mas mais uma vez com a inconveniência de necessitar de conhecimento prévio sobre os sistemas dos drones.

Para responder a estas necessidades, esta tese propõe uma plataforma modular de controlo baseada em eventos para abstrair os processos de automação e integração da complexidade subjacentes aos drones. Enquanto que a plataforma permite que as aplicações controlem e interajam com os drones, a sua complexidade é resolvida dentro da plataforma, simplificando o processo de integração. Além disso, com a plataforma proposta, a automação e funcionalidades do drone podem ser estendidas para estender as funcionalidades de drones mais limitados.

A plataforma desenvolvida foi testada em diferentes cenários, tanto ao nível das suas funcionalidades como ao nível da análise de desempenho. Os resultados mostram que, além das funcionalidades suportadas, a plataforma consegue suportar o controlo e gestão de pelo menos até 64 drones em simultâneo sem ter modificações significativas nos atrasos de comunicação e throughput.

Abstract

Nowadays, drones are becoming more common in our daily lives. Since drones are agile, affordable and diverse, they make an excellent platform to carry devices around (e.g., sensor arrays, cameras, small computers). With these capabilities, they become an excellent tool for tasks like: explore and study hazardous areas, agriculture monitoring, help on the detection and fight in fires, and crowd surveillance. To perform such tasks, automation and integration tools are a must have, so that the development can focus on the application itself and not on the issues related with the integration and automation of the drone system. Current available drones are not capable of properly handling such complexities in a seamless way. For instance, some levels of automation are already possible, but require vendor specific hardware and software; for integration, some offer SDK or API interactions, but once again with the inconvenience of requiring extensive knowledge about drone systems to implement.

To address these issues, this thesis proposes a modular event-driven control platform to abstract automation and integration processes from the underlying complexities of the drones, while the platform lets the applications control and interact with the drones. The drones' complexities are resolved within the platform, therefore simplifying integration process. Moreover, with the proposed platform, drone automation and functionality can be extended across distinct brands of drones, while some may already support some features, others may not, and in that case the platform modules may intervene to extend the features of less capable drones.

The developed platform has been tested in different scenarios, such as in terms of its functionalities and in terms of performance analysis. The results show that, besides the supported functionalities, the platform is able to handle the control and management of at last 64 simultaneous drones without significant changes in the communication delays and throughput.

Contents

List of Figures

List of Tables

Acronyms

ADC Analog-to-Digital Converter.

AMQP Advanced Message Queuing Protocol.

API Application Programming Interface.

CAN Controller Area Network.

CLI Command Line Interface.

CPU Central Processing Unit.

DIY Do It Yourself.

ESC Electronic Speed Controller.

EU European Union.

FC Flight Controller.

FDR Flight Data Recorder.

GCS Ground Control Station.

GNSS Global Navigation Satellite System.

GPIO General Purpose IO.

GPS Global Position System.

HAT Hardware Attached on Top.

I2C Inter-Integrated Circuit.

IMU Inertial Measurement Unit.

IP Internet Protocol.

JSON	JavaScript Object Notation.
LXC	Linux Containers.
MCU	Microcontroller Unit.
MQTT	Message Queuing Telemetry Transport.
OS	Operating System.
RAM	Random Access Memory.
REST	Representational State Transfer.
RTK	Real Time Kinematic.
RTOS	Real-Time Operating System.
SAR	Search and Rescue.
SBC	Single Board Computer.
SD Card	Secure Disk Card.
SDK	Software Development Kit.
SOA	Service-Oriented Architecture.
SPI	Serial Peripheral Interface.
UART	Universal Asynchronous Receiver-Transmitter.
UAV	Unmanned Aerial Vehicle.
USART	Universal Synchronous Asynchronous Receiver-Transmitter.
USB	Universal Serial Bus.
VM	Virtual Machine.
VTOL	Vertical Take-Off and Landing.

Chapter 1

Introduction

1.1 Context

Unmanned Aerial Vehicles (UAVs), commonly known as drones, have long been used by military forces in diverse scenarios (e.g.: Aerial Surveillance, High-risk Bombing). For the last few years, interest in aerial drones has grown, both within the scientific community, and with the general public.

That interest extends to a wide variety of areas (e.g.: cinematography, journalism, security forces), from professional to leisure purposes. Drones can now be used in virtually all aspects of our daily lives as a result of their versatility and variety in shapes, sizes and capabilities.

The interest in drones continues to grow in areas like wildfire or crop monitoring, catastrophe recovery or search and rescue. In these areas, drones represent an effective tool to perform repetitive tasks like gathering aerial images at regular intervals, or transporting cargo between two points without the need for large-scale infrastructure in place (e.g., roads and bridges). However, drone automation and integration tools still require further development.

Although some levels of automation (e.g.: auto-land/takeoff, hover, go-to location) already exist, they lack awareness of the drones' surrounding environment and threats, and therefore they are effectively flying blind. As a consequence, tools associated with the currently available drones require a person to either pilot or observe the entire flight to ensure safety of both the aircraft and its surroundings.

In order to pilot or observe the flight of a drone, a 2.4 GHz radio controller or a Ground Control Station (GCS) software is required. In most of the cases, the usage of GCS still requires a radio controller to be present and active, once again for safety purposes.

Moreover, GCSs are in most cases vendor or technology specific, which limits integration to a handful of vendors or technologies mostly supporting a single drone being controlled at a time and requiring a constant dedicated direct wireless connection to the Flight Controller (FC), generally through the means of a Universal Serial Bus (USB) radio (commonly known as a 'dongle') specific to the underlying communication protocol.

In recent years, FCs have extended their interaction capabilities by allowing external devices to interact with them through Application Programming Interfaces (APIs). These APIs are still a limiting factor to integrability due to their reliance on physical connectivity between entities.

Given these shortcomings, the present work aims to develop a modular platform where each module is single-handedly responsible for monitoring, controlling or extending the functionality of a drone. Each module abstracts the underlying complexity and requirements from other modules, simplifying interactions and improving integrability. Finally, we take advantage of the improved integration and abstractions of the platform to extend the automation capabilities of drones.

1.2 Objectives

The main purpose of this dissertation is to provide a platform that abstracts and decouples the various UAV control processes, therefore building an effective tool to improve the integration of drones with other applications, functionalities and mechanisms.

To reach this goal, we go through the following steps:

- Develop a solution that enables the decoupling of generic applications from the underlying communications to the FC;
- Provide an abstraction from the underlying API implementation;
- Develop proof-of-concept applications that take advantage of the new platform to control and monitor a drone.

1.3 Contributions

As a result of the work presented in this thesis, a modular platform which abstracts and simplifies drone interactions has been created. The usage of brokers at the communications core of the platform abstracts modules from the underlying communications' details, allowing them to focus solely on publishing and consuming messages to and from their respective channels.

Four applications were developed and integrated to take advantage of the platform: a GCS-like web application which allows for high level control and monitoring of drones;

a mobile web-based control application which allows for high-level control of drones; an application that streams and processes sensor data from drone flights in real time which is capable of computing new flight variables, detecting and notifying the platform of abnormal/relevant drone behaviors; and a web-based data analysis application which allows to monitor and review historical sensor data from drone flights.

Finally, two articles were published with results obtained from the study presented in this document.

The first one was published in INForum 2017, in October 12th and 13th, in the communication category, entitled "Automated Flying Drones Platform for Automatic and Remote Sensing". It was approved and presented through oral presentation and poster.

The second one was published in VEHITS 2018, in March 2018, as a position paper, entitled "Towards an Automated Flying Drones Platform". It was approved and presented through oral presentation.

This platform has been presented in RTCM (Rede Temática de Comunicações Móveis) seminar in June 2018 with great success.

1.4 Document Structure

This thesis is organized as follows.

- **Chapter 1** contains a generalized overview of the work.
- **Chapter 2** contains the current state of the art in the field of FCs, autopilot software, GCSs and drone platforms, and cloud telemetry platforms.
- **Chapter 3** presents a proposal for the platform architecture.
- **Chapter 4** details the implementation; integration choices are presented and exemplified.
- **Chapter 5** presents the platform use cases and performance evaluation.
- **Chapter 6** concludes the work in this document and proposes future work to be done.

Chapter 2

Related Work

This chapter starts by exposing a brief history about drones, how drones can be improved, and a resume about areas where drones can make a significant difference.

Following this brief introduction, Flight Controllers (FCs) are explained, along with their general capabilities and functionalities, and then several state of the art FCs are discussed.

A section about community autopilot software follows, starting with an introduction about their importance and the main differences about the two distinct implementation approaches, followed by a discussion about several state of the art autopilot software.

To introduce the tools for drone task automation, the next section focuses on the exploration and discussion of several state of the art Ground Control Stations (GCSs) or drone platforms. It starts with an introduction about how automation can be performed in state of the art drones, and then it follows the discussion of their differences and capabilities.

After the exposure of the drones, their key parts and the set of automation tools currently available, a discussion follows on real world applications where drones are already used to gather value added information for crops, aerial surveillance, or humanitarian aid.

Finally a discussion about several state of the art cloud based telemetry platforms, which is a key part of the proposed platform in this work, is presented.

2.1 Unmanned Aerial Vehicles

Unmanned Aerial Vehicles, commonly known as drones, trace their origins back to the First World War, where they were initially used to carry heavy payloads of explosives across long distances. Over the years, successive technological evolutions contributed to the refinement of Unmanned Aerial Vehicle (UAV) technology, and its accessibility to the general public. The applications for aerial drones have since grown immensely by virtue of their versatility, reliability, and ease of use [1].

Aerial drones are particularly well-suited for hazardous tasks like criminal pursuit and eruption surveillance, or wildfire observation without exposure of human lives to potential threats. Moreover, with automation capabilities, aerial drones are well-suited for repetitive tasks like aerial surveillance and image gathering, or the transportation of cargo in areas without supporting ground infrastructure. Such conceptually simple tasks are crucial for several civil application scenarios, such as [2, 3]:

Science and Research

- Meteorological research
- Volcanic eruptions – surveillance, tracking and monitoring ash clouds
- Agriculture – mapping plant growth and issues, moisture levels, crop yields
- Measuring nuclear contamination
- Forestry and Natural Resources Management

Public Safety

- Emergency communication network aid
- Monitoring of natural disasters: landslides, tsunamis, wild fires, volcanic eruptions, flooding, storms, hurricanes, avalanches, tornadoes
- Searches for missing people
- Post-disaster-relief operations

Structure inspections

- Oil pipeline inspection
- Solar panel inspections
- Power line / cable inspections
- Bridge inspections
- Monument inspections

Across the literature, aerial drones are classified in several aspects. Due to the lack of standards, such classifications are still relative to the author, country or field of activity; in general, two classifications are identical: type and size.

The most common types of aerial drones are: fixed-wing, helicopter and multi-copter; but there are others less common like hybrids, and zeppelins or balloons. Regarding size, aerial drones are classified as: very small, small, medium and large; where very small is informally subdivided into mini and micro [4].

2.2 Flight Controllers

The heart of any drone is the FCs; it is the device responsible for performing control tasks and stabilization adjustments required to keep the vehicle in a steady and balanced state, while also enabling external control of said vehicle. It is composed of all the necessary sensors and processors to achieve a stable flight, and are operated by an Autopilot software which will be detailed in the chapter 2.3.

In general, FCs require at least an Inertial Measurement Unit (IMU), barometer, magnetometer, Microcontroller Unit (MCU) and several communication ports to interact with the engines or servos. While stable flight is achievable with these sensors, it limits the drone to indoor flight or requires a human operator to correct movements from external forces like wind, and gusts or currents. Moreover, the built-in magnetometer is often prone to magnetic interference from auxiliary equipment, resulting in erratic behavior like toilet bowl effect.

Counteracting these undesired effects, an external magnetometer and Global Position System (GPS) module is often used, and placed in an isolated area without interference. With the advantage of the GPS, FCs can correct movements from external forces which is required for safe outdoor operation of drones.

Control is commonly provided by the drone user himself who triggers the desired adjustments to the aircraft's throttle, pitch, roll, and yaw using sticks on a separate radio controller. In turn, a radio receiver aboard the drone transmits these inputs to the flight controller by means of a physical connection, as depicted in Figure 2.1.

Several FCs allow for more advanced handling: as opposed to directly inputting fine control parameters, the user can request simple tasks such as autonomous take-off and landing, waypoint flight, and return to the take-off location. More complex tasks such as planning and automatic path following are also possible in some state-of-the-art flight controllers. Most of these tasks rely heavily on GPS-assisted control and an external GCS application to issue the desired tasks, Figure 2.1.

We will now discuss FC offerings from three major vendors: DJI, MicroPilot and Emlid; and three other offerings from community design: PixHawk, OpenPilot Revolution, and Sparky.

2.2.1 Vendor FCs

We will now discuss FC offerings from three major vendors: DJI, MicroPilot and Emlid.

DJI

DJI, a Chinese vendor, sells UAVs that are affordable, reliable, and easy to use. DJI quickly established itself as the number-one brand in the field of aerial drones. After success with their initial offering of UAVs, FCs and gimbals, their range now includes drone cameras and camera stabilizers as well [5].

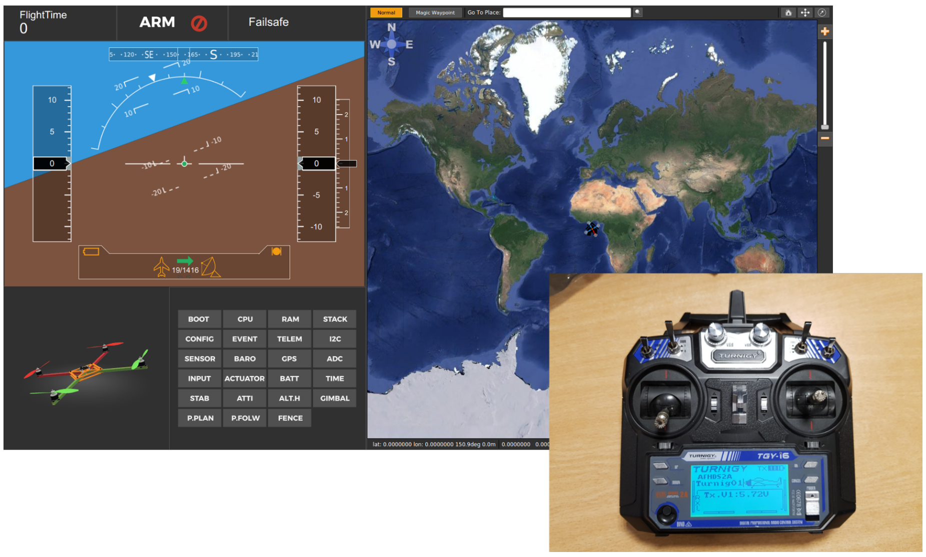


Figure 2.1: Example of a GCS and a Remote Controller

The latest advancements from DJI in UAVs and FCs include the Matrice UAV Series and the A3/N1/N3 FC Series. The Matrice UAV Series is aimed at resilient and adaptable UAV frames on top of which users can add external sensors as needed. The A3/N1/N3 FC Series extend UAV capabilities by allowing greater precision and improved SDK tools for programmatic flight parameter interaction and control [6, 7, 8].

The A3 FC is the current flagship of DJI, available in Basic and Pro versions, and retailing for ∼1000€ and ∼1600€, respectively. The A3 Pro ships with a full set of sensors and parts, while A3 Basic features only a minimal set of parts that can be expanded as desired. Its major features include Real Time Kinematic (RTK) positioning, triple redundant GPS and IMU sensors, and Software Development Kit (SDK) support [7].

The N3 FC is a more cost-effective controller, similar to the A3 FC but retailing for ∼350€. The N3 features dual redundant IMU sensors and SDK support, and can be upgraded with an A3 upgrade kit to enable triple and dual redundancy for IMU and GPS sensors, respectively [8].

DJI offers robust solutions of UAVs and FCs, and although their offerings retail for higher than the competition, the ease of usage and reliability of their products is typically superior. On the other hand, FCs developed by DJI run closed-source firmware, limiting their use as a platform for research.

MicroPilot

MicroPilot is a Canadian company focused on high-end FCs, and several companion sensors (magnetometer, ultrasonic altitude sensor). MicroPilot started its FC product line in 2003, with one 28g FC for fixed-wing UAVs, which rivaled any other on the market at the time. One year later, MicroPilot launched an SDK for their FC products, allowing for customization of flight parameters and functionalities. In 2007, MicroPilot launched the MP2128^{3X}, their first FC that features Vertical Take-Off and Landing (VTOL) capable UAVs [9, 10, 11].

The latest FCs from MicroPilot are the MP2128^{g2} and MP2128^{HELI2}. Although they are similar in technical specifications, and both have VTOL capabilities, the first one does not support helicopter UAVs. For simplification, we will look at the MP2128 FCs as one and include their MP2028^{g2}, which is has the second best technical specifications [12].

MP2128^{HELI2} is the current flagship FC from MicroPilot, retails for ∼8000€. MP2128^{HELI2} features high quality IMU and GPS for increased precision, up to 24 servos or motors, support for Controller Area Network (CAN) and RTK, and autonomous stall and tumble recovery. The MP2128^{HELI2} ships with a full set of sensors and weights 40g [12].

MP2028^{g2} is an FC from MicroPilot that has a lower quality grade than MP2128^{HELI2} which retails for ∼3500€. Like MP2128^{HELI2}, MP2028^{g2} features high quality IMU and GPS for increased precision with a higher error rate than MP2128^{HELI2}, up to 16 servos or motors, and lacks the support for CAN, RTK, or the autonomous stall and tumble recovery. The MP2028^{g2} ships with a basic set of sensors and weights 24g [12].

Also from MicroPilot, the MP2128^{3X}, although it is not an FC, it is an enclosure that combines three MP2128^{HELI2}. MP2128^{3X} offers a fully triple FC redundancy where each FC is completely autonomous and can assume control over any of the other enclosed FCs. It ships for ∼25000€ with a triple set of sensors and FCs.

MicroPilot reveals a high-end quality on parts and manufacture for their FCs, but retails to the highest price of all the competitors. FCs offered by MicroPilot are a result of the long experience of the company and the partnership with their major consumers like RAF, NASA, and Northrop Grumman [9, 13]. While MicroPilot FCs are unmatched in a quality standpoint, much like DJI, their FCs run closed-source firmware, limiting their use as a platform for research.

Emlid

Emlid is a Russian company focused on precision RTK Global Navigation Satellite System (GNSS) and FCs for UAVs. Emlid started out with two FCs (Navio+ and Navio2) and an RTK (Reach RTK), and now sell a new FC called EDGE. Their FCs can be interacted with via a Python Application Programming Interface (API), and are based on an open-source project (ArduPilot) thereby allowing full firmware customization [14, 15, 16].

Navio2 is an FC focused on the core requirements for a UAV. It supports up to 12 servos/motors, plus Universal Asynchronous Receiver-Transmitter (UART), Inter-Integrated Circuit (I2C), and Analog-to-Digital Converters (ADCs) for interfacing with sensors. A dual IMU is built-in for increased precision. On Navio2, the Ardupilot software runs directly on a Raspberry Pi over an Unix-based system, and its sensors are built on a Hardware Attached on Top (HAT) that connects directly to the Pi's General Purpose IOs (GPIOs). The choice of running a full operating system enables full networking capabilities and allows for an easier integration with external sensors and control platforms [15].

EDGE is the current flagship FC from Emlid, retailing for ∼700€. EDGE features dual IMU and power supplies for increased precision and redundancy, FullHD video streaming, a dedicated Central Processing Unit (CPU) for navigation, and a second CPU operating a Linux system. The platform provides a pair of Universal Serial Bus (USB), CAN, and UART ports, plus support for up to 12 servos or motors [16].

Emlid offers a robust open-source platform with the autopilot software running on top of a full operating system, which allows for greater flexibility and customization at all levels. Navio2 FCs are advertised as the more general purpose platform, while the EDGE FCs are aimed at providing FullHD video streaming from the drones.

2.2.2 Community FCs

We will now discuss FC offerings from community design: PixHawk, OpenPilot Revolution, and sparky.

Pixhawk 2

Pixhawk was developed in parallel with the PX4 autopilot software by a community of students at ETH in Zürich [17]. PX4 has since then became a collaborative open-source project to the development of a complete end-to-end platform for UAVs [18].

Pixhawk 2 features five UART, two CAN, one I2C, one Serial Peripheral Interface (SPI), three ADCs, one micro USB, support for Secure Disk Card (SD Card) and RTK, redundant power supply, and up to fourteen servos or motors. Pixhawk 2 ships with an ARM Cortex-M4F CPU, triple redundant IMU, double redundant barometer, and PX4 autopilot software (detailed in chapter 2.3) for ∼300€ [19, 20].

Similar to Emlid, PX4 autopilot software runs on top of a full operating system; Pixhawk is a reliable and robust solution as a FC. Its redundancy, reliability and the amount of communication ports suggests a great level of attention to the detail in the development of Pixhawk 2 [21]. Moreover, the open-source nature of Pixhawk 2 is a good solution for a research platform.

OpenPilot Revolution

Revolution was developed by the OpenPilot community. OpenPilot was a group of volunteers with the focus on development of an open-source autopilot firmware capable of handling fixed-wing, multi-rotor, and helicopter UAVs. To host the OpenPilot autopilot firmware, the group developed their own FC board, entitled as ”Revolution” [22, 23, 24].

Revolution features one Universal Synchronous Asynchronous Receiver-Transmitter (USART), one I2C, one Serial, one micro USB, redundant power supply, up to six servos or motors, and a telemetry radio transmitter. Revolution ships with an STM32F4-series MCU, IMU, barometer, and OpenPilot autopilot firmware (detailed in chapter 2.3) for ∼50€ [25].

Revolution is affordable, compact, open-source, and its autopilot firmware can be changed later on. Therefore, Revolution represents a good option to be used as a platform for research.

TauLabs Sparky 2.0

Sparky 2.0 was developed by the TauLabs community. TauLabs was a volunteer group of individuals interested in UAV technology, focused on the development of high quality open-source firmware for FCs. TauLabs participated in the early beginnings of OpenPilot, eventually forked for greater degree of freedom [26].

Sparky 2.0 features three USART, one I2C, one CAN, one micro USB, up to six servos or motors, telemetry radio transmitter, and an internal flash memory that can be used for logging. Sparky 2.0 ships with an STM32F4-series MCU, IMU, barometer, and TauLabs autopilot firmware (detailed in chapter 2.3) for ∼40€ [27].

In comparison to Revolution, Sparky 2.0 is similar and slightly larger in size, which also makes Sparky 2.0 a good option to be used as a platform for research.

2.3 Community Autopilot Software

Autopilot is the term called to the software running on the hardware of each FC: through the combination of the FC hardware and the autopilot software, the UAV is capable of a stable flight.

The capabilities of each autopilot software are specific: while some can achieve certain levels of complexity in autonomous flight, some others may only provide hover or stabilization capabilities. In general there are two different types of autopilot software, the firmware based autopilots which run directly on the FC board, or the software based autopilots which run on top of an operating system.

Firmware based autopilots tend to be smaller and lighter, with a small software footprint both in code and computation resources, although in most cases, they lack abstraction and networking capabilities and may become complex.

On the other hand, autopilot software runs on top of an operating system, which already has several abstraction layers, networking capabilities, although they tend to be resource greedy.

We will now discuss several open-source autopilot software options.

ArduPilot

Ardupilot is one of the most well established open-source autopilot software. It has over five years of intense development and is capable of controlling multiple types of vehicles, from UAVs to boats and submarines. Ardupilot is shipped with several commercial drones from brands like 3DR or PrecisionHawk and is used by corporations like NASA, Intel or Boeing. Ardupilot has been used in automation of heavy farming equipment, autonomous

UAV aerial surveillance, ground mapping, and Search and Rescue (SAR) missions with successful results [28, 29].

Ardupilot autopilot software features several simple autonomous tasks like takeoff, land, and return-to-home, or more complex tasks like autotune and path-follow. Moreover, Ardupilot supports GCS or API interaction and runs on top of an operating system (currently supports Linux and NuttX, a normal Operating System (OS) and a Real-Time Operating System (RTOS), respectively) [30, 31].

Given the open-source nature of Ardupilot, and its success in several use cases, it represents a good autopilot software to be used for a research platform. Moreover, being an autopilot software that runs on top of an OS, it has the inherent benefits of the OS, but on the other hand, it requires a companion board to run the OS and actively uses the companion board CPU.

PX4

The development of PX4 autopilot software was started by a community of students at ETH in Zürich [17]. PX4 has since then became a collaborative open-source project to the development of a complete UAV end-to-end platform. PX4 was recently integrated to the open-source project Dronecode, which started early this year (2018) and is leaded by the Linux Foundation with the focus on the development of a platform which contains a complete set of tools to work with UAVs in more complex use cases [18, 32].

Similar to the Ardupilot, PX4 features the same simple autonomous task, but in comparison, it contains an autonomous "Follow Me" task that uses an Android mobile device to feed GPS coordinate updates to the drone, and a precision landing functionality. On the other hand, it lacks the autotune of Ardupilot which eases the drone calibration. Also in similarity to Ardupilot, PX4 autopilot software supports GCS or API interaction and run on top of an OS, but in this case it only requires a POSIX-API compliant OS [33, 34, 21].

PX4 is in many aspects similar to the Ardupilot: both are suitable tools for a research platform and their main differences are the community sizes and the features that each of them support. PX4 is more focused on UAVs, and therefore, is more specialized with less community adaptations; in fact PX4 has largely more branches than the Ardupilot and a less active community.

OpenPilot and TauLabs

OpenPilot started in early 2010 with the focus on creating an open-source autopilot firmware for UAVs to the civilian and academic communities. OpenPilot is aimed to support fixed-wing, multi-copter and helicopter UAVs. TauLabs is a group of developers who made part of the OpenPilot firmware, eventually forked from OpenPilot and started their own autopilot firmware entitled TauLabs; their fork was motivated by the need of more freedom in the project decisions [22, 23, 26].

OpenPilot and Taulabs feature similar automation tasks, where TauLabs stood out by the autotune capabilities which were not supported by OpenPilot. Being an autopilot firmware, they did not require companion boards neither used part of their resources for flight operations, although both projects seemed to be gaining momentum and considerable community sizes. Both projects were discontinued in 2015 and 2016, respectively [35, 36].

dRonin

dRonin is an open-source autopilot firmware which focuses on the devolopment of an autopilot software capable of handling all kinds of UAVs and in all kinds of tasks, from racing to autonomous flight. dRonin has support for fixed-wing, multi-copter, helicopter, rovers, and boats [37].

dRonin autopilot software features multiple simple autonomous flight capabilities like land, return-to-home, and hover, but is also capable of more complex tasks like path-follow, point-of-interest or autotune. Also, dRonin has support for GCS or API interaction, a large range of supported FCs, and runs directly on the FC without requiring a companion board and use its resources [38, 39, 40].

The dRonin project is a good open-source autopilot firmware for a research platform with an active community and a wide range of supported FCs. Also, dRonin has been launching a major release of its software approximately every six months, and is fully independent.

2.4 GCSs and Drone Platforms

Across all the available UAV Systems, high level command of UAVs requires the use of a GCS application, or API/SDK. GCSs in general are computer installed applications, which make GCSs dependent on OS compatibility given by the developers. On the other hand, with the development of APIs or SDK tools, UAV communication protocols like UAVTalk or MAVLink started to emerge, allowing for some level of compatibility between distinct UAV systems and GCSs [41, 42, 43].

GCSs are crucial for some drone tasks, like parameter configuration, software updates, sensor calibration, or hardware diagnosis through physical connection to the FC, most GCSs rely on direct link communication to perform high level control of drones, usually through some sort of radio modem or adapter. Also, GCSs lack the capability to control multiple drones simultaneously; the ones that support require a dedicated radio per drone which is undesirable for large scale drone deployments [4].

The limitations of standard GCSs gave room to drone platforms, their focus is to extend the functionality of both UAVs and GCSs implementations. Some are based on GCS refactors which change the underlying connectivity from direct links to Internet Protocol (IP), allowing for multiple drones without dedicated radios. Others take a step forward and develop cloud based solutions with web browser GCSs or API services that relay the user inputs either to the drone or a remote GCS host in range of the drone.

We will now discuss several implementations of GCSs or drone platforms and their features.

Visionair

Visionair is a GCS software application developed by UAVNavigation, a Spanish company focused on development of UAVs, external sensors and other UAV parts like parachutes or cameras [44, 45].

Visionair features simple high level commands like hover, return-to-home, takeoff, and landing, but also more complex like path-planner with up to one hundred waypoints. Visionair supports control to up to 16 drones through dedicated radio links, command relay to a remote GCS through client-server IP connectivity, and area coverage optimization where, given a geographic area, it returns the most efficient coverage path. UAVNavigation also claims it is currently developing a flight formation feature which allows the GCS user to pilot up to three drones in flight formations [44].

Although Visionair supports 16 drones, they must be in autonomous mode, since a GCS only supports one drone at a time to be remotely piloted. This becomes evident when UAVNavigation claims to be developing formation flight functionalities and limits the feature to three drones. Moreover, in order to control 16 drones, in total, 16 dedicated radio modems need to be connected to the GCS application computer or the remote GCSs [44].

Visionair presents a good GCS application which stands out for their multi-drone support, the area coverage optimization, and remote GCS connectivity, but lacks integration capabilities and only supports UAVNavigation FCs.

Airware

Airware is an American company that focuses on the development of an analytics cloud platform to work with drone telemetry and aerial camera data manipulation for industrial areas like mining, construction, and insurance [46].

Airware cloud platform allows their users to gather, process, store and share their aerial data. Although it is not clear which drone brands are supported and how the data gathering is performed, some public videos of the company show several distinct drones like DJI being used [47].

Airware supports automatic 2D and 3D generation of orthophotography or structure modeling and several volumetric measurement tools, and cloud storage for the obtained results. Moreover, Airware cloud platform supports path-planning to both planning and flight stages, whether it supports multiple drones is also unclear [47].

Airware is a good example where drones are completely integrated with other complex tools like orthophotography generation and offer a value added set of mechanisms for automation of drone operations.

PrecisionHawk

PrecisionHawk is an American company that focuses on the development of an end-to-end platform for aerial data. PrecisionHawk started in 2010 with their product "Wini-

Hawk", back then they focused on support to vineyard and executed tasks like aerial image collection for crop data extrapolation or just chasing pest birds away from the crops [48, 49].

PrecisionHawk has since then moved their focus to three products: PrecisionFlight, PrecisionViewer, and PrecisionMapper. Both PrecisionFlight and PrecisionViewer are applications for computers, smartphones, or tablets, and while the PrecisionFlight focuses on task planning and execution; the PrecisionViewer focuses on survey data revision and exportation. The PrecisionMapper is on the other hand a cloud based solution much similar to the Airware, as it allows to process, analyze and share 2D and 3D [48, 50, 51, 52].

While it is not clear if PrecisionHawk applications are compatible with multiple OSs, they are compatible with iOS and macOS devices. In terms of comparison, PrecisionHawk and Airware products seem to be very similar, although the information about Airware is less clear in terms of specifications and capabilities. Therefore, PrecisionHawk gives the consumer a better picture of what and how their products work and what options are available.

PrecisionHawk is a great example of a complete integrated solution for drones, which gives to their users a value added set of tools to take advantage of the capabilities of their drones. Although they claim to support "all drones", they fully support DJI, MicroPilot and their own drones, and offer a limited support to MAVLink capable drones.

DroneDeploy

DroneDeploy is an American company that focuses on the gathering, processing and storage of orthophotography or 3D models. DroneDeploy currently offers two products, one mobile app and a cloud based solution for data aggregation and management [53, 54].

The mobile application is compatible with Android and iOS devices, and it can be used to pilot, manage and monitor drone flights or review and analyze gathered data, either from telemetry or external sensors on board of the drone (e.g.: infrared or camera feeds). The mobile application allows for live orthophotography generation on site, without the aid of external devices or internet connection [55].

The cloud based solution provided from DroneDeploy acts as an aggregation and analytics platform for the gathered data, and also, as a flight planing and management system integrated with the mobile application. This way, it allows other users to improve flight paths for the next flight of the drone, without the intervention of the drone pilot [56].

DroneDeploy offers a good set of tools to obtain aerial imagery and gather topographic data, like PrecisionHawk solutions; this requires local on-site hardware to control the drone and qualified pilots to either overview or control the flight. Although the support is limited to DJI drones, the strengths of DroneDeploy solutions is the live map generation and the management cloud based platform for data analytics and exchange.

FlytBase

FlytBase is an American company, founded in late 2017 and awarded with several American Startup awards. FlytBase focuses on the development of a cloud based platform for drones, to ease the integration and automation processes with drones [57, 58].

The platform developed by FlytBase is composed of two components, FlytOS and FlytCloud. FlytOS is a linux system responsible for time critical operations and is packed with several API tools that abstract the FlytBase platform from the underlying FC. The FlytCloud is a cloud system that processes heavy computation and integration functionalities of the platform [59, 60, 61].

Also, FlytBase has a starter kit available, called "FlytPi" which consists on a Raspberry Pi board with FlytOS already installed and some integration peripherals to connect to the FC and battery of the drone. FlytBase claims that this starter kit is already compatible with PixHawk and DJI FCs [60].

FlytBase is the first commercial available platform capable to integrate and extend current drone capabilities. It is interesting to analyze its implementation, and realize that its architectural bases are similar to the ones developed in this work. Although similar, its platform focuses on providing the tools for clients to develop their own solutions, and do not provide already built-in functions like aerial path optimization, cooperative drone tasks, or scheduled missions.

MultiDrone

MultiDrone is a collaborative project under the Horizon 2020 research and innovation programme of the European Union (EU), which focuses on the development of an innovative intelligent multi-drone platform to cover outdoor events like music festivals and sports.

The goal of MultiDrone is to increase current drone autonomy in decision making to minimize the workload and interventions of the crews, and to increase the robustness and safety mechanisms of drones allowing for tasks with errors or without pilot crew.

Although the final objective of MultiDrone is to provide aerial footage, in order to achieve the project goals, it is planed to build a platform somehow similar to the one proposed in this work [62].

2.4.1 Discussion

There are several interesting approaches, some focus on integrability, others in extensibility, and more. The last two are similar projects, one commercial solution, FlytBase, and one funded by EU, MultiDrone. It is clear that no single solution can make a good general purpose platform for autonomous drone operations that does not include extensive programming skills and high knowledge about UAV systems. In the table 2.1 we present a summary of the discussed options in this section.

Table 2.1: Drone GCSs and platforms overview.

Name	UAVs Types	FCs Suported	Multi-Drone	Notes
Visionair	Multiple	UAVN only	Limited support	Limited to flight planning, monitoring, requires computer
Airware	Multiple	Unclear	Limited support	Limited to flight planning, monitoring, flight command and data gathering + analyzes.
PrecisionHawk	Copter Fixed-wing	DJI MicroPilot Own Package MAVLink Compatible	Unclear	Supports flight planning, monitoring and command Works as an aerial data aggregator
DroneDeploy	Copter	DJI only	Unlikely	Supports flight planing, monitoring and command Works as a aerial data aggregator
FlytBase	Copter Fixed-wing	DJI MAVLink Compatible	Limited support	Offers an integrated platform for clients to develop their own solutions with abstraction of the underlying drone.
Multidrone	Unclear	Unclear	Yes	Mostly aimed for aerial footage, but it is assumed the project will require some platform for abstraction and integration.

2.5 Related Work

The scientific community has been proposing a wide number of services and applications for drones, extended to a multitude of areas like agriculture, gas detection, deliver supplies. In this section we will discuss some of the proposed applications for drones.

MedizDroids Project

In the MedizDroids project [63] it is proposed the usage of drones to ease the process of controlling multiple infectious diseases (e.g., Malaria, Chikungunya, Dengue fever). In most cases, these diseases are propagated through mosquitoes by carrying the disease to non-carriers after biting infected people or animals.

Medizdroids aims to control the population of mosquitoes to slow the propagation rate of these diseases. Currently this kind of control is already done through insecticide spraying to populated areas both indoor and outdoor, and to common incubation areas for mosquitoes. These methods are usually performed by people carrying spraying backpacks or by ground or air vehicles for larger spraying areas. The authors claim that these processes are not affordable and sustainable in the long term, and also that for the case of backpack spraying, it can potentially expose their carriers to dangerous environments or health risks.

To perform these tasks in a way that is affordable and sustainable, the MedizDroids Project aims to use drones to replace the ground and air vehicles in spraying and the people with the backpack spraying. Moreover, they also claim that current available drones cannot meet the engineering requirements to be used in these tasks, and therefore, they plan to create a software architecture that can handle these automation needs in a Service-Oriented Architecture (SOA) manner. According to their publication, they managed to develop a GCS-like application that can interact with the FC through MAVLink, and propose further real-life testing of the system. However, after exhaustive search about the project, we did not find any further documents about this research.

Flight Control System for Drones

In [64] the authors propose a modular concept architecture to simplify the development of FC systems for Drones. Such architecture is proposed to be highly flexible to physical and logical changes, therefore giving the architecture adaptation capabilities to be used on multiple applications. In the proposal the authors specify that the architecture should allow for SOA technologies and that its modules should be loosely coupled to increase modularity. By building an architecture that is compliant with such requirements, it is claimed that building a specialized application over this architecture should be direct and simple.

Much like the work proposed in this thesis, the authors claim that current requirements of drones in terms of integrability require further development, as its relationship between hardware and software is tightly coupled, and therefore, not suitable for fast and low cost application development.

Although similar to ours, this proposal focuses only on the internal workings of the drone, which is still required to the proposed platform in this thesis, but in our case, the Drone System is only one of the parts of the platform which the authors do not approach in their work. Moreover, their work only overviews a theoretical approach to the architecture, and shows no results or actual real-life implementations of the system.

Cooperative Drone Surveillance

In [65] the authors propose a fleet of drones to capture aerial surveillance images from a previously configured area. They take in consideration collision avoidance to both other drones and marked obstacles. In their work, mostly simulated, they propose several algorithms that improve the efficiency of multi-camera image gathering and to optimally spread the drones to improve the surveillance imagery. Also, they have developed a visual tracking system to locate neighbor drones visually, which they reached real-life testing with several successful results.

Search and Rescue Drone Fleets

In [66] the authors propose fleets of drones to help search and rescue teams on their tasks in scenarios of catastrophes or disasters. They propose that multiple types of drones are used in fleets to perform several key tasks in such scenarios, namely: temporary communication structure, up-to-date maps of the affected areas, search for areas with better chances to find alive victims. To achieve this, they propose that drones are equipped with multiple sensors (e.g., infrared cameras, deep penetration radars), and communications equipment not only for the temporary communications support, but also to detect radio transiting equipments on the affected areas.

Medical Transports with Drones

In [67] it is proposed that drones are used to transport critical medical supplies to areas in need, being that a mass casualty scenario or sudden hospital pharmaceutical lack of stocks. In normal circumstances, these transports are done through ground vehicles which are prone to transit delays, or aerial vehicles which are costly to both operate and maintain. With the proposed solution, the authors claim that delivery times can be decreased and costs can be lowered significantly. Moreover, they claim that the same platform can even be used for off-shore or remote incidents to transport supplies and equipment in case of an abnormal crisis situation.

Agriculture Assessment with Drones

In [68] drones are proposed to facilitate the classification of plowing depths of agriculture fields. Currently such processes are performed by means of satellite imagery using optical and multi-spectral techniques, which can give important information about the health of crops to the agriculturists (e.g., feeding of soils, protection from insects and fungi, rate of growth). In this case study, they equipped drones with a Xtion Pro sensor to gather depth data from the crop fields, and concluded that such approaches are feasible with promising results compared to the current solutions based on satellite imagery.

Gas Detection and Mapping with Drones

In [69] a small and energy gas detection system was developed to be used in unmanned vehicles (ground or aerial). For their case study a commercial UAV was used in their field tests. The sensor is capable of detecting gas concentrations in a relative small window of time, which makes the sensor usable withing drones. With such a sensor developed and tested, they propose that drones can be used to detect and map city wide concentrations of gas and create an accurate picture of areas with possible gas leaks or potential gas pollution issues. Finally, the authors propose that small solar panels are used to extend the autonomy of the drone. Their tests show consistent results that effectively support the feasibility of solar panels in small multi-copter UAVs, where flight times can be extended over 3-4 mins; on the other hand, larger UAVs only show less that 1 min gains in flight time.

AirChat Network Monitoring with Drones

The AirChat project [70] proposes drones as a mean to transport a small system that monitors nearby connections of the FireChat application, and is aimed to be used over crowds to gather the metadata transmitted by the application. The authors claim that, with the AirChat project, they are able to track individuals through the metadata gathered by the system onboard the drone, and with that tracking capability, they can view their progression over time and what are the common communication networks they participate.

Persistent Surveillance with Multiple Drones

In [71] it is proposed a surveillance system where multiple drones cover a given area. Within these areas, sub-areas are defined and, for each sub-area, a timestamp is associated with each passage of a drone. The authors propose an algorithm that is capable of coordinating the three drones to fully cover the surveillance area while keeping the age of the last past as minimal as possible. Results show that it is possible to create a system where drones surveillance a given area while keeping the distance between observations low, even while refueling procedures are added to the experiments, by allowing some drones to refuel, others will take their place in the monitoring tasks and so on.

Network Supporting with Drones

In [72] it is described an experimental analysis of the challenges and opportunities of using drones to support wireless networking connectivity. The authors discuss the size and weight limitations imposed to the networking hardware, the link conditions that are inherently inconstant due to the high mobility of the drone, the restrictions about drone autonomy capacities and the requirements to autonomously control the drones. They concluded that there is a high demand for scientific investigation and room to achieve real impact application solutions using drones, and that such solutions should be generic and flexible to be easily adapted to its surroundings.

In addition to the presented works, we will now go over some key technologies that helped the execution of the platform proposed in this thesis.

The fast evolution on the cloud and IoT technologies created and catalyzed several ecosystems of solutions to support distributed system and on-line drones control and sensing solution. Cloud computing can improve the limited computational capabilities of resource constrained mobile nodes and enhance the stability of drones systems [73].

The work proposed in [74] describes a cloud based system for city-wide unmanned air traffic management to keep the city safe using a control system for collision avoidance. However, the authors do not clarify how the proposed cloud-based platform reacts when the flight volumes increase. Therefore, drones cloud-based systems can allow an effectively deployed solution in the aftermath of a disaster for effective disaster response and mitigation [75].

Communication between drones, users and the Dronemap planner cloud solution through the MAVLink protocol is presented in [76], which is supported by commodity drones. The delivered experimental results show that Dronemap Planner is efficient in visualizing the access to drones over the Internet, and provides developers with appropriate APIs to easily program applications for drones. However, due to the asynchronous behavior of Internet, the QoS of drone control over the Internet should be monitored, and the impact of wireless communication delay and quality of drones' management over the network should be investigated as well.

2.6 Conclusion

This chapter started by exposing the history of drones and possible improvements that would make a significant difference in some areas.

Then, the key components of drones were addressed, namely the FC, autopilot software and the tools that provide automation functionalities.

Finishing the chapter, the related work in the literature that comprises similar approaches to ours is presented and discussed with respect to our approach.

Chapter 3

Platform

This chapter starts by exposing the current challenges involved in task automation scenarios for drones, and why they require further development. In consequence of the presented objectives and scenarios, a set of functional and non-functional requirements is defined.

Following the requirements, the organization and choices made at the architecture level are exposed and addressed all the entities that make part of the proposed architecture.

Finally, it is presented a technical overview of how the architecture will integrate drones and how the several entities will be interconnected between them.

3.1 Challenges

In general, most of the current *drones* are controlled by a piece of hardware called a Flight Controller (FC) (also known as auto-pilot). From a radio receiver on board, the *drone* communicates with it through a physical connection, and receives the user inputs (e.g.: up, down, left, right, etc.) and proceeds accordingly. Depending on the capability of the FCs and their firmware, they can support multiple physical communication protocols, functionalities and automation capabilities.

Considering the best FCs currently available (exposed in chapter 2.2), most of them already support higher level commands (e.g.: Global Position System (GPS) go-to, GPS path-planning, auto takeoff/landing, go home, etc.) which need to be issued either by an Application Programming Interface (API) through an active direct connection or a Ground Control Station (GCS) application installed on a computer or smartphone.

GCSs (more detailed in chapter 2.4) present interesting features like live sensor data display and the ability to issue high-level commands; historically, they were a key tool for pilots to remotely fly aerial vehicles. Nevertheless, they have their drawbacks, for instance: most require a single dedicated radio per connected *drone*; very few can connect through Wi-Fi thus restricting its range; each brand/group of FCs has their own implementation therefore lacking uniformity between them.

In the last year (2017), an evolution started to occur: API interaction support from the FCs started to be generally supported, even by the most closed source commercial versions. Not surprisingly, such APIs are implementation specific and severely lack standardization, although some of them followed open-source protocols attempting to reach some level of standards.

With the rise of FC APIs, GCSs can be improved by separating the visualization and interaction from the command issuing and connection management; in other words, GCSs can become visually and interactively similar to humans or machines, but its essence on how the command is issued and transmitted can change according to the requirements of the desired aircraft. Given this, it is a significant step forward for *drones*, but still far from the desired, since APIs have their own current drawbacks: only accessible through direct physical connection (e.g.: serial, UART), and few are well documented or fully functional due to its infancy. Also, in order to interact with the API, given the nature of its connection, normally an extra piece of hardware will be required to communicate with the FC. In most use cases, such hardware is a Single Board Computer (SBC).

With the above rationale, this document will address a methodology to allow generic applications and GCSs to systematically issue control commands to an FC and retrieve live telemetry information about its state.

3.2 Requirements

Given the objectives previously proposed in this document, this section will detail the requirements to have into consideration in the architectural choices of the platform: For organization purposes these requirements are separated into functional and non-functional requirements.

3.2.1 Functional requirements

Most of GCSs currently available require a direct wireless connection to the *drone* through dedicated pairs of transmitters/receivers, which requires to be previously configured by pairing both devices on the ground and *drone*. As a requirement to the current work, defined as **direct control decoupling**, *drones* and GCS should allow for freely inter-exchange if desired, without major reconfigurations in all of them.

Moreover, GCS are vendor specific, supporting only a few different types of FCs, which limits the flexibility to exchange the adjacent technologies. The **direct control abstraction** is a requirement to define a generic set of commands (e.g.: up/down, left/right, forward/backward, go-to GPS location), which will be used to issue commands to the drone; therefore, the abstraction from the underlying FC is possible.

Following the same reasoning, if the command issue is limited to a few types of FCs, the same applies to the sensor data from the drone. **Live telemetry abstraction** is defined to address this issue, allowing to generalize the telemetry feed from the aircraft.

Finally, the technical aspects of communications between drone and GCSs/APIs are strongly dependent on the implementation, not only in terms of required hardware, but also in relation to communication protocols. In this work, referred as **communications abstraction**, it is desired to abstract such implementations from the underlying communication methodologies, allowing for a flexible exchange of adapters according to application needs.

3.2.2 Non-functional Requirements

For the architecture development, a set of non-functional requirements are defined. Starting with **modularity**, by dividing each set of well-defined problems/contexts modularization contributes to the ease of code/application maintenance and integration.

On the other hand, **extensibility** is also a concern to keep in mind through the architecture development. Even with highly modular systems, changes or extensions to modules can represent the full redesign of the entire module which should be avoided.

Controlling *drones*, it is crucial that situation awareness and commands are transmitted/received in due time to compensate or act over possible issues during flight. Therefore, **low latency** must be ensured between the information source and its destination; the maximum acceptable delay is set to 1000ms.

Regarding the drone and its researching purpose, it should be able to perform vertical takeoff/landing, therefore not requiring a large space or launch/retrieval apparatus for takeoff/landing. Moreover, it should allow to carry payloads up to 1 Kg (e.g.: small sensor arrays, cameras, communication interfaces); this is an important step to test integration with other devices, on-board and nearby the vehicle. Completing requirements for the drone, it should be able to hover, easy to perform maintenance and have a flight autonomy of at lease 10 min with the required payload.

Finally, the architecture should support multiple active drones, in the sense of having capability to simultaneously receive and process telemetry data streams and issue control commands, and to support collaborative missions that require multiple drones at once.

3.3 Architecture

In the following sections, a detailed analysis and description of the proposed architecture and its internal components will be explored. This proposal attempts to respect all of the previously exposed functional and non-functional requirements. It is divided into two major component aggregations entitled **Drone System** and **Ground System**.

To integrate and perform interactions between both components, publish/subscribe design patterns are used, therefore enabling high level of decoupling, transport layer abstraction and interaction management. This comes even more useful by allowing drones to individually subscribe to task specific channels; the same can be said to applications, scripts or systems that either use or belong to the architecture.

3.3.1 Drone System

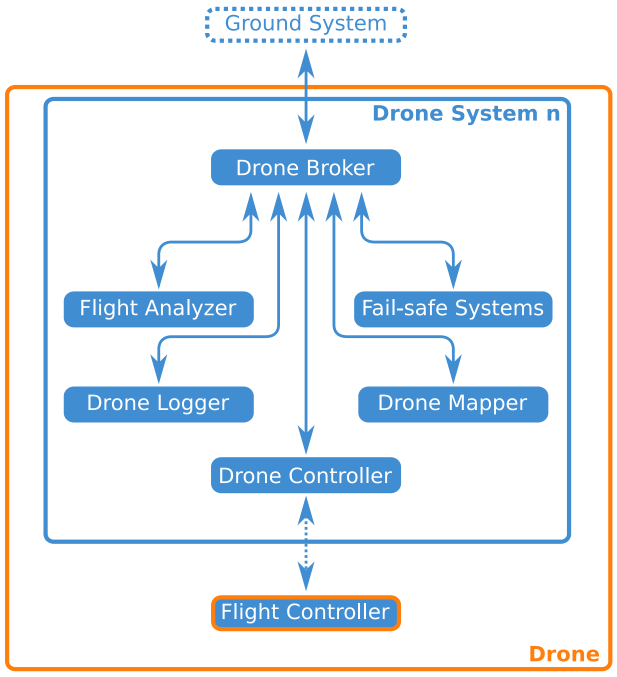


Figure 3.1: Drone System Architecture

The Drone System is composed by the flight components that will play a critical role towards the safe operation of the vehicle.
Drones are a complex aggregation of hardware equipments with their own specifications, technical aspects and firmware, but all of them are controlled through a microprocessor with auxiliary sensors that can either be built in or external to the microprocessor board, known as **autopilots** or more commonly **FCs**. Further reference to these technical aspects

will be mostly mentioned through their common term, FC.
The Drone System architecture is illustrated in Figure 3.1, which contains the components that belong to the Drone System. A description of the internal components and their relevance to the overall architecture are presented in the following paragraphs:

- The **Drone Broker** is the main communication mediator on board of the drone; it works as a *middleware* between the several modules and the underlying network. Through extensive use of this module, the architecture benefits of low coupling between modules and better integrability. It also has the responsibility to directly relay back and forward messages with the Ground System (detailed in chapter 3.3.2).

- The **Flight Analyzer** connects to flight telemetry data to process and analyze behaviors of the drone. This process shall be simple and direct, since processing power can be scarce on board of the drone and it can not be monopolized. Through pattern detection and behavior profiling, this system can detect anomalies to actively notify other modules of particular behaviors. This module favors the architecture to decouple the detection and analysis of certain events from the actions that such event requires, for example, a fail-safe module can be waiting to receive certain events to be deployed instead of actively detect such event.

- The **Fail-Safe System(s)** is a mechanism that, when triggered, is able to minimize the consequences of a failure. These actions can be very simple like preventing takeoff or forcing a safe landing on the detection of low battery levels. The actions can also become more complex like parachute deployment in a catastrophic engine failure.

- The **Drone Logger** connects to the necessary broker channels or topics in order to create a local copy of all the events occurred within the drone for debugging and registry purposes.

- The **Drone Controller** acts as an adapter design pattern, translating broker command messages into FC understandable messages, contributing for abstracting the platform from technical aspects required to properly interact with the FC, therefore complying with the *direct control abstraction* requirement.

- The **FC** flies and corrects the flight behavior of the drone; it is presented in the architecture to host the command end-point and telemetry data source.

- The **Drone Mapper** provides further extension of current geofencing functionalities of drones. It communicates with its other half, Ground Mapper, to support dynamic map loading based on current GPS data and a given radius. Moreover, it is capable to provide richer information like obstacles and minimum/maximum altitude restrictions, common to urban scenarios.

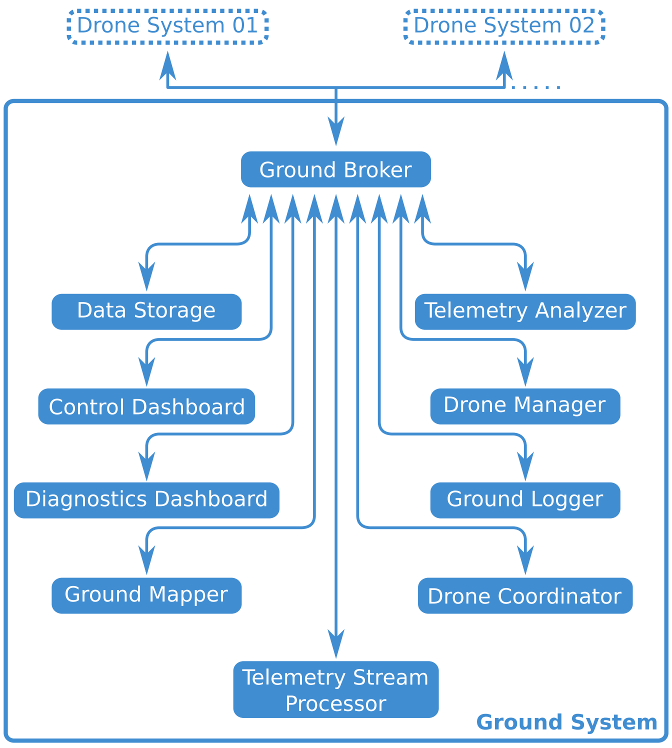


Figure 3.2: Ground System Architecture

3.3.2 Ground System

The Ground System is composed of services, systems and processes that are auxiliary to flight (e.g. data storage, management, front ends, etc.) or can extend flight functionalities (e.g.: support in-flight drone exchange, automated flight plan for area patrolling or aerial mapping/surveillance). To give an answer to the non-functional requirements, components of ground system must be cloud deployable through the usage of a Service-Oriented Architecture (SOA), allowing that each module can grow independently, giving room for multiple instances of the same module for higher throughput, if required.

Once again brokers play a key role through their message routing capabilities. With brokers, development and production instances can co-exist side by side without needing to build a development oriented deployment of the entire platform. The Ground System architecture is illustrated in Figure 3.2. In the following paragraphs, a description of each component will be provided.

- The **Ground Broker**, like the Drone Broker, serves as the main communication mediator to the Ground System. To be able to grow (vertically scalable), it has added responsibility of filtering and routing of each individual drone' messages. Since it can be a platform bottleneck, it must be a cluster (horizontally scalable) capable broker system, which benefits the scalability of the overall architecture.

- The **Drone Manager** acts as a proxy entity, which capability is crucial to the platform architecture because it can easily redirect requests to different endpoints (drones) without major reconfigurations to the edge entities (drones or controlling applications), therefore, answering the functional requirement of *"direct control decoupling"*. The Drone Manager is also capable of tracking the state of each drone (e.g. waiting, disconnected, low battery, flying) and serves a web service to issue high-level control commands to drones.
- The **Data Storage** is a persistent data storage system for later analysis or auditing purposes of the platform.
- The **Ground Logger** (there is a similar system in the Drone System) connects to the required broker channels or topics in order to create a local copy of all the events occurred within the Ground System for debugging and registry purposes.
- The **Diagnostics Dashboard** is a visual front-end that can access performance and sensor data from the entire platform with low latency, allowing for visual trouble diagnosis. It also contains historic information through older datasets.
- The **Control Dashboard** is a GCS like front-end in which users may control their active drones through high-level commands (e.g.: up/down, left/right, etc.). It also allows control for embedded-drone sensors like video cameras, thermal cameras, etc.
- The **Telemetry Stream Processor** subscribes to telemetry data sent by the drones, and it processes the received data to generate new information to feedback the platform (e.g.: compute number of packets received by drone, compute statistical data for drone behavior analysis, etc.).
- The **Telemetry Analyzer** connects to the flight telemetry data to analyze behaviors of the drone and detect anomalies. Through this module, the Ground System can react to behavior changes in order to mitigate any issue (e.g. parachute deployment, emergency landing). Unlike its similar process on board of the drone, due to computational capacity available, it can perform complex analysis of flight parameters potentially gaining extra knowledge and awareness about the drone flight conditions.
- The **Ground Mapper** extends the natural geo-fencing supported by most of the drones. It has the responsibility to track each drone location and actively update it for hazards, safe zones, landing zones, no fly zones, etc. Using dynamic map loading techniques, therefore it saves memory and storage in the drone, and it also removes the need to re-update maps and geo-fencing configurations, keeping all the drones constantly up to date.
- The **Drone Coordinator** has the task of ensuring that drones in a certain area can coexist without interfering with each other. Connected to flight telemetry, it can detect distances between each drone, and upon trespassing a minimum safe distance,

it can notify each of them and even suggest safe actions on how to proceed, like an aerial traffic controller.

3.4 Technical Overview

In this section, technical details of the platform will be explained and detailed, starting with the way the entities within the Drone System are integrated followed by a similar approach to the entities within the Ground System.

3.4.1 Drone Integration

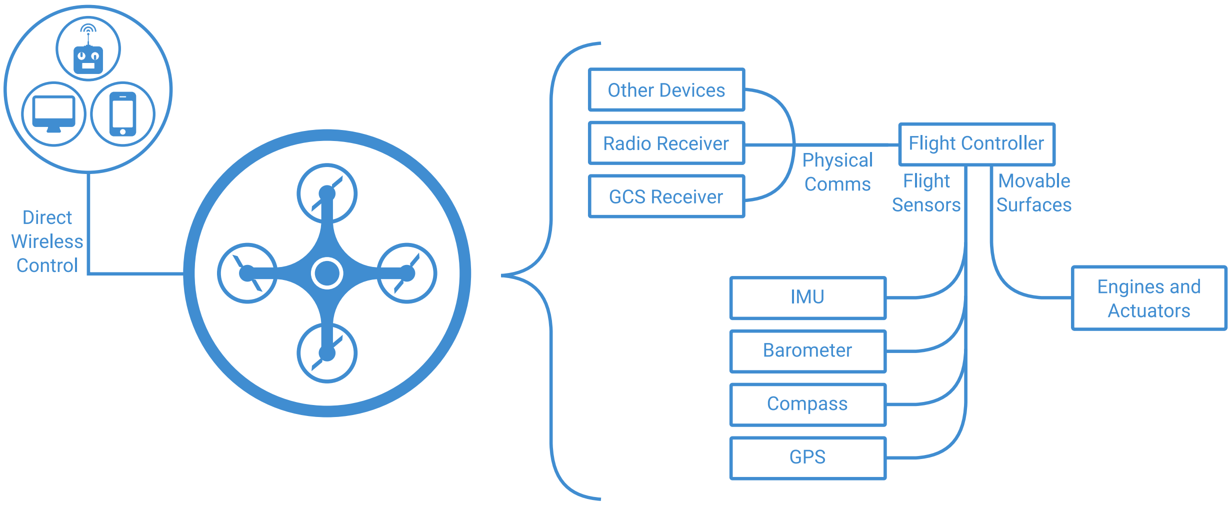


Figure 3.3: Generic drone overview, remote controller and drone with inside view of main components

Drones are a complex aggregation of hardware equipments working together to achieve a controlled flight. They are controlled by a FC which requires input from external communication receivers, and flight sensors in order to act over the movable surfaces, represented in Figure 3.3.

Due to the nature of the FC and its critical operational performance, it does not use a full operating system which could cause serious problems due to Central Processing Unit (CPU) scheduling and possible concurrency issues. On the other hand, for the proposed architecture to work, interface abstraction, multitasking and applications are a 'must have' for abstraction support, meaning that an extra computational node is required on board. For this same reason, external communications and flight sensors take special relevance: they are respectively the communications' link and the source of flight behavior data, this way supporting interactions between the architecture and the FC. In Figure 3.4 the computation node is already integrated with the FC.

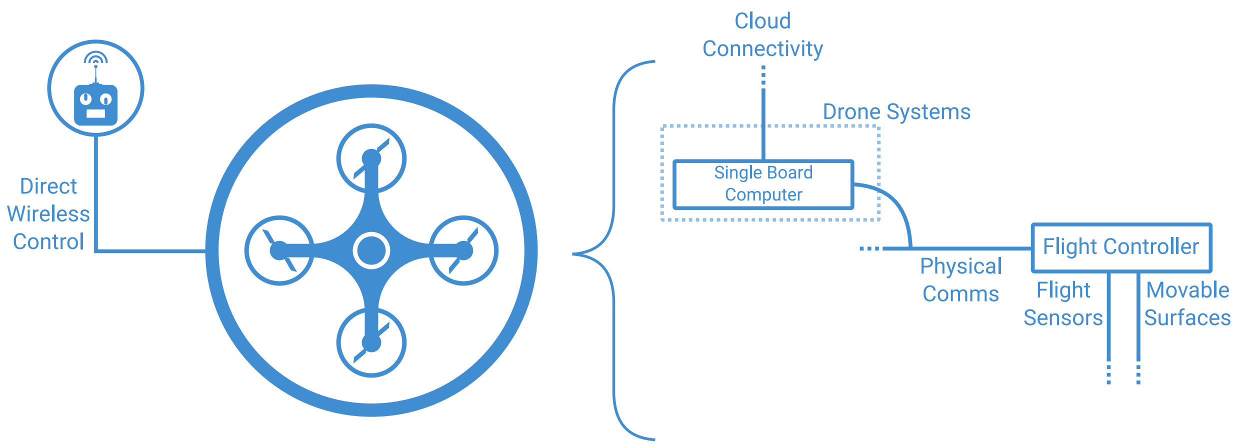


Figure 3.4: Generic drone overview, with architecture integration

Given that each FC has its own implementations, the communication methods or the communication protocols change. Therefore the architecture relies on the drone controller to perform the integration and manage all connectivity to the drone. The drone controller implements direct control abstraction and is one part of the direct control decoupling. In the case of non-functional requirements, it makes the link to the FC modular, adaptable and extensible.

3.4.2 Drone System Integration

This section describes the internal working processes of the Drone System. It will detail how messages flow in and out from each element, and how and why they are relevant to the overall architecture.

One of the main features of message brokers is to support message routing. This allows publishers and consumers to selectively inform the broker about the content/topic of the messages they want to send/receive. In this architecture this feature is extensively used to properly route messages between components. For this purpose, four contexts/topics are considered: control, telemetry, emergency and map. To better exemplify the message routing within the drone, a schematic overview is illustrated in figure 3.5.

The **Drone Logger** subscribes to all types of messages, logging them chronologically in log files in the file system of the computation node. Logging is extremely relevant for debugging and auditing proper working of the application. These logs can be then transferred to the Ground System if necessary for long term archiving due to resource limitations of

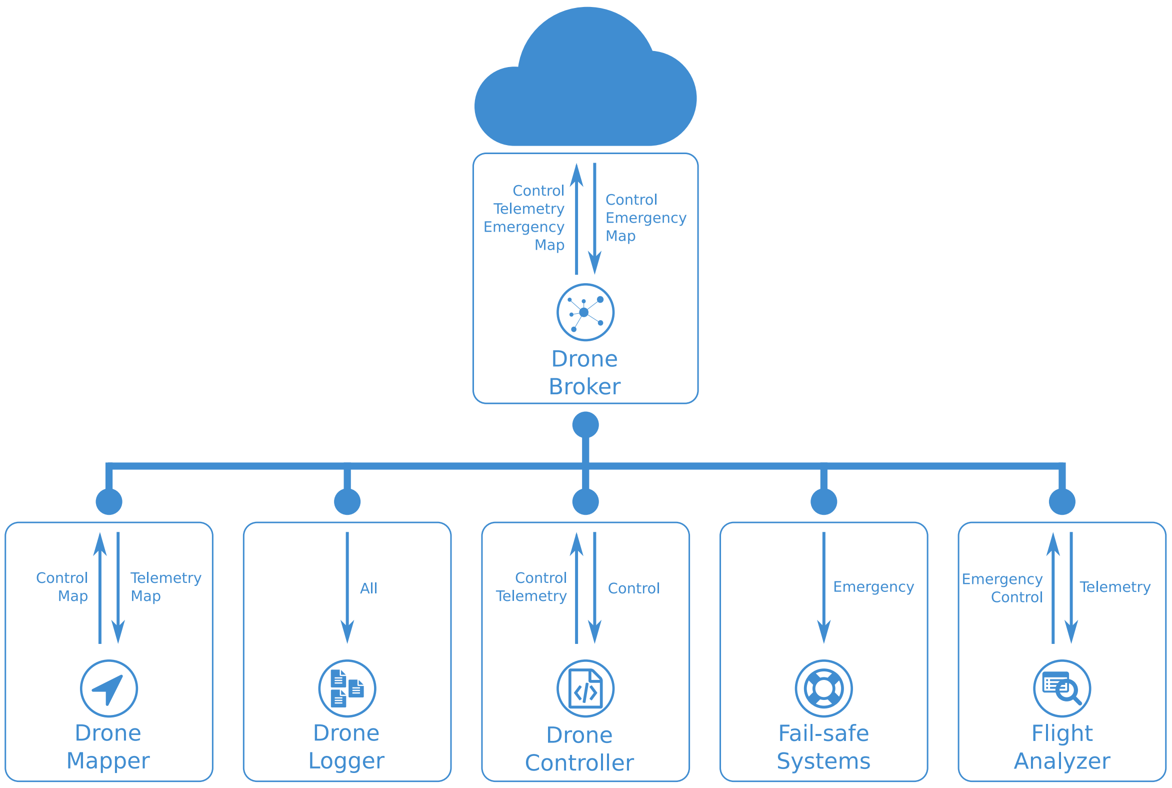


Figure 3.5: Drone Broker message routing schematic

the computation node.

The **Drone Mapper** will subscribe to telemetry messages, using GPS coordinates received to track the current location of the drone. It will also publish and subscribe to map messages. As previously mentioned, it will work in a dynamic map loading basis, therefore needing to interact with the ground mapper to update the map through the integration of configurable parameters and current drone location. It has the responsibility to offer extra information based on GPS coordinates or other geo-location identification techniques to other components in the system: as an example, given a set of GPS coordinates, it returns information about hazards in the area, minimum/maximum flight altitude, safe to land areas.

The **Drone Controller** is the source of all telemetry messages coming from the FC, publishing them to telemetry. The Drone Controller also publishes and subscribes to control messages: these are high level commands like up/down, left/right, go to some GPS location, etc. In order to directly issue control commands to the FC, it must first translate them to understandable commands. Finally, it publishes and subscribes to map messages: the Drone Mapper can be asked for information about a given geo-location; in order for a command to be validated as safe, interaction between these two components may occur.

The **Fail-safe System(s)** will subscribe to emergency messages. The entire system or systems are a last resort tool to minimize damage for both the drone and its surroundings. Their actions will be to deploy a security measurement like a loud buzzer and a parachute or any other preemptive safety measurement required depending on the issue.

The **Flight Analyzer** subscribes to telemetry messages and publishes in emergency and control. Through analysis of sensor data coming from the FC, it looks for signs of detectable abnormal behavior and ensures that sensor readings are within configured thresholds. In case an issue is detected, it attempts to solve or minimize the issue. As an example, a drone in the middle of a flight suddenly stops sending telemetry data, which may be a sign of severe malfunction and may be a scenario of falling: a control message is issued to power off the drone, and an emergency message is sent to deploy parachute and buzzer.

The **Drone Broker Relay** acts as an interconnection agent between the drone and Ground System. It subscribes and publishes to messages on both brokers. These messages are then relayed based on configured parameters, which work as a filter to reduce network bandwidth and processing power usage. Message filtering is performed to only propagate messages with meaning and relevance to the drone operations. Looking from the perspective of the Drone Broker, it subscribes to emergency messages in case some critical issue occurs at Ground System level, publishes telemetry for further analysis and both publish and subscribe to map and control messages, so interactions can be done between components of both systems.

3.4.3 Ground System Integration

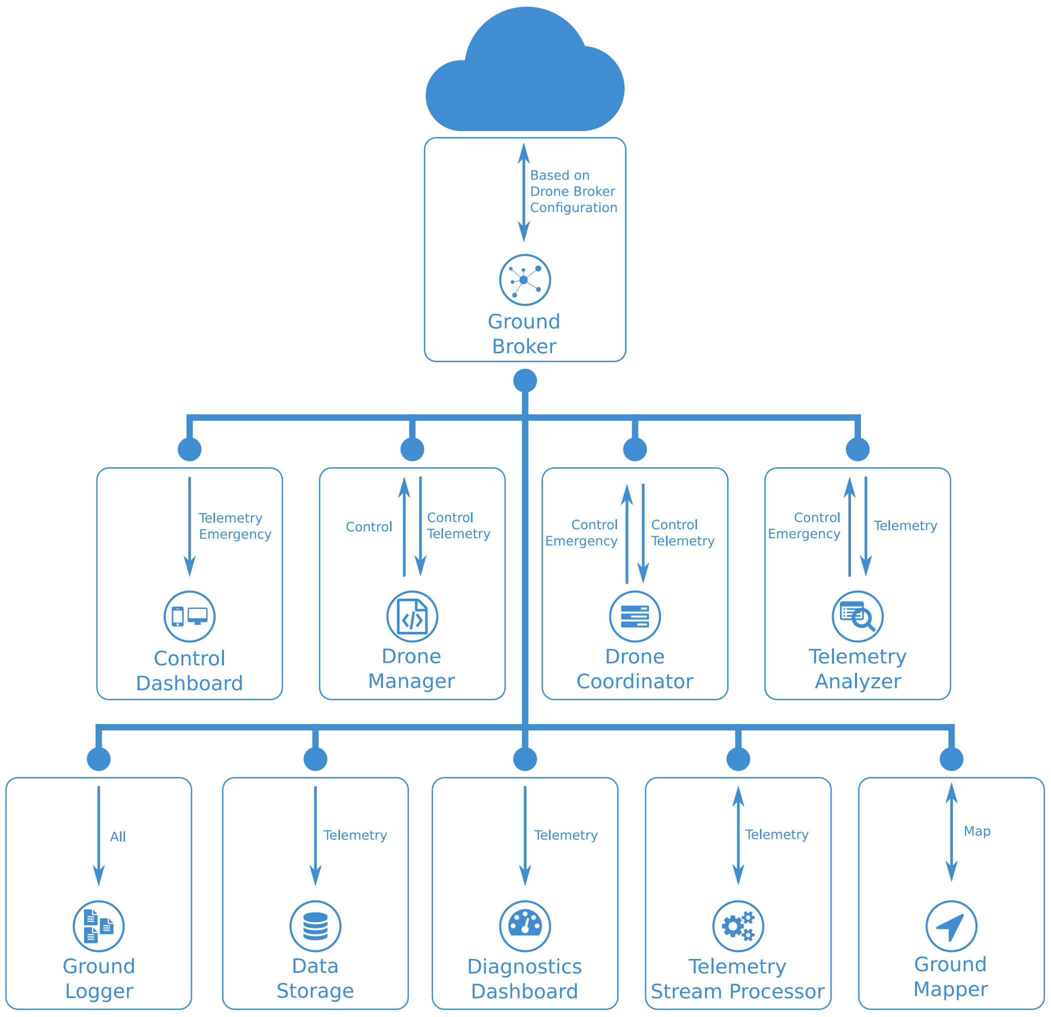


Figure 3.6: Ground Broker message routing schematic

This section details the internal operation of the Ground System. In the case of Ground System components, five contexts/topics are considered: control, telemetry, emergency, map and sysMetrics. Figure 3.6 exemplifies the message routing within the Ground System.

The **Ground Logger**, subscribes to all messages and chronologically stores the message exchange in files; this can be valuable information for debugging and auditing purposes.

The **Data Storage**, in charge of long term storage of telemetry data, subscribes to telemetry messages and stores sensor data information into a time series database in order to be used in flight history reviews or as data sets for further data exploration.

The **Diagnostics Dashboard** is a web based dashboard for flight data overview, therefore making it portable and usable by any device with web browser support. Through the subscription to telemetry messages, it is able to display histograms with near to real time sensor data information. Moreover with the usage of a connection to the data storage database, history review of the data can be displayed.

The **Control Dashboard** is a web based dashboard where users may issue commands to the drone and other connected components (e.g.: video cameras, thermal cameras, sensor arrays). The architecture is able to support these components through extension of the current systems and new message routing contexts/topics. In terms of look, it shall be similar to a generic GCS, since its functionalities are similar to a standard GCS, although with the support to the architecture features.

The **Systems Monitor** subscribes to sysMetrics messages to have an overview of the entire architecture performance and component behaviors. Through the analysis of these metrics, it can detect possible performance degradations or even total failure of the components, which in the case of being a flight critical component, it is able to emit an emergency command to the drones notifying them about the issue.

The **Telemetry Stream Processor** publishes and subscribes to telemetry messages. Through processing sensor data coming from each drone, it generates new telemetry messages with added information like number of sensor readings per second, or instant power consumption, removing that extra processing from other components using the telemetry messages.

The **Telemetry Analyzer** subscribes to telemetry messages. It has the responsibility to analyze behavior patterns and abnormal signs. From the architectural point of view it is similar to the flight analyzer, although in this case, through the availability of more resources, this can perform more complex analysis to the data (e.g.: machine learning, multi-drone data correlations).

The **Ground Mapper** publishes and subscribes to map messages. Both Ground Mapper and Drone Mapper work together to give accurate mapping data to the drone, further extending the normal geo-fencing supported by most FCs.

The **Drone Coordinator** subscribes to telemetry and control messages in scenarios where drones may work together or in close proximity. Through the reception of telemetry data from each involved drone and its issued commands, it can estimate what each drone is doing and at which time. For the case when drones may get too close to each other or fly by each other, it can send control commands directly to each of them overriding the controls of the current controller as briefly as possible.

3.5 Summary

This chapter starts by presenting the challenges involved in task automation scenarios for currently available drones, and why they require further development. In consequence of the presented objectives and scenarios, a set of functional and non-functional requirements was defined.

Following the requirements, the organization and choices made at the architecture level were exposed and all the entities that make part of the proposed architecture were addressed.

Finally, a technical overview of how the architecture will integrate drones and how the several entities will be interconnected between them was exposed.

Chapter 4

Implementation

This chapter starts by exposing the drone used in this work, with a brief exploration of the several drone options and the rationale behind the final decision on the picked drone.
Then, the implementation decisions about the Drone System are exposed with an overview of the rationale behind the implementation choices made for each entity. Finally, a similar description is provided for the Ground System and its internal entities.

4.1 Drone

Based on the requirements presented in Chapter 3.2, the drone shall be able to perform vertical take-offs/landings, hover and carry up to 1Kg payload. Reviewing the possible configurations, shapes and capabilities, a small sized quad-copter is the best suited platform given its mechanical simplicity and stability. By having only four movable surfaces (four rotor engines fitted with fixed pitch propellers), the control is based on rotor speed adjustments. This is similar to an helicopter that can also fly with four movable surfaces (two rotor engines fitted with variable pitch blades), but has its mechanical complexity increased.
Other advantages of multi-copters are the smaller propeller spans in relation to helicopter, a linear growth in terms of carry capacity in relation to the number of rotors (optimal values are approximately 600g per rotor fitted with 22mm fixed pitch propellers), resilience to rotor or propeller failures when using more than four rotors (which is the minimum required to properly control a multi-copter).

Regarding other hardware parts required to assemble the drone, several options were explored. A Do It Yourself (DIY) solution was chosen with the following components: 3S - 5000mA/h Li-Po battery, four 30A Simon Electronic Speed Controllers (ESCs), four A2212/13T 1000KV rotors with 22mm blades and, most importantly, a Flame Wheel F450 ARF Kit from DJI for the frame and wiring exposure, ease to change peripherals.
An OpenPilot Revolution Flight Controller (FC), equipped with an external Global Position System (GPS), magnetometer, and power consumption modules, is used. Also, a

radio receiver is added and connected to the FC, for safety purposes.

As referenced in chapter 3.4, an extra computational node is required to support the **Drone System**; a Raspberry Pi 2 equipped with a 4G Universal Serial Bus (USB) dongle, running Raspbian as Operating System (OS), is used. In relation to other Single Board Computers (SBCs) at our disposal (Pine64, Raspberry Pi 1/2/3), this one has a better balance between power consumption and processing power; however, if for any reason processing power becomes an issue, the OS abstracts deployment from the underlying hardware. These pieces of hardware in conjunction with an Arduino Nano and a 5V 5000mA/h Power Bank represent the full payload in the drone. The Arduino Nano is used for direct control, which is out of the scope of this work, and the Power Bank is used to power both the Raspberry Pi and the Arduino Nano.

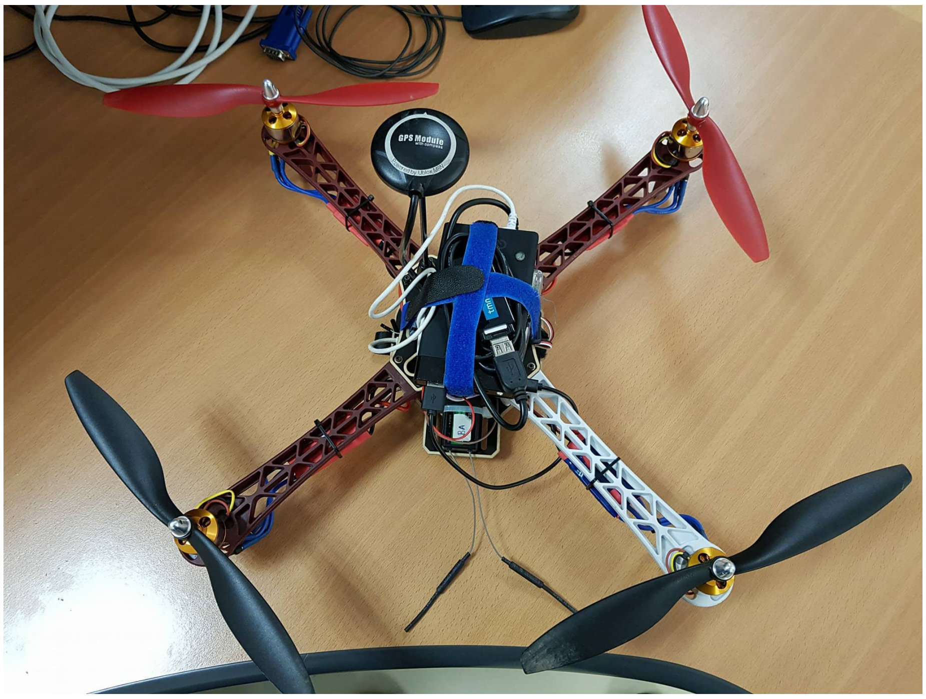


Figure 4.1: Full *drone* setup implementation

Finally, the option to use a secondary extra battery (powering the Raspberry Pi 2 and Arduino Nano) is founded by the fact that, otherwise, the main battery would be draining its own flight autonomy just to power the payload components while not in-flight. However, with a second, smaller battery just for the payload components, its flight autonomy capacity is kept and has the **Drone System** active and waiting for activation.

4.2 Drone Systems

This section contains the implementation choices made of the **Drone System**.

To allow the drone to be abstracted from the platform, an entity called "Drone Controller" was created to act as the adapter to translate the platform commands into commands that can be understood by the drone (Fig 4.2).

In order for the commands to reach the drone, an entity called "Drone Broker" was created to work as an abstraction layer between the platform and the underlying communication protocols through a relay functionality built within the entity. The relay is responsible for the exchange of messages between the Drone System and Ground System and comprises filtering capabilities.

The Flight Analyzer offers a simple system to detect events based on the live telemetry data produced by the drone. This is useful to detect not only abnormal behaviors but also to potentially detect the effectiveness of the commands sent to the drone. Finally, and aggregated to the Flight Telemetry, the Drone Logger will be explained and detailed; its main objective is to work as a Flight Data Recorder (FDR) black box on-board of the drone.

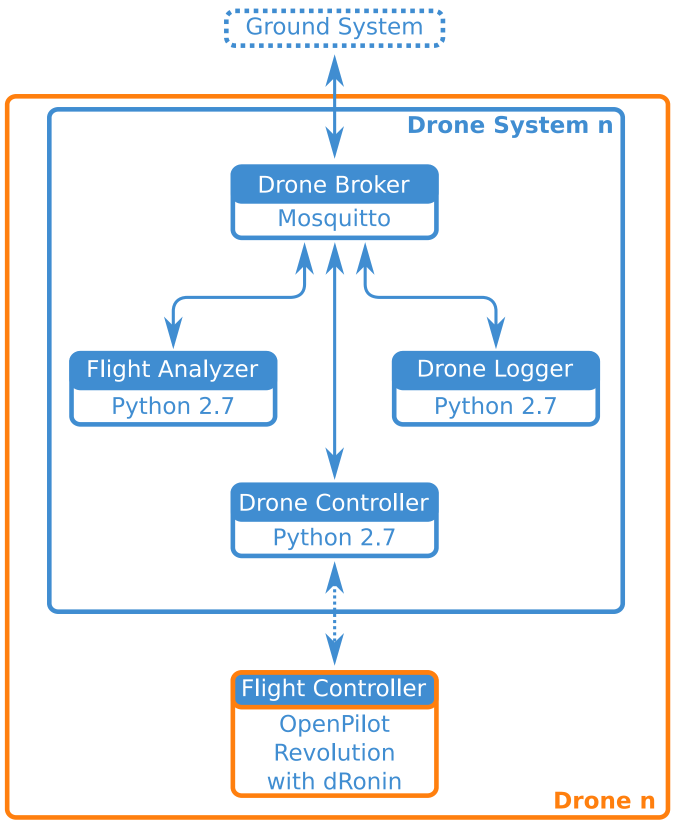


Figure 4.2: Drone System implementation overview

4.2.1 Drone Broker and Relay Mechanism

By architectural design, the Drone Broker needs to be as light as possible in terms of Central Processing Unit (CPU), memory and storage consumption, since the computation resources are very limited within the drone. Moreover, the ratio between flight time autonomy versus total weight has to be a well balanced trade-off. Considering these facts, Mosquitto was chosen to implement this architectural element.

Mosquitto is an open-source project from Eclipse Foundation written in C, where its main focus aims for a small code/network footprint broker. The executable code is around 120kB consuming approximately 3MB Random Access Memory (RAM) with 1000 clients connected[1]. Moreover, Mosquitto makes use of Message Queuing Telemetry Transport (MQTT) protocol to receive and transmit messages, which was also developed to be a lightweight publish/subscribe messaging transport protocol[2]. Finally, the support for MQTT bridges has also been an advantage to chose Mosquitto, with the possibility of being the broker itself to bridge the **Ground System** with the **Drone System**; this will be an added value feature to the platform design.

Through the use of MQTT protocol, Mosquitto connects to the ground broker, further detailed in section 4.3, relaying configured message topics like: Control, Emergency or Map back and forth, while tagging the topic with the respective drone identification string, which is previously configured in the Raspberry Pi. It also filters incoming messages from the ground systems through the use of topic filters based on the same drone identification string, therefore removing unwanted messages.

If topics were tagged only by its base topic (e.g.: Control, Emergency, Map), the bridge would not be able to distinguish outgoing from incoming messages, causing infinite message loops between the brokers. To prevent these loops within the bridge, each topic was subdivided into at least two subtopics: *.in and *.out (e.g.: Control.in, Control.out, Map.in, Map.out). This way each message is clearly distinguishable, therefore avoiding relay loops. Also, although the relay is implemented in the Drone Broker, it does not actively filter the messages; that task is delegated to the Ground Broker through exchange filtering rules, therefore removing the overhead of receiving and filtering unnecessary messages.

Figure 4.3 shows the possibility to have distinct drones simultaneously connected to the same Ground System, and the way multiple drones may receive "broadcast" messages. For example, "unicast" messages are identified with the respective droneID; however, for "broadcast" messages, specific topics may be used. Finally, these filtering and routing mechanisms are automatically implemented on the Ground Broker, based on the configuration file of the Drone Broker.

[1]https://projects.eclipse.org/projects/technology.mosquitto

[2]http://mqtt.org/

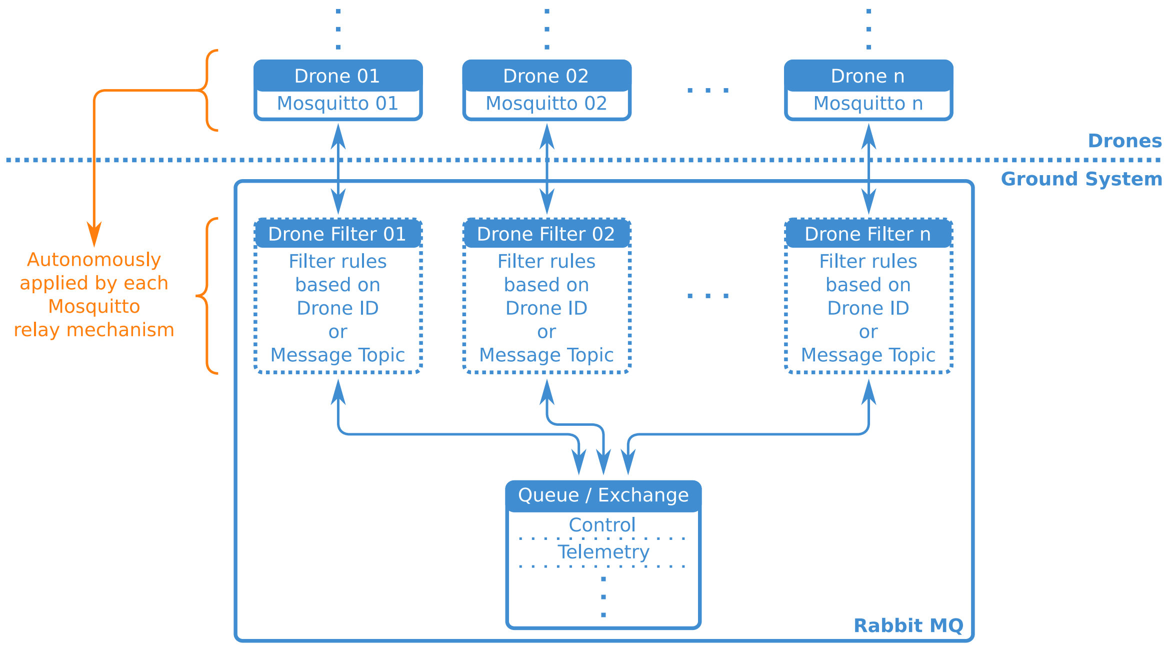

Drone 01
Mosquitto 01
Drone 02
Mosquitto 02
Drone n
Mosquitto n
Drones
Ground System
Autonomously applied by each Mosquitto relay mechanism
Drone Filter 01
Filter rules based on Drone ID or Message Topic
Drone Filter 02
Filter rules based on Drone ID or Message Topic
Drone Filter n
Filter rules based on Drone ID or Message Topic
Queue / Exchange
Control
Telemetry
Rabbit MQ

Figure 4.3: Overview of the filtering and routing mechanisms of the platform

4.2.2 Flight Controller

The FC hardware and firmware have the responsibility of controlling all the flying aspects in the aircraft. All flight parameters are relayed to the FC in order to be executed by the *drone* (e.g.: move left, climb, descend, take-off, go-to GPS coordinates), although more complex flight options can be processed and prepared outside the FC. Ultimately they are translated into FC understandable commands and then transmitted.

The FC hardware chosen to accomplish this task is the OpenPilot Revolution, overviewed in Chapter 2.2. It was packed with OpenPilot firmware from factory; the last stable release of OpenPilot was in May 15th, 2015. Since the development of OpenPilot firmware stalled for almost two years, it was chosen to replace it with dRonin, which is an open-source firmware compatible with the OpenPilot Revolution that has better documentation and an active community. The release version 2017-02-13.1 of dRonin is used in the scope of this document, which was published in March 9th, 2017.

Compared to other firmwares compatible with OpenPilot Revolution, dRonin offered, among other features, an AutoTune functionality for fine tunning the rotor variables, improving stabilization and avoiding shaking. To have the best results in terms of stabilization and shaking, each time the payload is changed in weight, balance or engine/ESC are replaced, the configuration variables should be readjusted.

Currently this process is manually performed by the *drone* operator, by adjusting the parameters by trial and error, which can become extremely time consuming. On the other hand, with dRonin, the *drone* has to be operated for 30 seconds in AutoTune mode and accept the parameters recommended through the Ground Control Station (GCS) application.

OpenPilot Revolution supports interaction with radio receiver and serial emulated through USB. The first is used with manual radio transmitter, which sends user inputs through a unidirectional radio channel, therefore not supporting command feedback. The second is used either by the GCS application for configuration and management, or by an Application Programming Interface (API) for control and telemetry interaction. This API is natively supported by dRonin and uses an open-source communication protocol called UAVTalk, enabling exchange of data with the connected client. Currently supported clients are GCS bundled with dRonin and its Python API.

4.2.3 Drone Controller

Most of current drones require a GCS or API to enable remote interaction and automated flight, which in general is implementation specific, and therefore unable to be reused in other drones which support different GCS or API.

In our implementation, the Drone Controller abstracts and decouples the surrounding components of the platform from implementation specific needs, therefore standardizing interactions with drone.

In chapter 3.4 the architectural details for the Drone Controller were discussed, in which it was defined that communications would be established through means of a publish/subscribe design pattern. This communication methodology is generally used to transmit *String* based messages between publishers/producers and subscribers/consumers. In our case, these messages are formatted in JavaScript Object Notation (JSON), which is a *String* based format, and is also supported by virtually all programming languages.

From the previous chapter, it is known that our current FC uses a serial communication port, emulated through a USB connection and that supports an API that uses UAVTalk as its communication protocol. The Drone Controller acts as an adapter between the platform and the FC formated messages, therefore translating JSON into UAVTalk and vice versa. In Figure 4.4 it is shown this bridging between JSON and UAVTalk formats, which is crucial within the platform: it is through extensive use of this mechanism that the platform achieves abstraction requirements defined in chapter 1.2. The implementation of this module is performed in Python 2.7 and supports the following 11 commands:

- **CONNECT:** provides connection initialization between the Drone Controller and the Drone Manager. Its aim is to formally register and authenticate the drone.
- **OBJREQUEST:** is a debugging purpose command and requests internal FC objects.

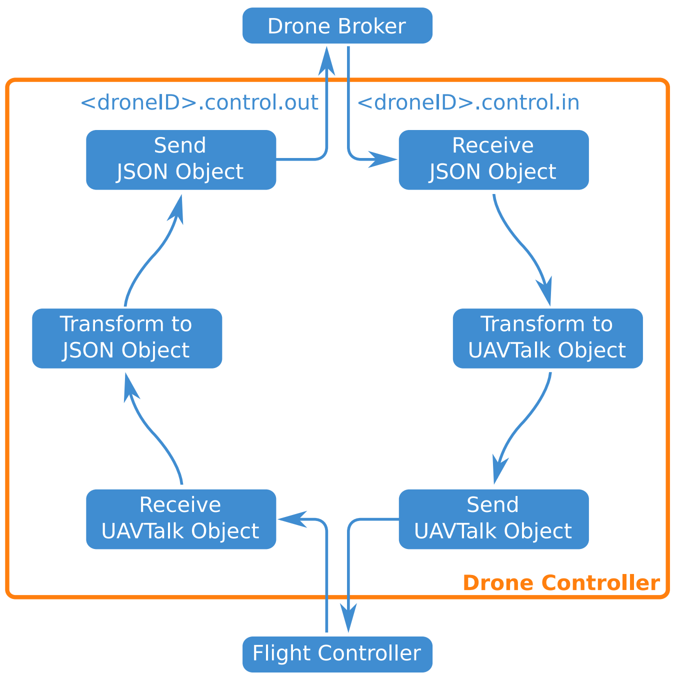


Figure 4.4: Main workflow of the Flight Analyzer implementation.

- **NAVIGATE:** given a set of GPS coordinates, it commands the drone directly to fly towards that location, maintaining its current altitude.
- **DOWN:** given a distance in meters, it executes a vertical movement relative to the drone current location towards/against the ground if the value is respectively positive/negative.
- **NORTH:** given a distance in meters, it executes an horizontal movement relative to the drone current location towards/against cardinal north direction if the value is respectively positive/negative.
- **EAST:** given a distance in meters, it executes an horizontal movement relative to the drone current location towards/against cardinal east direction if the value is respectively positive/negative.
- **ARM:** sets the drone in a flight-ready state, starts all the rotors in an idle rotation and waits for more commands.
- **DISARM:** sets the drone in a safe state, shuts down all the rotors and ignores all commands that require rotor operations.
- **LAND:** initializes the landing procedure of the drone.
- **TAKEOFF:** initializes the take-off procedure of the drone leaving it hovering at a configurable altitude.

- **DISCONNECT:** it is meant to terminate the connection, it unregisters and deauthenticates the drone.

Since the communication medium is a serial emulated over USB, it is bounded to be accessed only by one process at each given time. Due to this technical restriction, Drone Controller also has the task of relaying sensor telemetry data from the FC to the platform. The complete flow of the Drone Controller is represented in a flowchart in Figure 4.5. The application uses two threads, one to translate commands to the FC and one to publish the telemetry feed to the platform. Also note that the application is already prepared to handle other types os commands, not completely related to control the drone but to control other onboard systems if needed. Currently it is also used to control onboard cameras (take photographs or capture video streams).

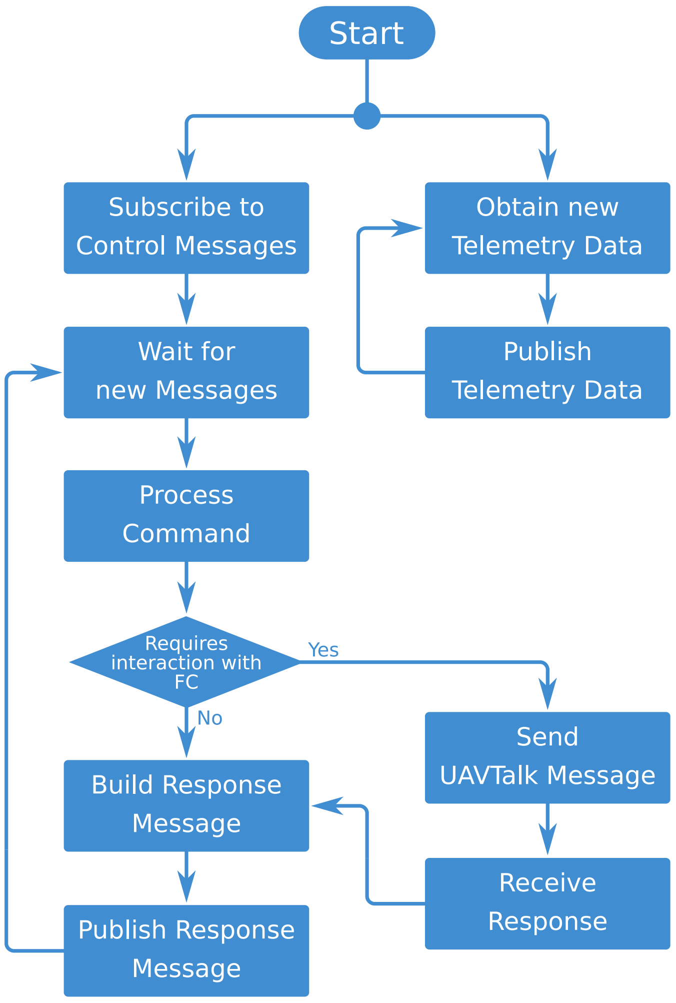
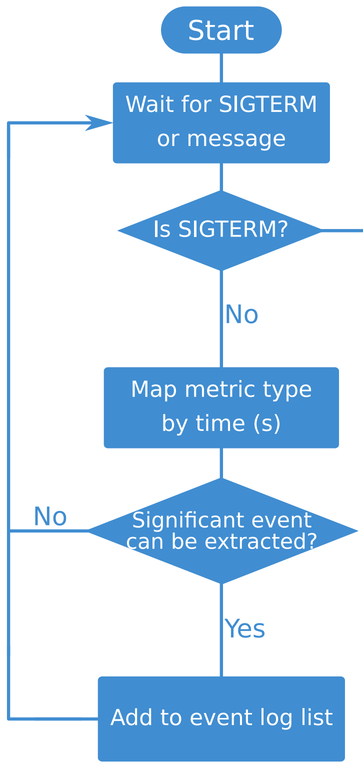


Figure 4.5: Flow chart for the Drone Controller implementation.

4.2.4 Flight Analyzer and Drone Logger

The Flight Analyzer is developed with the objective of detecting simple events based on telemetry data being published into the Drone Broker. It is built in Python 2.7 and is able to effectively detect events such as battery connection, rate of telemetry messages in general and by measurement.

The Flight Analyzer is very important to be used in early stages of the platform development to determine whether or not the telemetry stream is stable and if chronological order of packets is ensured. The workflow of this early version of the Flight Analyzer is depicted in Figure 4.6, and does three tasks. First, it logs all the received messages into a Map that uses the timestamp of the message as its key. Second, it performs several simple preprogrammed analyses to find if an event can be extracted; if so, the Flight Analyzer logs the detected event into an event list. Finally, when a SIGTERM is detected, the module saves all the data to a file for later analysis. At the time of the development of this version of the module, the brokers of the platform were not yet implemented. As a consequence, the Flight Analyzer was directly connected to the FC.


Start
Wait for SIGTERM or message
Is SIGTERM?
Yes
Write data to file
Stop
No
Map metric type by time (s)
No
Significant event can be extracted?
Yes
Add to event log list
Figure 4.6: Flow chart for the Flight Analyzer implementation.

In a second evolution of the module, the flowchart was simplified and parts of the code were then reused to build the Drone Logger module. Initially they were combined because of the Serial connection used to communicate with the FC. As seen in the last section, only one application may be in use of the Serial port at each given time. Given that the serial port was no longer required and the Drone Broker was already fully functional, the Flight Analyzer only needs to subscribe for Telemetry Messages and wait for new messages to arrive: in the case of a detected event, it simply needs to publish the event back to the broker. This flowchart is depicted in Figure 4.7.

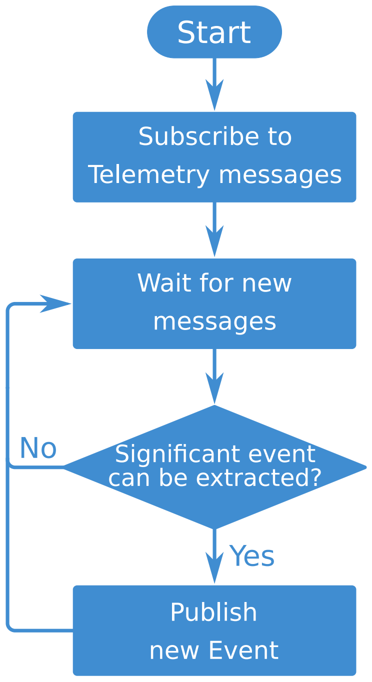

Start
Subscribe to Telemetry messages
Wait for new messages
Significant event can be extracted?
No
Yes
Publish new Event

Figure 4.7: Flow chart for the final Flight Analyzer implementation.

The Drone Logger acts similar to a FDR black box of an airplane. FDRs of airplanes record everything that is going on within the airplane, from the pilot inputs to the performance of the engines. In our case the Drone Logger records all commands that are sent to the drone and all the data that can be collected from the telemetry and other channels of the Drone Broker. The flow chart of the Drone Logger implementation is presented in Figure 4.8. Once again, like an airplane, the black boxes are useful to understand what happened onboard the drone in case of a crash. Although in our case, we have live telemetry data reaching the Ground System, there is the possibility for complete loss of connectivity and therefore, loss of live telemetry. For that specific case, the Drone Logger is a key factor to rebuild the last moments before the crash and possibly understand the reason for the crash of the drone.

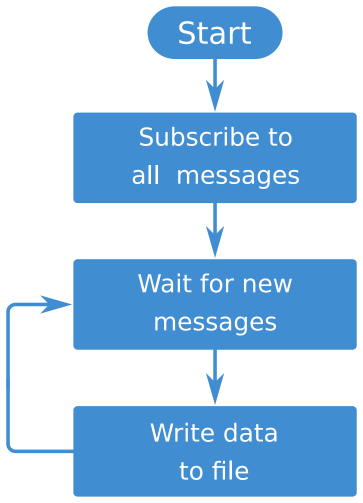


Figure 4.8: Flow chart for the Drone Logger implementation.

4.3 Ground System

This section contains the implementation choices made within the **Ground System**, and a description of the reasoning used to justify the implementation choices.

The Ground Broker is implemented using RabbitMQ; it has multiple static queues and exchanges for routing and simplification purposes.

The Drone Manager is implemented using NodeJS. It works as a development/debug Control Dashboard, and as a Representational State Transfer (REST) endpoint to issue high level commands to each drone.

The Control Dashboard is implemented in two distinct languages: a first crude version of the interface was developed for debugging and simple control, which was integrated with the Drone Controller module using NodeJS; for a more production like interface, a web application is developed in PHP 7.0.

The Data Storage is implemented using InfluxDb. This time series database is used to store historical telemetry data. It also requires a small Python 2.7 application to obtain the data from the Ground Broker and store it in the database.

The Ground Logger is the general log of the platform. It registers all events in the Ground System, and it is implemented using Python 2.7.

The Telemetry Analyzer and Telemetry Stream Processor are developed in a first stage as a single module to take advantage of the implementation based on the Kapacitor application. This has worked well for low levels of telemetry data, but started to fail with higher loads. For this reason, in a second stage the Telemetry Analyzer is implemented using Python 2.7.

The modules developed in this work are represented in Figure 4.9 with a layout of the implementation.

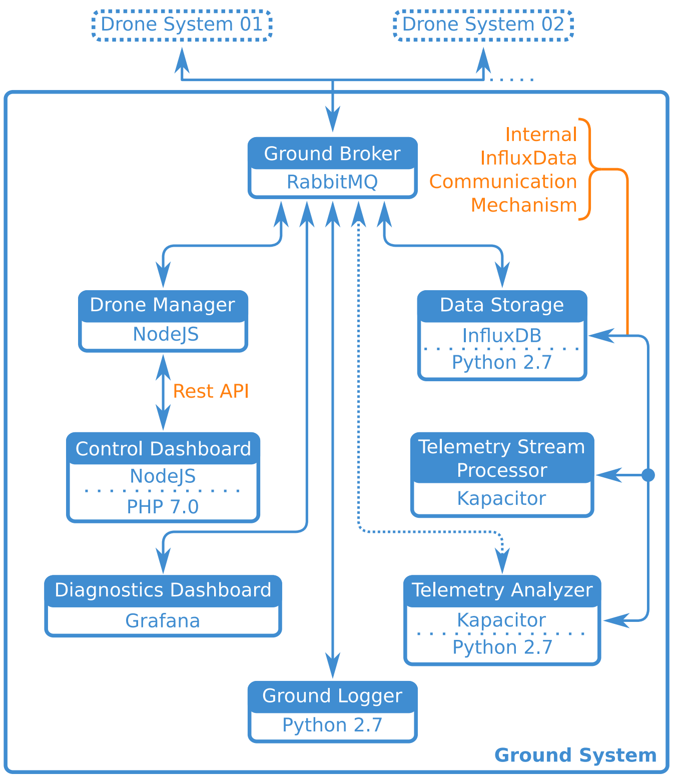


Figure 4.9: Ground System implementation overview

4.3.1 Ground Broker

Acting as an aggregator to all the Drone Systems connected to the platform, the Ground Broker is the endpoint to publish/subscribe messages within the Ground System. Because at this point Mosquitto is developed into our architecture (chapter 4.2.1), in order to take advantage of its bridging functionality, MQTT protocol must be supported by the Ground Broker or an adaptation mechanism would be necessary to translate into other protocols. Since the Ground Broker is the main communication hub for both Ground and Drone Systems, and it is where they merge their messages, it must be reliable, robust and extensible to avoid potential future bottlenecks.

For implementation purposes suiting the architectural requirements, three brokers were studied: ActiveMQ, Kafka and RabbitMQ. ActiveMQ and RabbitMQ are mostly identical: both have the same working principles and features, implemented in Java and Erlang respectively. The choice between the two was purely based on previous knowledge and practice in the usage of RabbitMQ. Kafka looked more promising: it is built in Java, it supports a significant amount of features, modules and extensions; it can potentially implement three of the Ground Systems modules: Ground Broker, Telemetry Stream processor and Telemetry Analyzer. However it was discarded in favor of RabbitMQ which offered a web-based management interface tool which eases the management and monitoring of Queues/Exchanges/Permissions and faster development start due to its better documentation and practical examples to start with. The Advanced Message Queuing Protocol (AMQP) protocol is used to interconnect the internal modules of the Ground System, and MQTT is used as the bridging protocol to the Drone System.

Figure 4.10 is a representation of how each module connects to the Ground Broker and how the messages are internally routed. This figure is a second part of the functionalities implemented within the Ground Broker, which were explained in the Chapter 4.2.1. In the left, four modules are connected to distinct exchanges; these exchanges are previously configured to the RabbitMQ and work as an authentication and permission security feature, because each of the modules has its own permissions to their respective exchanges. Moreover, the exchanges are already configured to route their messages to the appropriated queues.

4.3.2 Drone Manager

According to the architecture design, the **Drone Manager** connects to the Ground Broker to publish and subscribe for control messages. It has the task of tracking connection status of each drone, through the connect/disconnect command, and also to work as an intermediary between the controller client and the Drone Controller itself. In order to support the drone replacement mechanism, it has the task of mapping which controller is controlling each drone and redirect their commands to the correct drone; it acts like a proxy rerouting each request to its proper place based on rules and available resources.

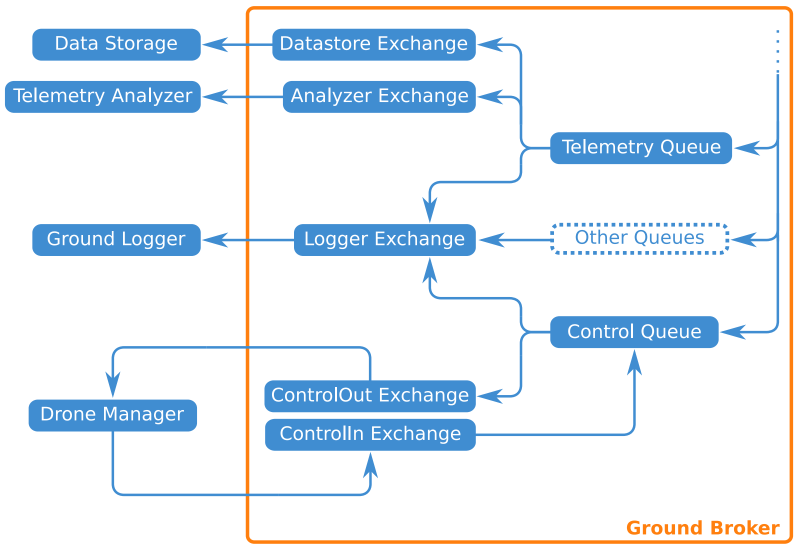


Figure 4.10: Internal Message queuing and routing implementation.

A simple version of the module is developed using Python 2.7, aimed to be initially used for testing purposes through Command Line Interface (CLI) which afterwards would be extended to a web-based API using JSON. Eventually, such extension was made later into development through a small NodeJS application which replaced the previous developed version. This version maps the messages received from the web API and converts them into control messages to the Ground Broker. Such process is represented in Figure 4.11 and the following commands are implemented: NAVIGATE, DOWN, NORTH, EAST, ARM, DISARM. These commands are translated into drone understandable commands by the Drone Controller.

4.3.3 Data Storage

In the development stage, the only data that needs to be long term stored is the flight telemetry data. It is extremely useful to detect visible patterns in the flight behavior and, in the case of a drone crash, be able to diagnose which is probable cause, much like an aircraft FDR black box but with the advantage of not being on-board the aircraft. Moreover, it represents invaluable data to be further used in the future of the platform for possible machine learning purposes, which in our view will eventually be extended to the platform.

Considering the nature of the data being currently stored, it made sense to be implemented in a time-series database. Both Graphite and InfluxDB were considered at first, in which InfluxDb was chosen for its greater support in client languages and schema-free storage. Moreover the InfluxDB applications family is easily integratable with Kapacitor, which could be handful for our Telemetry Stream Processor and Telemetry Analyzer.

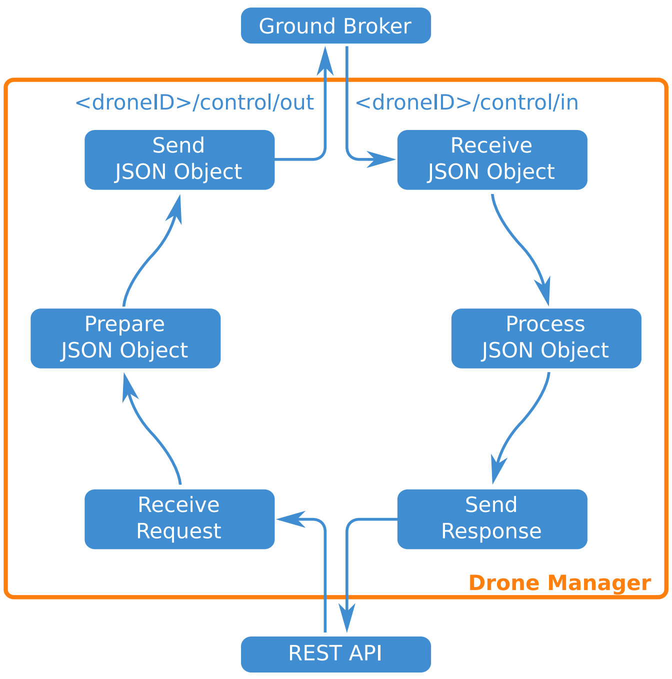


Figure 4.11: Workflow of the Drone Manager for the conversion of messages/requests.

To populate the data to the database, a small standalone Python 2.7 application is created to make the bridge and to translate the telemetry data into the database object format; the flow chart of this application is represented in Figure 4.12. The application starts by connecting to the InfluxDB and then subscribes for telemetry messages at the Ground Broker where it waits for new telemetry messages to arrive. At that point it converts each message to a valid database object format.

The InfluxDB implementation proved to be effective to the initial purpose of the platform, however an potential issue was discovered which may compromise the initial considerations about InfluxDB within the platform. In future evolutions of this work, this question should be reviewed.

4.3.4 Diagnosis Dashboard

With flight telemetry data being received and stored by the platform, a tool to visually display and analyze such data is required. With such tool, most of the troubleshooting of drones can be done remotely or even in flight.

It is imperative that the tool can support low delay between the data reaching the Ground System and being displayed in the dashboard, and the ability to filter data based on dates, drone IDs and type of sensor/metric. Based on the choice made in the last section about the data storage, the selection of this tool requires to be capable of directly interact

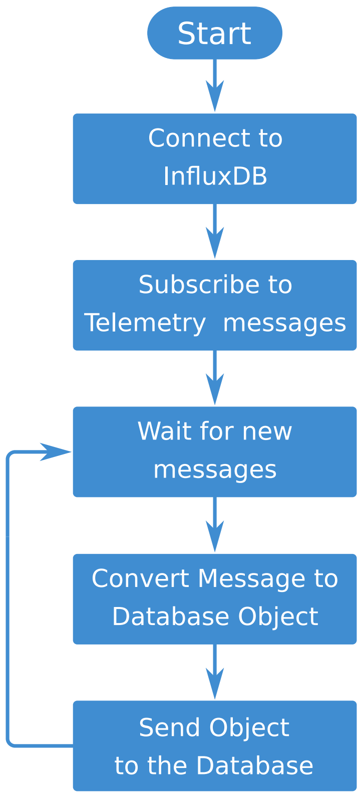
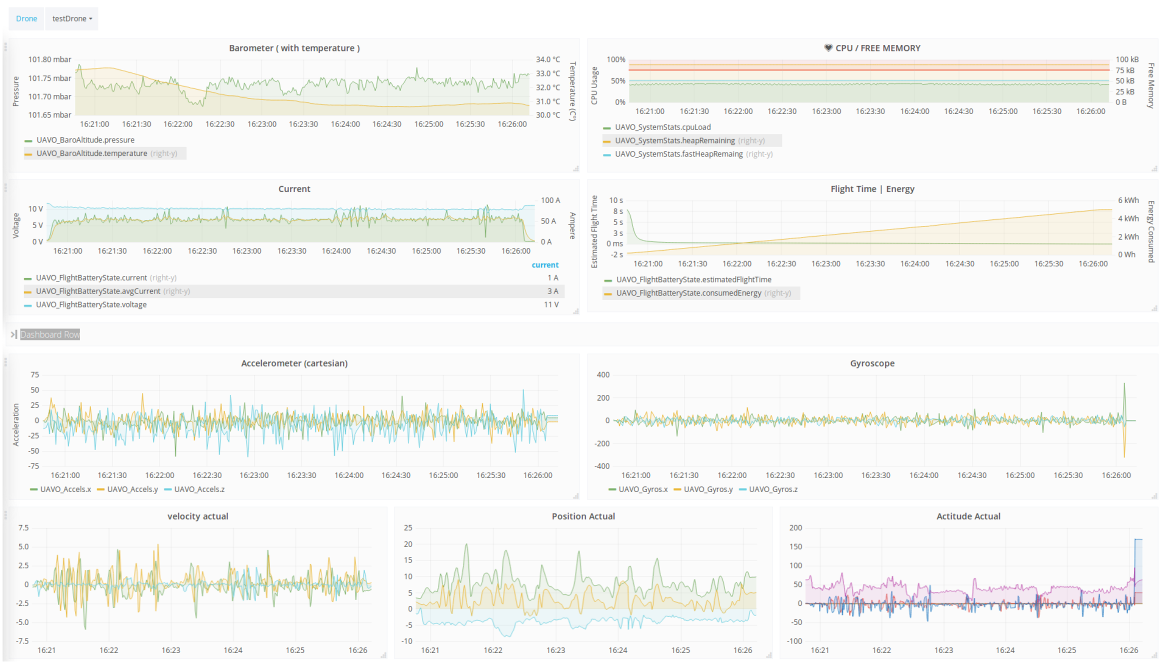

Start
Connect to InfluxDB
Subscribe to Telemetry messages
Wait for new messages
Convert Message to Database Object
Send Object to the Database

Figure 4.12: Flowchart of the Python 2.7 application that interconnects the Ground Broker to the InfluxDB.

with InfluxDB to facilitate the integration of the tools in use.

Grafana is chosen to implement this module, as it supports all the required features and is also compatible with other time series databases. Moreover, Grafana has a built-in alerting system which can be integrated with multiple technologies (e.g.: slack, email) and has customizable templates that allow for a great level of data display customization. Moreover, the compatibility with other time series databases is a potential advantage since it allows for improved adaptability. As mentioned in the last section, there is a strong possibility of changing the current data storage implementation, and the work done within this module may be completely reused.

Figure 4.13 depicts an example of a dashboard representation. Through the dashboards, we can easily plot and combine multiple telemetry data and perform simple aggregation/transformations with the live telemetry data being received. This feature is particularly useful for debugging the drone or, in case of a crash, to understand what might have happened before the crash. These graphs are capable to show near to real time telemetry data being transmitted from the internal flight sensors, making the requirement for direct drone connectivity only in very specific cases.

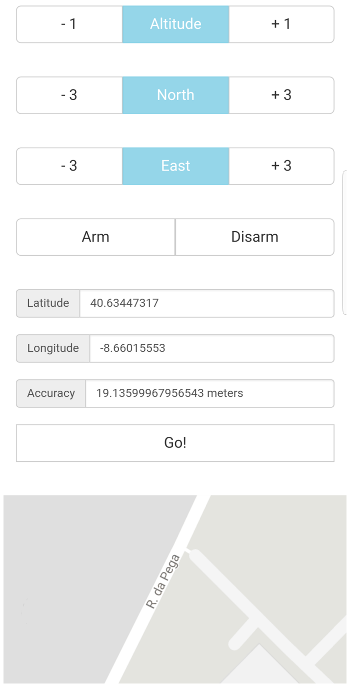


Drone
testDrone
Barometer (with temperature)
Pressure
Temperature (C)
UAVO_BaroAltitude.pressure
UAVO_BaroAltitude.temperature (right-y)
CPU / FREE MEMORY
CPU Usage
Free Memory
UAVO_SystemStats.cpuLoad
UAVO_SystemStats.heapRemaining (right-y)
UAVO_SystemStats.fastHeapRemaing (right-y)
Current
Voltage
Ampere
UAVO_FlightBatteryState.current (right-y)
UAVO_FlightBatteryState.avgCurrent (right-y)
UAVO_FlightBatteryState.voltage
Flight Time | Energy
Estimated Flight Time
Energy Consumed
UAVO_FlightBatteryState.estimatedFlightTime
UAVO_FlightBatteryState.consumedEnergy (right-y)
Dashboard Row
Accelerometer (cartesian)
Acceleration
UAVO_Accels.x
UAVO_Accels.y
UAVO_Accels.z
Gyroscope
UAVO_Gyros.x
UAVO_Gyros.y
UAVO_Gyros.z
velocity actual
Position Actual
Actitude Actual

Figure 4.13: Web based drone telemetry dashboard example, where multiple sensory data (battery levels, power consumption, barometer, accelerometer) are plotted.

4.3.5 Control Dashboard

Developed in NodeJS, we implemented a small web interface that has the capability to send basic navigation commands to the platform. This web interface is implemented in early stages of development of the platform. Its main purpose was to simplify the interface to test the response of the drone to the commands. As a consequence of its simplicity and effectiveness, it is still used mainly for debugging or testing purposes. This web interface is represented in Figure 4.14, and it supports the following commands: GPS coordinate, up/down, east/west, north/south; the web interface has also a small map that can be clicked to select a new GPS point to send to the drone, and a button to send the current location of the device to the drone.



Latitude 40.63447317
Longitude -8.66015553
Accuracy 19.13599967956543 meters

Figure 4.14: Web based *drone* control dashboard example, which is more mobile friendly, where commands can be issued to a given drone.

A more complex GCS like interface is also developed. The web-based interface is capable of displaying drone telemetry data in a near to real time basis, and directly control a previously selected drone. In order to support this web interface, it connects to the telemetry message channel filtering the data to the respective active drone, and through the previously mentioned JSON API to issue control commands to the selected drone. The interface is illustrated in Figure 4.15.

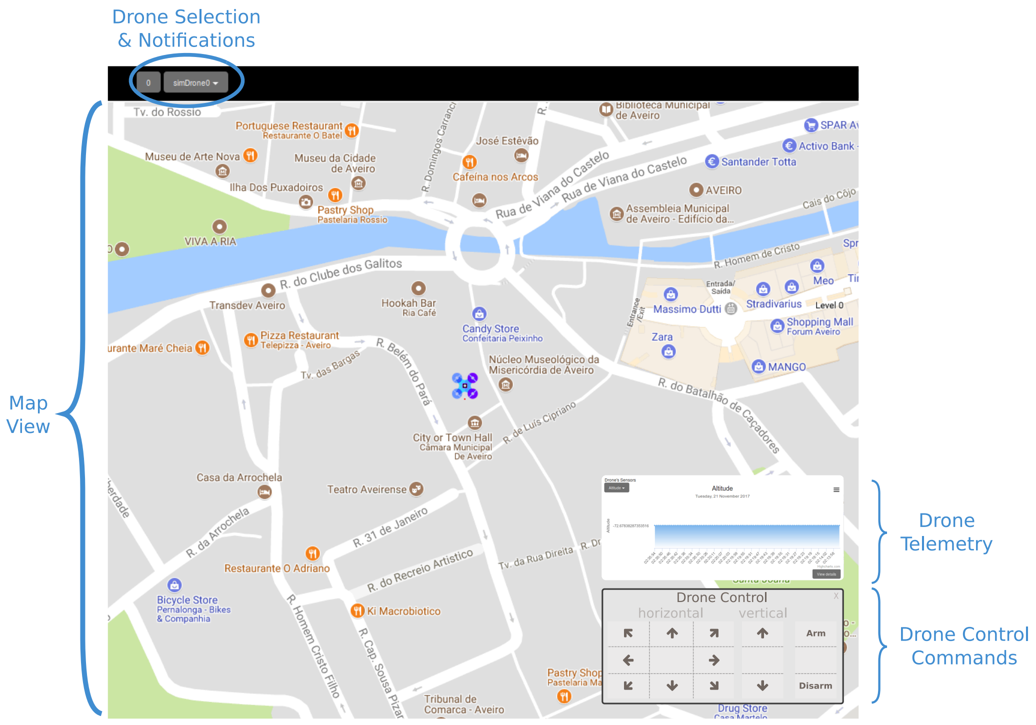


Figure 4.15: Web based GCS like control dashboard example, where commands can be issued to a given drone and some of its telemetry data.

4.3.6 Telemetry Modules

The first implementation of the **telemetry stream processor** and **telemetry analyzer** is merged into the same base implementation while keeping its modularity, by using Kapacitor from the InfluxDB software family, through its *TICKscript* language.

This approach was promising at first: it is supposed to work in an internal publish/subscribe design pattern between Kapacitor and InfluxDB. As data is issued to storage within the database, a queue is populated with the same data being later received by Kapacitor. Preliminary tests with the implementation worked well and several features have been implemented, like power calculation, number of drones connected, and number of telemetry packets by drone.

When the case studies have been tested, it was verified a serious flaw within the InfluxDB and Kapacitor relationship: in the case of receiving about 2000 measurements per second depending on the setup configurations, InfluxDB starts to bottleneck the data being transmitted to the Kapacitor resulting in faulty measurements.

TICKscript is indeed a good language to be used for this kind of job: it is intuitive, easy to learn and read even when not fully understanding the language; unfortunately, by using the free versions of the supporting software (InfluxDB, Kapacitor), it ceases to work with higher throughputs of data (with more than 6-8 drones, the performance is seriously compromised). Moreover, only the payed version of the software, which has to be hosted on their cloud, has the clustering capability.

To correct this issue to some extent, both modules were converted later into Python 2.7.

4.4 Summary

This chapter started by exposing the drone used in this work, with a brief exploration of the several drone options and the rationale behind the final decision on the chosen drone.

Then, the implementation decisions about the Drone System were exposed with an overview of the rationale behind the implementation choices made for each entity. Finally, a similar approach was done for the Ground System and its internal entities.

Chapter 5

Platform Evaluation

This chapter presents and discusses the tests and results obtained from this work. It starts with an overview about how the Ground System modules were deployed and what hardware specifications have supported the same deployment. Then, it follows with the conclusions obtained from the initial drone telemetry behavior analysis. It is then followed by the stream processing mechanisms for drone telemetry implemented in the platform, and an analysis about the performance of the platform when hosting multiple drones in simultaneous operation. Finally, it presents an analysis about telemetry gathered from drones operated by the platform, and multiple cases of telemetry data that can be automatically combined by the platform to diagnose potential malfunctions or data extrapolation.

5.1 Deployment

This section will detail the deployment choices made to develop and isolate each of the modules contained within the Ground System. It will also document the specifications of the hardware used to host the deployment of the platform.

Ground System Modules

Through all development phases, non-functional requirements (Chapter 3.2) enforced each module to be self contained and as loosely coupled as possible. Also, by architectural choice, each module interconnects to each other through the use of a publish/subscribe design pattern, which further enhances the modularity by decoupling each module from the underlying network and communication implementations.

From a deployment standpoint, this means that each module can be deployed into single or multiple machines, the latter being a more complex scenario prone to eventual challenges. On the other hand, it can add more redundancy and resilience to failures.

Overlooking into the architecture, the most crucial and prone to failure module is the **Ground Broker**, which represents the main connectivity endpoint for the entire platform. Considering that a drone totally loses connection, such scenario has near to no effect into the overall performance, since the event can be easily detected and a replacement drone can be dispatched to continue the work. Such statement cannot be applied into the Ground Broker, since latency or simply jitter on message exchanging mechanisms can have serious negative effects to the overall performance.

The same is also true in other key elements within the Ground System, like the Drone Manager, Telemetry Analyzer or Telemetry Stream Processor. For this reason, it is decided that the platform deployment shall be performed as close as possible to a distributed scenario. This way each node has responsibility over a few modules of the Ground System, therefore ensuring the inner communications and decoupling work efficiently, as expected, and leaving room to add clustering or multi-node to modules that would require more computational power.

Ground System Hardware

The hardware that supports the Ground System can indirectly impact the platform and become the performance limiting factor; the hardware has its inherent limits which can only be overcome by horizontal or vertical scaling. Given the Ground System modularization analyzed in the last chapter, the software is able to support both hardware scaling options; therefore, deployment can be done either by single or multiple physical computing nodes.

As it is imposed by current available hardware, the Ground System deployment is performed in a Dell PowerEdge R710 equipped with dual Intel Xeon, dual Gigabit Ethernet, 120 GiB memory, 6 Tb Storage with RAID 6 and running VMWare ESXi 6.0. Later in the development, an 120Gb SSD is added to the system for caching or Virtual Machine (VM) dedicated storage.

To host the platform, a VM is configured with 8 core Central Processing Units (CPUs), 8 GiB memory, 30 + 100 Gb virtual storage (for Operating System (OS) and Containers respectively). Because other parallel VMs are working in the server, this VM is more time and delay sensitive, therefore higher hardware access priority was set to this VM.

The VM setup includes Ubuntu 16.04 LTS with all available updates at the time. Linux Containers (LXC) are installed to isolate and simulate the separation between modules; a container system is chosen instead of a virtual machine to avoid the overhead caused by virtualization and ease to create and deploy containers “on the go” compared to virtual machines.

Docker is also a possible approach to use in this scenario. Docker has some advantages like a greater number of container images “ready to go” and large community; however, it lacks the fully working networking stack and is less flexible to add/remove applications as needed without rebuilding the container all together, which is desirable for the platform development phase.

Considering that LXC is chosen, both technologies are not mutually exclusive, therefore a hybrid solution can exist. Also in a “production-like” scenario, both technologies would be in equal grounds, although Docker would be more appealing for the amount of images “ready-to-go” available in the community.

5.2 Drone Telemetry Behavior

Telemetry sensor data from a drone can vary both in the amount and variety of measurements from internal sensors or metrics. By default dRonin has a set of configurations about which sensors/metrics and how many measurements are to be sent as telemetry data. These measurements are important as they give the position and state of the drone at each given moment.

To properly understand this behavior, an analysis is conducted to determine how much telemetry data is transmitted by each drone, therefore estimating how much effort from the platform is required to receive the measurements. It is also important to determine the metrics structure, how frequent they are and if they are related with other factors, like moving switches of the remote controller. This analysis is performed using the dRonin version “dRonin2017-07-17” which is entitled as“Neat”, and it is observed that the telemetry data included 17 distinct UAVTalk objects with an average frequency of ∼22.8 objects per second as shown in Figure 5.1.

The graphs in Figure 5.2 show the results of several days of combined telemetry data: they analyze the periods between object observations. This analysis shows that most of the objects have regular periods between measurements, like: UAVO_ActitudeActual, UAVO_Waypoint, UAVO_FlightBatteryState, UAVO_PositionActual; very few of them show irregular periods, like UAVO_SystemStats or UAVO_WatchdogStatus, as shown in Figure 5.3.

Also from this analysis, it is detected that if the Flight Controller (FC) is only powered up through Universal Serial Bus (USB) input, the amount of measurements is lower than when powered by the battery pack of the drone. Such behavior is related to the Global Position System (GPS) and magnetometer external sensors which require power input from the battery pack of the drone, and therefore do not transmit their measurements to the FC.

The tests are also extended for longer periods of time (up to 7 days). In this case, it is not considered the battery pack of the drone, which represented a slightly less amount of telemetry data due to the lack of external sensors. The results show that active connections in extended periods of time (over 24 hours) can result in random events of complete lack of measurements. As shown in Figure 5.4, these events normally last for periods of one

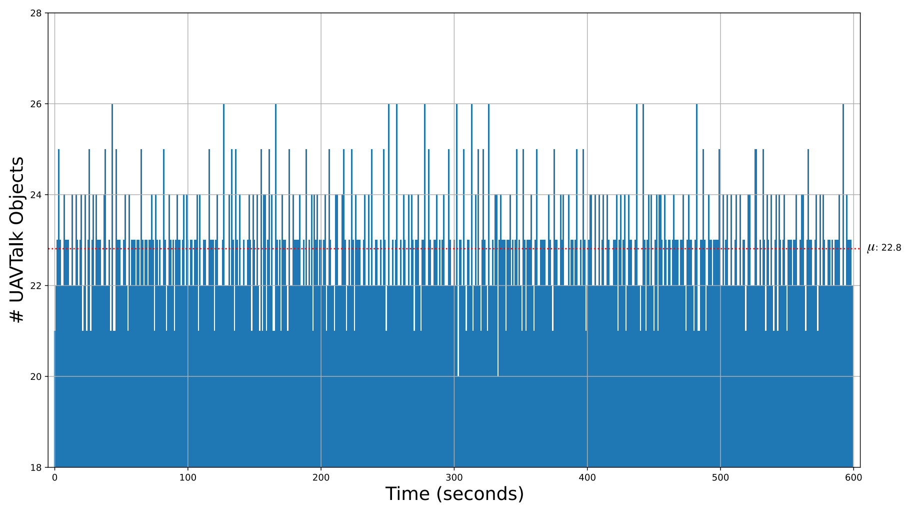
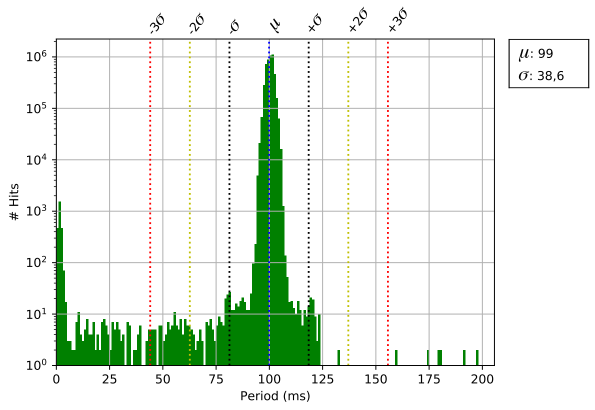
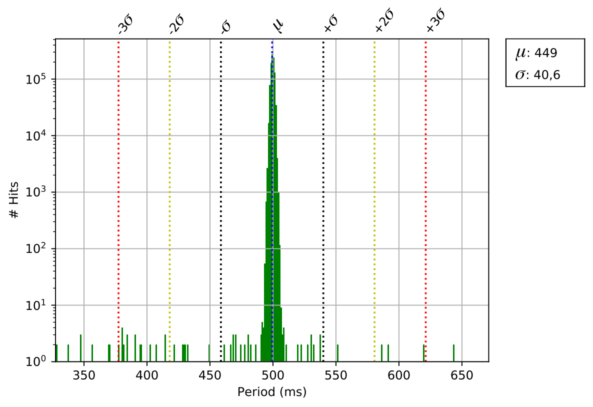


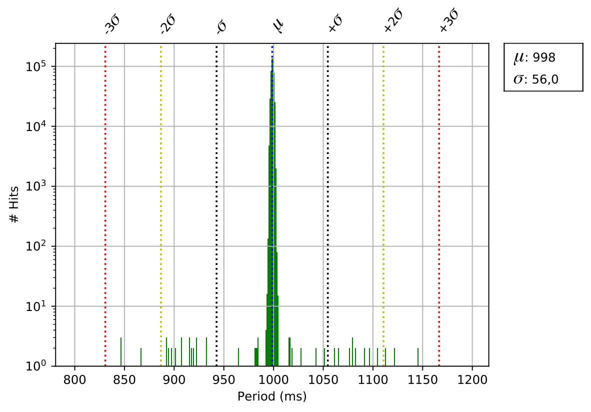
Figure 5.1: Telemetry object count by second



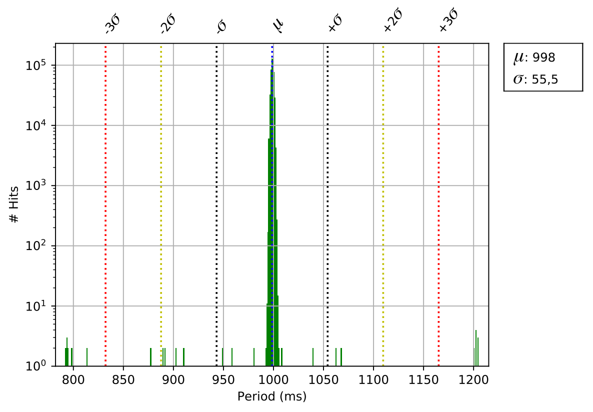
(a) UAVO_AttitudeActual object



(b) UAVO_Waypoint object

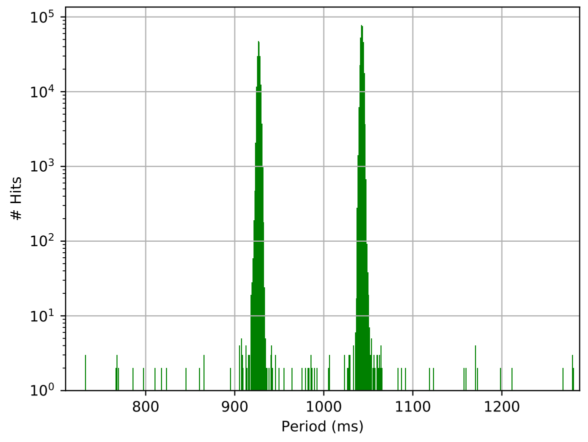


(c) UAVO_FlightBatteryState object

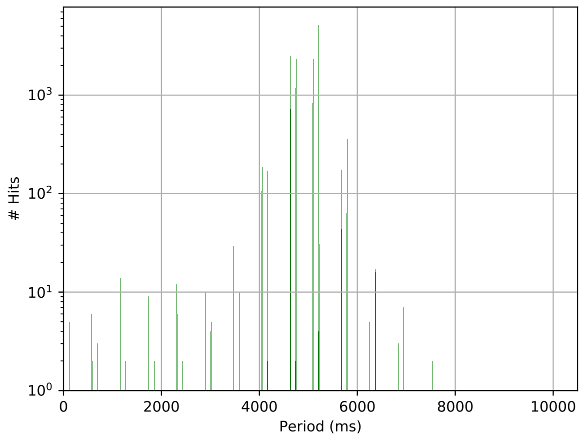


(d) UAVO_PositionActual object

Figure 5.2: Periods between each UAVTalk object that show regular behaviour.



(a) UAVO_SystemStats object



(b) UAVO_WatchdogStatus object

Figure 5.3: Periods between each UAVTalk object that show irregular behaviour.

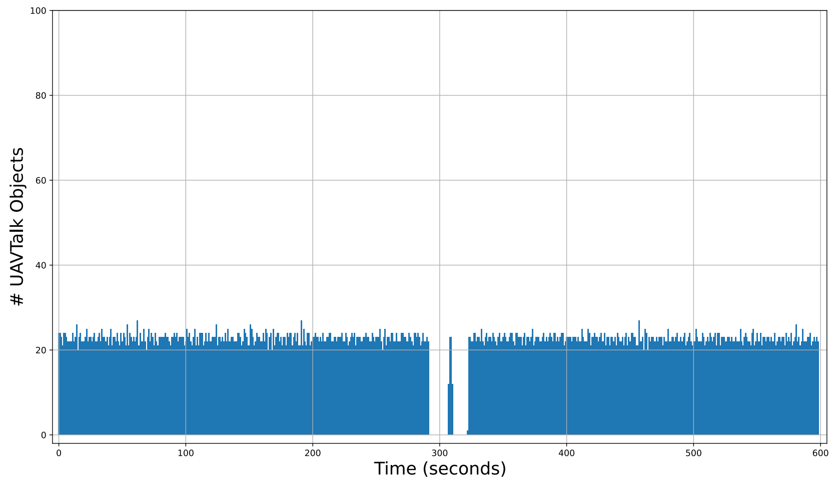
---

to five seconds and there are some extreme cases of up to sixteen seconds. Moreover, it is detected that the events where telemetry is absent have three distinct patterns: first, there is the fact of complete lack of the measurements as shown in Figure 5.4a where there are no measurements around the 300s mark; second, there are events where measurements seem to be buffered within the FC and then transmitted later on a bigger group as shown in Figures 5.4c and 5.4d; finally, such events can eventually lead to a complete loss of connectivity between the Single Board Computer (SBC) and FC as shown in Figure 5.4b. Several attempts to track the source of this behavior proved to be inconclusive; our best guess is a firmware glitch at the dRonin serial emulation which has known issues to become unresponsive after an abrupt disconnection and requiring a full FC reboot to regain connectivity.

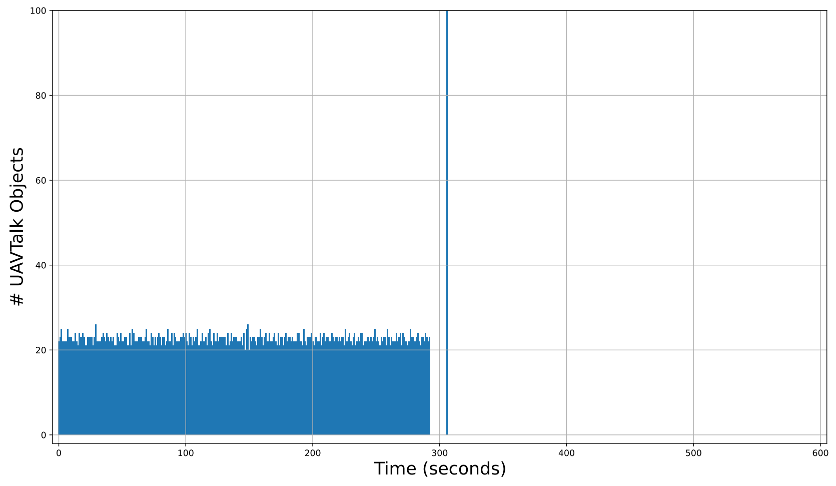
During the intensive flight tests with the drone already integrated with the platform, it was also detected a complete loss of connectivity in short flights; the stability was intensively tested again and did not reveal any signs of connectivity loss in periods lower than 24 hours. Further examination of the issue revealed that it was related to two factors: first, vibrations during flight briefly disconnected the emulated open serial channel; second, again as metioned in last paragraph, dRonin has an open issue about glitches, which leaves the emulated serial completely unresponsive after an abrupt disconnect. The dRonin developers claim[1] that significant efforts will be done in the next release to fix the issue.

Finally, the UAVTalk object called "UAVO_FlightBatteryState" is one of the telemetrey objects transmitted in the telemetry of the drone. Within this object, there is a

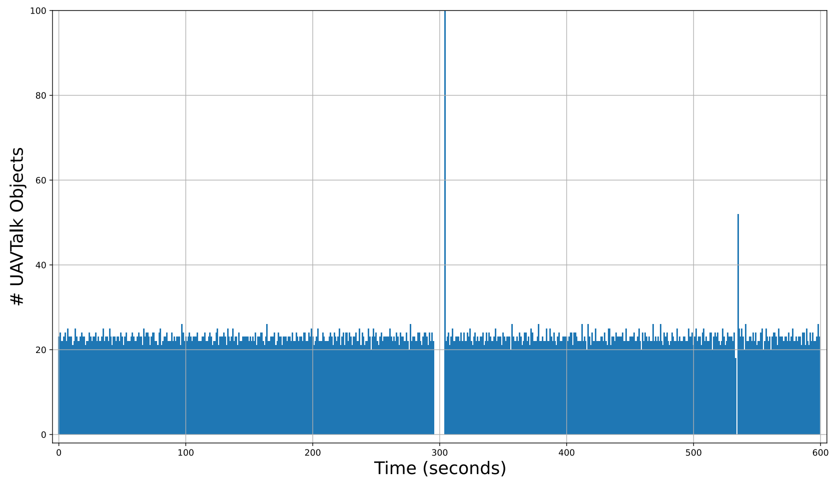
[1]https://github.com/d-ronin/dRonin/releases/tag/Release-20170717



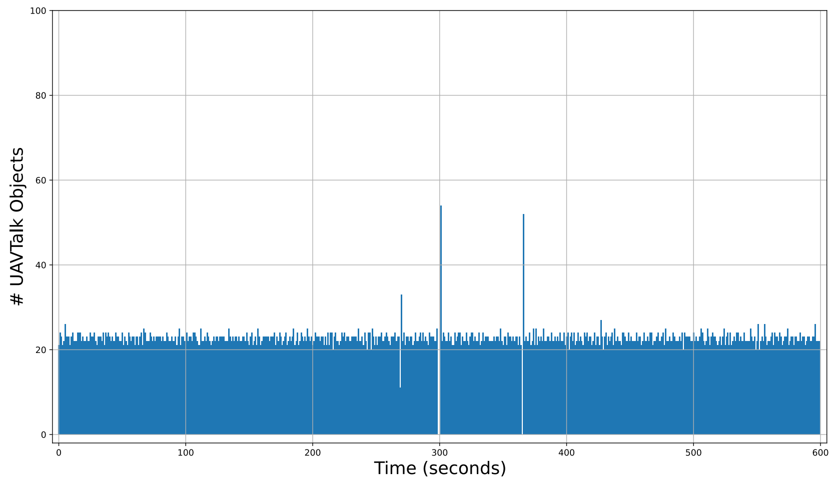
(a)



(b)



(c)



(d)

Figure 5.4: Examples of telemetry irregularities over long periods of time.

measurement called **estimatedFlightTime**. This measurement is calculated within the FC and relates to the expected amount of time the drone has left to fly safely before depleting its battery pack. The measurements were often inaccurate either by displaying unrealistic total flight times or negative flight times with a half capacity battery pack. This behavior often leads the FC into triggering an internal fail-safe measurement that disables the ability to arm the drone, and is only solvable by rebooting the FC. Several attempts to track the origin of the issue were inconclusive, since several sensors were used, and over time all sensors showed that same behavior. We consider this a cumulative error within the estimation function implemented in dRonin, since other aspects of the sensor metrics, like current voltage and tension seem accurate; this situation leads to the disable of the internal fail-safe trigger of the FC.

5.3 Telemetry stream processing

The telemetry data that is transmitted by the drone, in most cases, is raw data from their sensors. To obtain enriched information, the platform needs to transform the raw data into more understandable information. This process is achieved through the Telemetry Stream Processor module which is implemented using the Kapacitor from InfluxData. Using the TICKscript language, seven distinct data transformations were built and implemented in Kapacitor, they are: drone_detect, battery_detect, battery_alert, drone_pds, drone_pds_count, drone_pds_acceleration, drone_pds_power.

Drone_detect and **battery_detect** are simple binary processors, they detect the presence/absence of a drone and its battery, respectively. In this way, the platform has an indirect capability to determine if a drone or battery is being connected or disconnected. In the particular case of drone_detect, it is also capable of sending notifications directly to several communication channels, in this case, a slack channel receives a message every time the platform detects the connection or disconnection of a drone.

The **Battery_alert** script can detect the current battery level. With this information, it determines if a given drone is below or above the safe-to-fly battery level. In the case a given drone is detected with a battery below the safe-to-fly level, it sends a notification to the drone and to a slack channel. In the drone, the Drone Controller disables the possibility to arm the drone as a safety feature.

Drone_pds and **drone_pds_count** are counters: the drone_pds counts the amount of telemetry messages a given drone has sent in any given second; the drone_pds_count counts the amount of *drones* that sent telemetry messages in any given second. This information is useful for platform debugging purposes and to ensure that the platform is working as expected.

Drone_pds_acceleration and **drone_pds_power** are raw telemetry data transformers: the first takes the accelerometer 3-axis data and computes the total acceleration value; the second takes the battery current and tension to compute the power consumption. Both calculate these values for each given drone and metric message received. The information computed is afterwards stored in the time series database and displayed in the diagnostics dashboard, like the representation in Figure 5.5.

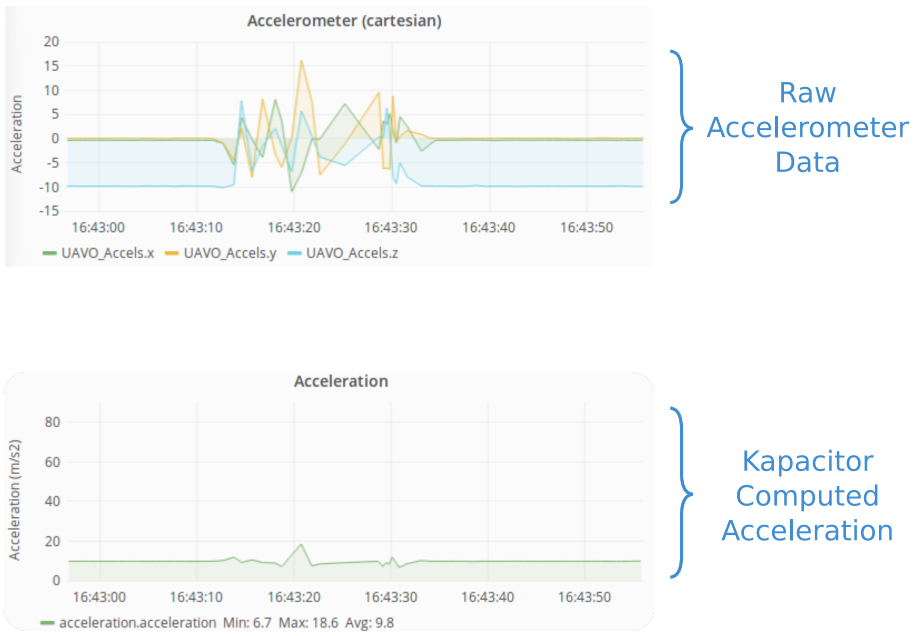


Figure 5.5: Example of telemetry data conversion with the Telemetry Stream Processor module.

5.4 Platform Performance

To ensure that the platform performs well with multiple simultaneous active drones, an emulator that mimics the behavior previously mentioned in this chapter has been developed.

With data similar to the one used for the analysis of the telemetry behavior of the drone, the emulator uses a previously generated file and mimics the rules contained within the file as accurate as a real drone; the only noticeable difference is that the values in the telemetry messages are always the same; however, it still accurately mimics the load on the platform in an end-to-end network load and throughput.

Networking delays are a fact that our platform has to deal with. Naturally, networking delays are not constant and depend on multiple factors (e.g., number of networks it has to go through, wireless or physical links, amount of data being transmitted). Although these networking delays change, they normally keep stable values under normal usage circumstances.

To abstract from networking delays and have accurate timings about clock synchronization, the platform experiments are conducted within the virtual machine used to deploy the Ground System. Each drone is emulated using a container instance that is running the Drone Broker and the emulation application; there is also a container that is responsible to spawn each drone emulator application within each container. These experiments are performed with powers of two numbers for the amount of drones in each trial, and lasted for one hour periods. Notice that these results in a real-life scenario would be higher due to the networking delays; even in extreme cases (over 10 seconds) of delay, the platform would still work and process the telemetry from the drone, but with the consequence of the inherent delay. As future work this should be addressed with a module which can detect the delays, and if needed, safely land the drone.

The results in Figure 5.6 show that the platform performs as expected with up to 64 emulated drones, in which the communications keep stable for the entire length of the experimentation period, and delivering a median and weighted average of 1 and 1.22 milliseconds of delay, respectively. Moreover, the results in Figure 5.7 show that telemetry messages are received at a stable rate of ∼22.8 messages per second per drone.

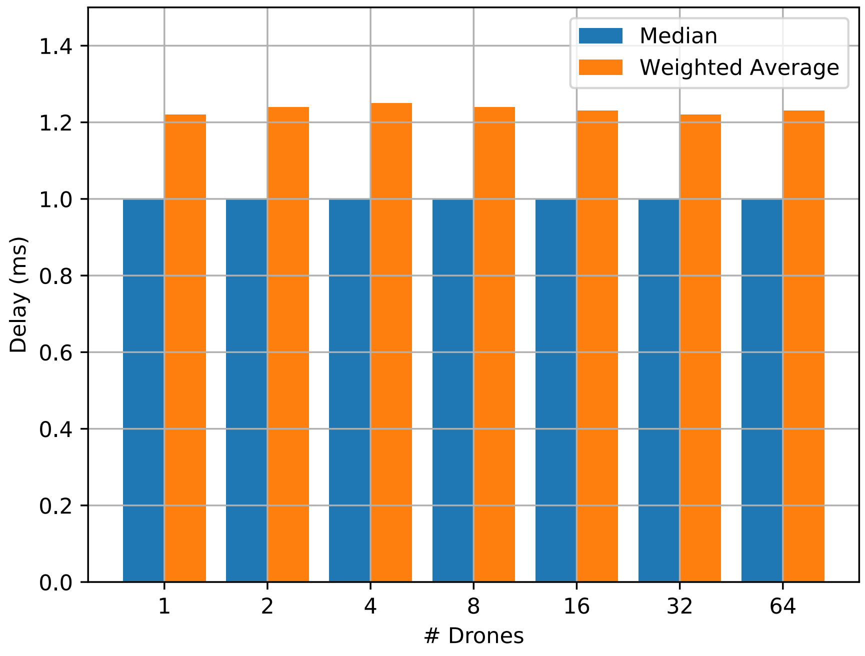


Figure 5.6: Median and weighted average delay values for up to 64 drones and 1 hour emulation.

To validate calculations made towards the median and weighted average, further analysis is performed into message count by delay and cumulative distribution probability. The results shown in Figure 5.8 validate the previous results and clearly demonstrate that more than 95% of the messages were received within 5 milliseconds, and a vast majority of messages were received with 1 millisecond of delay, representing ∼4.8 million messages count; the highest delay value was 148 milliseconds.

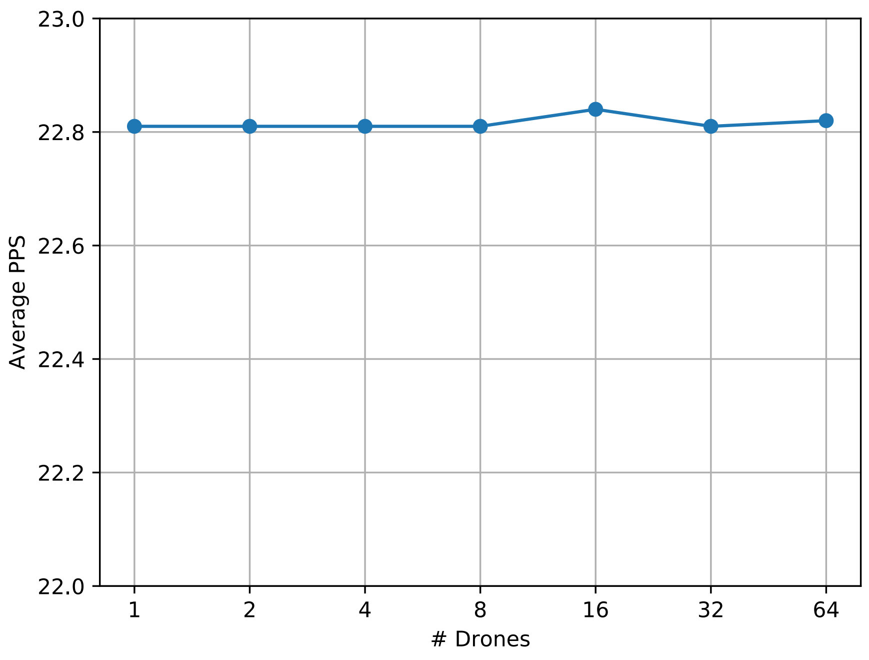


Figure 5.7: Average message count by drone for up to 64 drones and 1 hour emulation

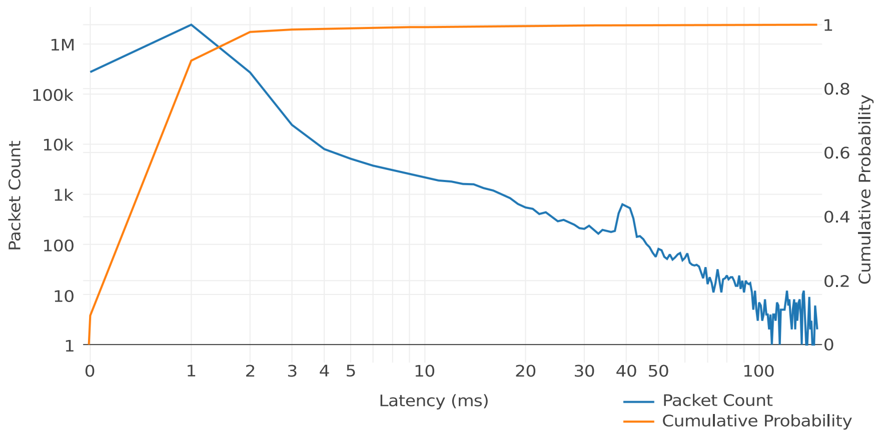


Figure 5.8: Message count by delay and cumulative distribution of messages in relation to delay with 64 drones.

5.5 Real-life drone deployment

To evaluate the platform in a real world environment, several experiments were conducted to test the entire platform: some aimed to perform simple tasks like make the drone climb 1 meter, arm, disarm and so on; others aimed to just gather telemetry data to further understand the behavior of the drone in-flight.

The results showed that the direct control abstraction and decoupling worked as expected. During the tests the platform was able to properly command the *drone* through multiple instances of the Control Dashboard as well as multiple clients controlling the drone, ignoring priority and concurrency issues to concurrent clients. Tests in this field used simultaneously the dashboards presented in Figures 4.14 and 4.15, in terms of multiple controlling devices, multiple mobile phones and laptops.

Near to real time telemetry was also shown to work properly using the Grafana dashboard which was presented in Figure 4.13. The only downside of this solution is the 5 seconds delay resultant from the maximum allowed refresh rate of 5 seconds inherent to the Grafana implementation.

The integration of Telemetry modules and Grafana also allowed the exploration of telemetry data from the drones while filtering it and integrating multiple sensors. To track its current relative position, dRonin uses a three-dimensional reference frame and a vector; its axis are called North, East and Down because they are referenced to the magnetic north, and east which represents the perpendicular axis in the horizontal plane oriented to the magnetic east; down is the vertical axis of the frame, which is positive towards the ground, therefore pointing down.

Atmospheric pressure decreases in relation to altitude; therefore, it can be combined with the "down" axis of the drone. Figure 5.9 is a representation of a 10 minute flight where it is analyzed that atmospheric pressure and "down" are indeed related and, as expected, dRonin uses pressure values to obtain a the "down" value.

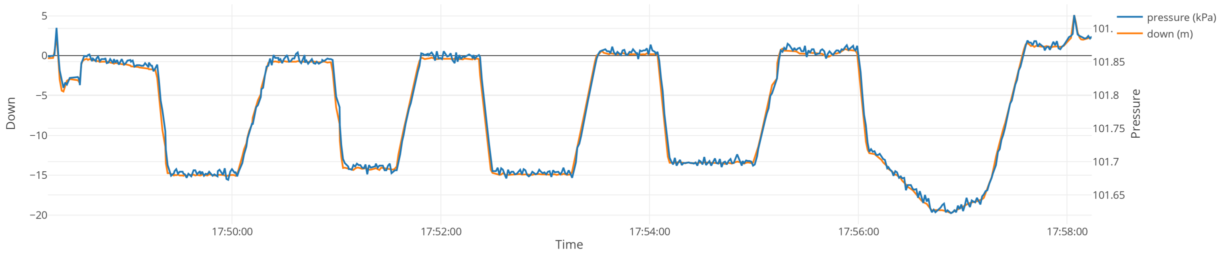


Figure 5.9: Atmospheric pressure vs "Down" from in-flight telemetry.

In a further more complex combination, a test focused in trying to understand if it is possible to extrapolate what is the drone doing from its current power consumption. Notice that power consumption is not directly obtainable from the telemetry data; therefore the Telemetry Stream Processor needs to be actively computing the current power consumption values.

The experiment consisted in the drone climbing and descending in intervals while maintain its current position (hovering): a total of 5 climbs and descents of ∼15 meters were performed in the test. The results in Figure 5.10 show that it is possible to extrapolate the current drone behavior from the actual power consumption: in idle it consumes very little current, 13 Watts; also climb and descent is distinguishable from each other, denoted in the figure by two horizontal dashed lines (it is shown that climb consumes ∼600 Watts and descent ∼500 Watts, which is slightly less consumption than while hovering).

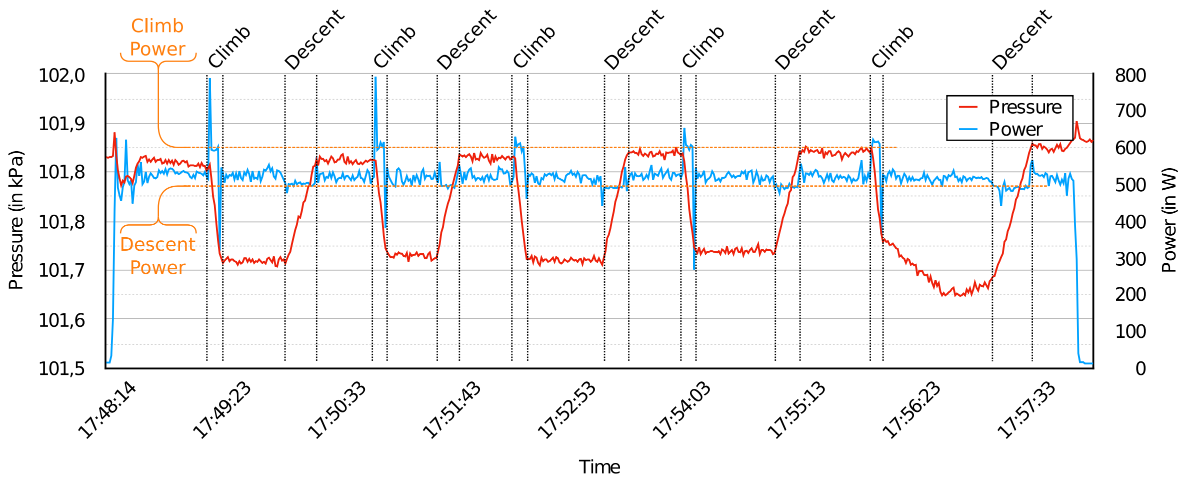


Figure 5.10: Power consumption versus drone movement analysis.

Other similar tests have shown that differences between hovering and moving are not so different in terms of power consumption, and that when climb/descent rate is slow, it is also not differentiated.

5.6 Summary

The resulting telemetry behaviors analyzed demonstrated how telemetry data flowed out of the drone. A comparison was made with the data output from the platform and the data obtained directly from the drone, in order to understand if there was any behavior change between the drone and the platform output. Such analysis concluded that the behavior was kept the same, even with high load of up to 64 emulated drones concurrently transmitting telemetry data. Finally, this chapter performed an analysis of several potential key telemetry data sources that can be used to detect events, malfunctions or data extrapolation for safe autonomous operation and diagnosis of drones.

Chapter 6

Conclusions and Future Work

This thesis aimed to provide an abstraction and decoupling platform to control Unmanned Aerial Vehicles (UAVs), which can improve integration and extension capabilities of current state of the art drones. The development process of the platform addressed the issues about: (1) how to access the drone; (2) how can the platform interact with it and what sort of behavior does the interactions with the drone have; (3) develop an abstraction and decoupling solution to allow the platform to interact with the drone; (4) develop modules to extend or improve the drone functionalities through the platform; and finally, (5) develop proof-of-concept applications that take advantage of the functionalities of the platform. The entire development process of the platform was achieved with the following characteristics:

- Access to the drone was done via Serial Communications emulated through Universal Serial Bus (USB) cable.
- Interactions were achieved through the usage of a Python Application Programming Interface (API) that used UAVTalk objects to communicate with the drone. The behavior of this API was proven to be stable enough for the platform to safely interact with the drone, but has known issues that are derived from a firmware glitch with the chosen Flight Controller (FC).
- Abstraction was achieved by using an adapter design pattern module called Drone Controller which translated platform commands into UAVTalk Objects and vice versa. Decoupling was achieved by using a modular development approach to the platform, and the extensive usage of Brokers to abstract the underlying communication mechanisms and routing.
- Several modules were developed to extend and improve the drone capabilities, like: the Drone Controller, Telemetry Stream Processor, and Telemetry Analyzer.
- Two web applications were developed to take advantage of the platform capabilities: the Control Dashboards that allows to control any drone connected to the platform,

and the Diagnostics Dashboard that allows for live telemetry observation of all connected drones, as well as serve as a remote Flight Data Recorder (FDR) black box of the current aircraft.

Furthermore, the proposed platform in this thesis was also tested in its capability to handle multiple drones simultaneously through drone emulations, which revealed that it is capable of handling up to 64 concurrent drones without significant deterioration to the normal communication delays and throughput. With the use of the proof-of-concept applications, we are also confident that the platform is a value added tool to further develop complex use cases for aerial drones.

6.1 Future Work

Considering that the work developed in this thesis comprises the first steps for a completely autonomous flying drones platform, there are several directions for the future research that are presented in this section. In the field of the work done in this thesis, there are some elements that should be improved, for instance: the complete removal of the Kapacitor implementations in the Telemetry Analyzer and Telemetry Stream Processor modules; the way Kapacitor was developed by InfluxData for the community version (Kapacitor simply cannot withstand the volume of data that the platform is capable of handling and therefore representing a bottleneck for the entire platform); the Dashboards and Drone Manager have many features that can now be developed on top of the current work; finally, the Mapper and Fail-safe modules were discussed as architectural modules for the platform, but were never developed because they were not required to develop the foundations of the platform.

Furthermore, and considering the extension of the platform and drone capabilities, several fields have an interesting room to considerably improve the architecture. Some of such fields are the following:

Multi-drone functionality The platform is already capable of handling multiple drones in the sense of control and telemetry, but further development efforts should extend these capabilities to drone cooperation and drone flight formations. The first addresses scenarios where drones can cooperate with each other to achieve more complex goals that would be more difficult or even impossible with single drones; the second addresses the difficulties of coordinating several simultaneous drones in close formations (e.g.: convoy, line or delta formations).

Drone-to-Drone communications There is room for relieving the Gound System of unnecessary communications like the ones between drones: if two drones are operating near each other, it is relevant that both have the location of each other to avoid collisions; in the context of cooperative work, it is relevant that both have shared information about the work that each of them is performing. Currently, such scenarios would require the Ground System to work as intermediary relay to support Drone-to-Drone communications.

Telemetry Analyzer The current implementation of the module is a simplistic proof-of-concept application. There is a large room to further expand this module in terms of reliability of autonomous drone operations without human supervision. Concepts like Machine Learning or Deep Learning are some of the possible forms that can definitely increase the capabilities of the Telemetry Analyzer module.

Drone Manager Integrations As explained in this thesis, this module is the integration point for other modules or applications to interact with drones. Further development of this module is already suggested, but this module should also have a special recommendation. For increased functionality, high level commands like area patrolling or mapping can benefit the overall capabilities of the platform; note that these are not simply high-level commands to be issued only to the drone(s). In the first case it is a complex routine that can be scheduled or triggered by external events; the second is a complex mechanism that has to autonomously compute a viable path that covers the pretended area, flies the drone, captures the necessary data, delegates such data to a distinct module/application, waits for the data process to finish, and then returns the requested data to the user.

Fail-safe System(s) In order to go beyond the detection of issues on the drone (malfunction or failures), the platform should be able to actively try to mitigate the consequences of such events. This requires the platform to be rigged with fail-safe capabilities and equipment like parachutes that can be deployed in the case of critical malfunctions, warning buzzers that can warn nearby persons to the presence of the drone, and safe to land areas where drones can safely land even in the case of malfunctions.

Latency Module The platform is, to some extent, delay sensitive. These delays can exist specially over the two edge entities (the user and the drone) because of their mobility, since it is expected that they may end up in a low network coverage area and that would result in aggressive delays. For the case of aggressive delays from the user connection, the issue is not that relevant, as depending of the task at hand it may not require constant user interaction and may only disrupt the telemetry data reaching the user. On the other hand, an aggressive delay on the drone connection would become potentially catastrophic, since the platform would lose the desired situation awareness from the drone. For these events, a module should constantly monitor the platform for sources of networking delay, with special attention to the connection between the drone and the platform, and in the case of abnormal networking delays, it should abort the mission and safely land the drone.

Bibliography

[1] J. F. Keane and S. S. Carr, "A brief history of early unmanned aircraft," in *Johns Hopkins APL Technical Digest*, 2013, pp. 559–570.

[2] M. Erdelj, E. Natalizio, K. R. Chowdhury, and I. F. Akyildiz, "Help from the Sky: Leveraging UAVs for Disaster Management," *IEEE Pervasive Computing*, vol. 16, no. 1, pp. 24–32, 2017.

[3] D. Moura, L. Guardalben, M. Luís, and S. Sargento, "A drone-quality delay tolerant routing approach for aquatic drones scenarios," in *IEEE IEEE Globecom 2017 - 8th International Workshop on Wireless Networking and Control for Unmanned Autonomous Vehicles*, December 2017, pp. –.

[4] John Wiley & Sons, *Introduction to UAV Systems*, 2012.

[5] "About DJI." [Online]. Available: https://www.dji.com/company

[6] "Drones DJI." [Online]. Available: https://www.dji.com/products/enterprise#drones

[7] "DJI A3." [Online]. Available: https://www.dji.com/a3

[8] "DJI N3." [Online]. Available: https://www.dji.com/n3

[9] "About MicroPilot." [Online]. Available: https://www.micropilot.com/about.htm

[10] "Contacts MicroPilot." [Online]. Available: https://www.micropilot.com/contact.htm

[11] "2028g News MicroPilot." [Online]. Available: https://www.micropilot.com/news-2003-jul-15.htm

[12] "FC Comparison MicroPilot." [Online]. Available: https://www.micropilot.com/pdf/brochures/brochure-MP2x28.pdf

[13] "Store MicroPilot." [Online]. Available: http://store.micropilot.com/category-s/36.htm

[14] "About Emlid Tech." [Online]. Available: https://www.facebook.com/pg/emlidtech/about/

[15] "Navio2." [Online]. Available: https://emlid.com/navio/

[16] "EDGE." [Online]. Available: https://emlid.com/edge/

[17] "ETHZ News Pixhawk." [Online]. Available: https://www.ethz.ch/en/news-and-events/eth-news/news/2016/01/eth-software-to-become-standard-for-drones.html

[18] "About Pixhawk." [Online]. Available: http://px4.io/about-us/

[19] "Pixhawk2." [Online]. Available: https://www.pixhawk.org/modules/pixhawk2

[20] "Pixhawk store." [Online]. Available: http://pixhawkstore.com.au/

[21] "PX4 Arch." [Online]. Available: https://dev.px4.io/en/concept/architecture.html

[22] "Home OpenPilot." [Online]. Available: https://web.archive.org/web/20110527225811/http://www.openpilot.org/

[23] "About OpenPilot." [Online]. Available: https://web.archive.org/web/20110613035025/http://www.openpilot.org:80/about/

[24] "Facebook OpenPilot." [Online]. Available: https://www.facebook.com/pg/OpenPilot/about/?ref=page_internal

[25] "OpenPilot Revolution." [Online]. Available: http://opwiki.readthedocs.io/en/latest/user_manual/revo/revo.html

[26] "About TauLabs." [Online]. Available: http://taulabs.org/about.html

[27] "Sparky2 TauLabs." [Online]. Available: https://github.com/TauLabs/TauLabs/wiki/Sparky2

[28] "About Ardupilot." [Online]. Available: http://ardupilot.org/about

[29] "Case Studies Ardupilot." [Online]. Available: http://ardupilot.org/casestudies

[30] "Flight Modes Ardupilot." [Online]. Available: http://ardupilot.org/copter/docs/flight-modes.html

[31] "Arch Ardupilot." [Online]. Available: http://ardupilot.org/dev/docs/apmcopter-code-overview.html

[32] "About Dronecode." [Online]. Available: https://www.dronecode.org/about/

[33] "Flight Modes PX4." [Online]. Available: https://docs.px4.io/en/flight_modes/offboard.html

[34] "Advanced Modes PX4." [Online]. Available: https://docs.px4.io/en/advanced_features/

[35] "OpenPilot Git." [Online]. Available: https://github.com/openpilot/OpenPilot

[36] "TauLabs Git." [Online]. Available: https://github.com/TauLabs/TauLabs

[37] "Home dRonin." [Online]. Available: https://dronin.org/

[38] "Flight Modes dRonin." [Online]. Available: https://github.com/d-ronin/dRonin/wiki/Flightmode-Settings

[39] "GCS dRonin." [Online]. Available: https://dronin.org/docs/initial-setup/getting-started/

[40] "Supported FCS dRonin." [Online]. Available: https://github.com/d-ronin/dRonin

[41] "API Ardupilot." [Online]. Available: http://ardupilot.org/dev/docs/code-overview-adding-a-new-mavlink-message.html

[42] "API PX4." [Online]. Available: https://mavlink.io/en/

[43] "API dRonin." [Online]. Available: https://github.com/d-ronin/dRonin/wiki/Development-UAVTalk-Protocol

[44] "GCS UAVNavigation." [Online]. Available: https://www.uavnavigation.com/products/gcs/gcs-software

[45] "About UAVNavigation." [Online]. Available: https://www.uavnavigation.com/company

[46] "About Airware." [Online]. Available: https://www.airware.com/en/about/

[47] "Platform Airware." [Online]. Available: https://www.airware.com/en/platform/

[48] "Home PrecisionHawk." [Online]. Available: https://www.precisionhawk.com/

[49] "About PrecisionHawk." [Online]. Available: https://www.precisionhawk.com/about

[50] "PrecisionFlight PrecisionHawk." [Online]. Available: https://www.precisionhawk.com/precisionflight-pro

[51] "PrecisionViewer PrecisionHawk." [Online]. Available: https://www.precisionhawk.com/precisionviewer

[52] "PrecisionMapper PrecisionHawk." [Online]. Available: https://www.precisionhawk.com/precisionviewer

[53] "About DroneDeploy." [Online]. Available: https://www.dronedeploy.com/about.html

[54] "Carrers DroneDeploy." [Online]. Available: https://www.dronedeploy.com/careers.html

[55] "MobileApp DroneDeploy." [Online]. Available: https://www.dronedeploy.com/app.html

[56] "Cloud DroneDeploy." [Online]. Available: https://www.dronedeploy.com/enterprise.html

[57] "About FlytBase." [Online]. Available: https://flytbase.com/about-us/

[58] "LinkedIn FlytBase." [Online]. Available: https://in.linkedin.com/company/flytbase

[59] "Platform FlytBase." [Online]. Available: https://flytbase.com/platform/

[60] "FlytOS FlytBase." [Online]. Available: https://flytbase.com/flytos/

[61] "FlytCloud FlytBase." [Online]. Available: https://flytbase.com/flytcloud/

[62] "About MultiDrone." [Online]. Available: https://multidrone.eu/multidrone-in-short/

[63] J.-t. Amenyo, D. Phelps, O. Oladipo, F. Sewovoe-ekuoe, 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," *2014 IEEE Global Humanitarian Technology Conference*, 2014.

[64] Z. Deng, C. Ma, and M. Zhu, "A reconfigurable flight control system architecture for Small Unmanned Aerial Vehicles," *2012 IEEE International Systems Conference SysCon 2012*, pp. 1–4, 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6189451

[65] M. Saska, J. Chudoba, L. Precil, J. Thomas, G. Loianno, A. Tresnak, V. Vonasek, and V. Kumar, "Autonomous deployment of swarms of micro-aerial vehicles in cooperative surveillance," *2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014 - Conference Proceedings*, pp. 584–595, 2014.

[66] D. Câmara, "Cavalry to the rescue: Drones fleet to help rescuers operations over disasters scenarios," *2014 IEEE Conference on Antenna Measurements and Applications, CAMA 2014*, 2014.

[67] C. A. Thiels, J. M. Aho, S. P. Zietlow, and D. H. Jenkins, "Use of unmanned aerial vehicles for medical product transport," *Air Medical Journal*, vol. 34, no. 2, pp. 104–108, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.amj.2014.10.011

[68] P. Tripicchio, M. Satler, G. Dabisias, E. Ruffaldi, and C. A. Avizzano, “Towards Smart Farming and Sustainable Agriculture with Drones,” *Proceedings - 2015 International Conference on Intelligent Environments, IE 2015*, pp. 140–143, 2015.

[69] M. Rossi and D. Brunelli, “Autonomous Gas Detection and Mapping With Unmanned Aerial Vehicles,” *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 4, pp. 765–775, apr 2016. [Online]. Available: http://www.tandfonline.com/doi/full/10.1080/17512786.2014.980596http://ieeexplore.ieee.org/document/7369958/

[70] L. P. Morrison, B. Team, B. Nguyen, S. Kannan, N. Ray, and G. C. Lewin, “AirChat : Ad Hoc Network Monitoring with Drones,” pp. 38–43, 2017.

[71] N. Nigam, S. Bieniawski, I. Kroo, and J. Vian, “Control of multiple UAVs for persistent surveillance: Algorithm and flight test results,” *IEEE Transactions on Control Systems Technology*, vol. 20, no. 5, pp. 1236–1251, 2012.

[72] M. Asadpour, B. Van Den Bergh, D. Giustiniano, K. A. Hummel, S. Pollin, and B. Plattner, “Micro aerial vehicle networks: An experimental analysis of challenges and opportunities,” *IEEE Communications Magazine*, vol. 52, no. 7, pp. 141–149, 2014. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84904677301{\&}partnerID=40{\&}md5=52929618d18d634d2e3e2c685ae05046

[73] J. Wang, C. Jiang, Z. Han, and Y. Ren, “Taking Drones to the Next Level,” *IEEE Vehicular Technology Magazine*, vol. 12, no. 3, pp. 73–82, 2017.

[74] A. G. Foina, R. Sengupta, P. Lerchi, Z. Liu, and C. Krainer, “Drones in smart cities: Overcoming barriers through air traffic control research,” *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pp. 351–359, 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7441027/

[75] C. Alex and A. Vijaychandra, “Autonomous cloud based drone system for disaster response and mitigation,” *International Conference on Robotics and Automation for Humanitarian Applications, RAHA 2016 - Conference Proceedings*, pp. 1–4, 2017.

[76] A. Koubaa, B. Qureshi, M.-F. Sriti, Y. Javed, and E. Tovar, “A service-oriented Cloud-based management system for the Internet-of-Drones,” in *2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, no. Section 3. IEEE, apr 2017, pp. 329–335. [Online]. Available: http://ieeexplore.ieee.org/document/7964096/