POLITECNICO DI TORINO

Faculty of Aerospace Engineering

Master Thesis

Development of an Open-Source Flight Controller for Rotary Wing UAVs

Supervisor
Giorgio Guglieri

Co-supervisor:
Simone Godio

Candidate:
Alessandro Minervini

April 2021

To my family.

Acknowledgements

I would like to acknowledge the co-supervisor of this thesis Simone Godio who contributed to this work with his experience and enthusiasm. I would also like to thank all the PIC4SeR staff for allowing me to access their laboratory instrumentation and collect some required data for this thesis.

Abstract

Due to the growing applications involving Unmanned Aerial Vehicles (UAVs), Flight Controllers for rotary wing UAVs are playing an important role in guaranteeing proper flight performances and high stability. Many companies are investing to improve drones' reliability and extend their use to several applications ten years from now. Flight performance of drones needs to be improved to meet this increasing demand, and researches about best-performing control strategies have been carrying on.
To increase flight performance and control strategies efficiency, an "open" platform is needed, so that the Flight Controller software can be fully accessible and easily modifiable. In the following thesis, an Open-Source Flight Controller for rotary wing UAVs is developed and tested in a real environment, while a Linear Quadratic Regulator (LQR) controller is implemented to perform the auto-stabilizing function and maneuvres.

Arduino STM-32 board is chosen as the micro-controller, and it is programmed by using Arduino Integrated Development Environment (IDE).
An Inertial Measurement Unit (IMU) is developed to perform sensor fusion between accelerometer and gyroscope sensors. For this purpose, a complementary filter is implemented in the discrete-time domain.
Particular attention is given to the software working frequency and the Pulse Width Modulated (PWM) signals generated by the micro-controller: STM-32 interrupt logic and timers documentation are analyzed in detail to avoid delays and provide a fast response to disturbances by motors.
The developed software is also enabled to communicate with a GNSS/GPS module, providing the Flight Controller with the longitude and latitude for position control and autonomous flight.

Before implementing the control algorithm, a simulation model of the quadcopter is developed in a Simulink environment to preliminary tune the LQR controller.
Finally, flight tests are performed, and satisfactory results about pitch, roll, and yaw response to commands demonstrate the capability of the developed Flight Controller to perform maneuvres properly. The LQR controller is found to be a valid alternative to the most common Proportional-Integrative-Derivative (PID) controller, and an efficient open-source platform to improve flight performances of quadcopters is provided.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Unmanned Aerial Vehicles (UAVs) are aircraft without human pilots on board. UAVs communicate with a ground station to send data and to receive commands from a transmitter, while most of them are also able to perform the autonomous flight, following a specific flight path or maintaining the hover flight condition.

At the beginning of the XX century, UAVs started to be considered a great resource for military applications and the forthcoming World Wars contributed to the growth of the research in autonomous flight and the production of these vehicles. After World War I, some vehicles were converted to pilotless aircraft thanks to advances in radio-controlling. It was the case of the de Havilland DH.82 Tiger Moth, which was renamed as DH.82B Queen Bee (Fig. 1.1). The Queen Bee was developed for aerial target practice and almost 300 copies were produced [1]. Many consider the Queen Bee as the first modern drone in history.

In 1943, First Person View Flights began: Boeing and the U.S. Airforce developed the BQ-7 (Fig. 1.1) converting B-17 models to radio-controlled assault drones. The aircraft was manned by a crew of two during the take-off and the initial climb, and after activating remote control, it was able to fly to the target on its own.
During the late 1950s, research programs aimed at developing surveillance UAVs. The

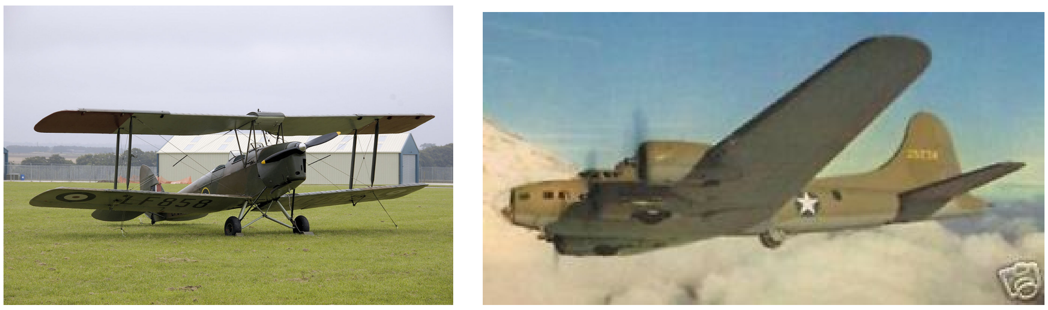
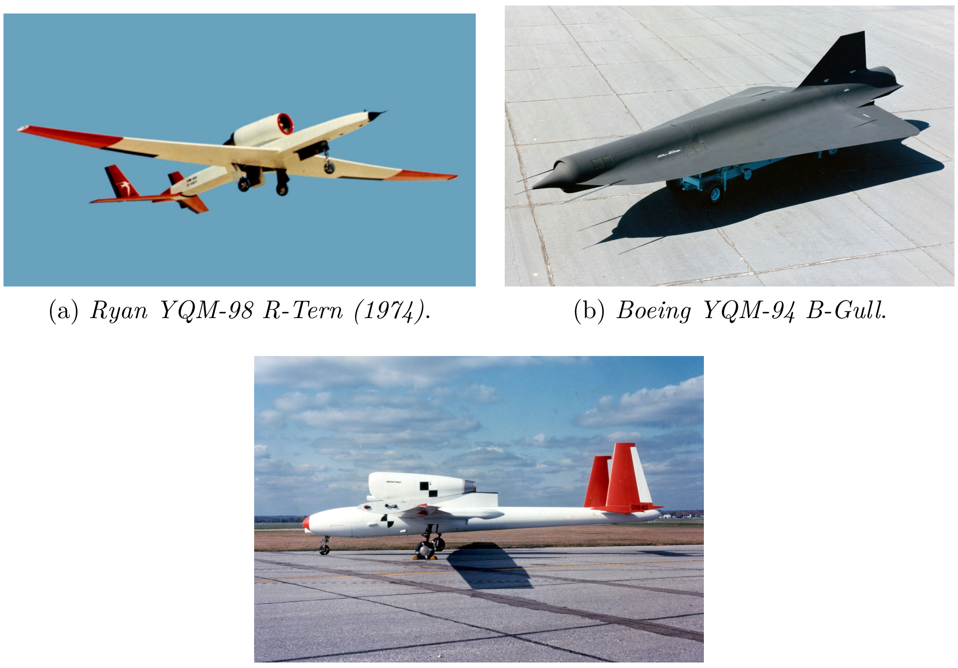

(a) *DH.82B Queen Bee.* (b) *BQ-7.*

Figure 1.1: The Queen Bee (a) aerial target practice, and the BQ-7 (b)

US especially meant to flew them over China and North Vietnam, obtaining information without risking the lives of pilots [2]. About ten years later, the first reconnaissance drones were developed from the US like the Ryan YQM-98 R-Tern (also called Compass Cope R, the Boeing YQM-94 B-Gull (also called Compass Cope B), and the Lockheed D-21 (Fig. 1.2).

In the early 1980s, it was clear that drones would have a growing role on the bat-


(a) *Ryan YQM-98 R-Tern (1974).* (b) *Boeing YQM-94 B-Gull.*

(c) *Lockheed D-21 (1964).*

Figure 1.2: Surveillance UAVs for military service

tlefields of the future [1]. Israel's UAVs program obtained great results, and some of Israel's models were purchased by the United States or produced under license. AAI RQ-2 Pioneer (Fig. 1.3) is an unmanned aerial vehicle produced from 1986 to 2007. It was developed by the Aircraft Armaments, Inc (AAI) Corporation, and Israel Aircraft Industries, and it was able to accommodate a greater payload than its previous versions.

In the 1990s MQ-1 Predator (Fig. 1.3) was developed. It was born as an aerial reconnaissance drone but later it was modified to carry and fire two AGM-114 Hellfire missiles. The MQ-1 Predator was produced until 2018 and it was employed in many missions such as the war in Afghanistan, Pakistan, the North Atlantic Treaty Organization (NATO) intervention in Bosnia, and recently in the 2014 intervention in Syria.

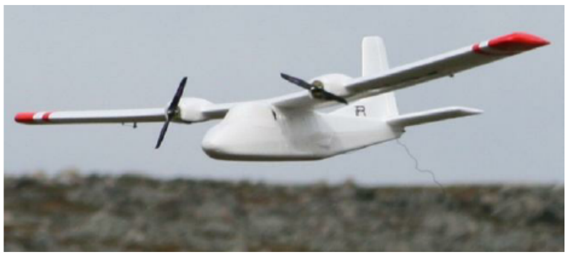
Figure 1.3: AAI RQ-2 Pioneer (1986-2007)

Figure 1.4: MQ-1 Predator (1995-2018)

1.1 UAVs typical configurations

Grouped in two categories, UAVs can be seen as rotary-wing and fixed-wing vehicles, according to the task to be carried out [3].
Fixed-wing UAVs (Fig. 1.5) are generally categorized according to the shape of their wing. Three main categories of fixed-wing UAVs exist, and they are straight wing, swept wing, and delta wing UAVs. These vehicles generate the lift through the wing itself, thanks to the forward airspeed provided by an internal engine or an electric motor propeller [4].
Rotary wing UAVs provide lift by one or more electric motor propellers which generate an upwards thrust. Multi-rotors are the most common rotary-wing UAVs, and they are divided into specific categories based on the number and location of motors on the frame. Common configurations are Tricopter, Quadcopter (Y, X configuration), Hexacopter, and Octacopter (Y, X configuration).

Fixed wing UAVs provide a good stability thanks to their configuration and they are able to perform a longer flight time than rotary wing UAVs because of the less power requirement. They are the best solution for covering large areas in a short time but they can not hover at a place or fly at low speeds. Another great limitation as regards fixed wing UAVs is related to the take off and landing: due to their configurations, these vehicles can only perform a Horizontal Take Off and Landing (HTOL), so they are not suitable for several applications.
Multi-rotors are generally preferred to fixed wing UAVs, thanks to their capability


(a) *Fixed straight wing UAV.*

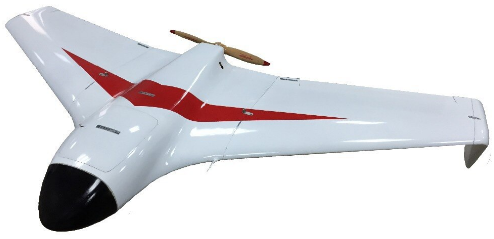
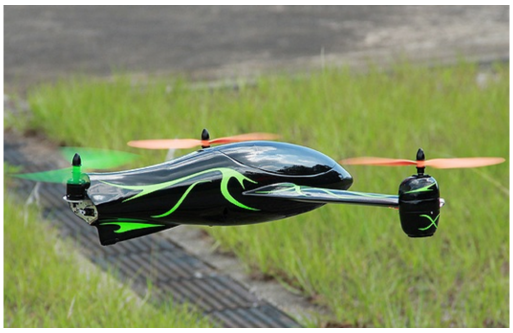
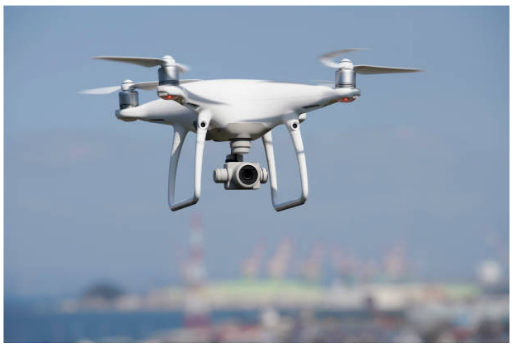
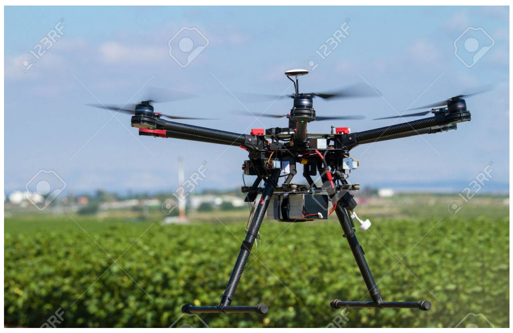

(b) *Fixed delta wing UAV.*

Figure 1.5: Two examples of fixed wing UAVs


(a) *Tricopter.*


(b) *Quadcopter.*


(c) *Hexacopter.*

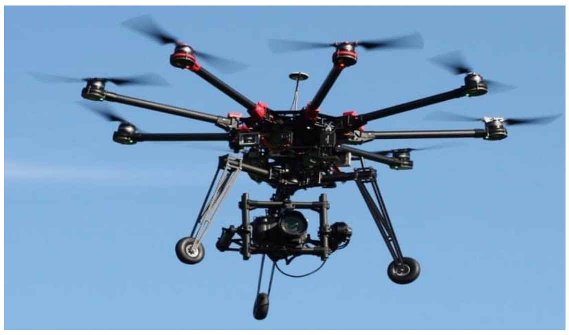

(d) *Octacopter.*

Figure 1.6: Most common configurations of rotary wing UAVs

to maintain hover flight condition and to perform the Vertical Take Off and Landing (VTOL). Instead of fixed wing, rotor wing UAVs are intrinsically unstable but they can provide good flight performance if robust control laws are implemented. In particular, quad-rotor is the most common configuration as it merges requirements about flight stability with a simple and feasibility configuration [5][6].

Quadcopters consist of four rotors that lift the drone and control its attitude and position by varying the RPM of each rotor. In order to avoid them from spinning around themselves, two opposite rotors spin in one direction, while the other two spin in the opposite direction [7]. In the following sections, modern UAVs' applications are presented. Since in this thesis a Flight Controller for rotary wing UAVs is developed, the attention is focused on rotary wings UAVs and, in particular, quadcopters.

1.2 Modern UAVs applications

During the last years, the role of UAVs drastically changed and their application regions extended to civilian applications, agriculture, law enforcement, and in the future, transport too. UAVs' configuration changed in favor of smaller dimensions to be more suitable for several applications, and researchers have been working on their design and development to increase the autonomy and endurance of these vehicles.

For many applications, Vertical Take-Off and Landing (VTOL) is preferred and good maneuverability and a stable hovering are required. Multi-rotor wing UAVs satisfy these requirements ensuring a suitable solution for several applications. In particular, the quad-rotor is the most common configuration which merges the requirement of flight stability with a simple and feasibility configuration.

Aerial surveillance

In both military and civil applications, UAVs have been employed for many years to perform aerial surveillance. The main task is to collect information from a specified area, for instance: remote areas where access is hampered by mountains, vast land areas without road networks, or areas afflicted by natural disasters. Surveillance applications include livestock monitoring, wildfire mapping, pipeline security, home security, and road patrol. Therefore, UAVs are becoming a valuable alternative for some police operations (Fig. 1.7) [8][9].

(a) *Police service in an urban context performed by an UAV.*

(b) *UAV employed in pipeline security.*

Figure 1.7: Two quadcopter UAVs in modern aerial surveillance applications

Package delivery

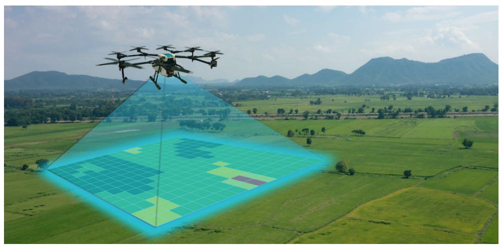
UAVs can significantly accelerate delivery times and reduce the human cost associated with the delivery [10]. Considering the increasing delivery activity and the more and more busy streets, exploiting the vertical dimension above cities would result in a significant reduction of the delivery times. This is the reason why some companies are investing in mini-drones for package delivery to consumers, especially during the last mile phase (Fig. 1.8). Companies and researchers are working to overcome the issues related to the safety of airspace, the privacy of citizens, and the tracking of these vehicles to make package delivery more suitable [11][12]. Other issues are related to the power supply which limits the delivery range and payload requirements.
In February 2021, the first test of an electric propulsion drone of 130 Kg (Fig. 1.9) has been performed in Turin by Leonardo company [13]. The project, named "Sumeri: Si salpa!", aims at delivering heavy goods up to hundreds of kilos in urban contexts.

Figure 1.8: Quadcopter UAV employed in package delivery

Figure 1.9: A capture of the 130 Kg drone employed by "Sumeri: Si Salpa" project [14]

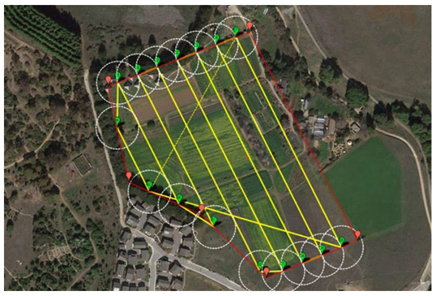
Surveying

A drone survey consists of the use of a drone to capture aerial data with downward-facing sensors, such as Red Green Blue (RGB) or multi-spectral cameras, and Laser Imaging Detection and Ranging (LIDAR) payloads (Fig. 1.10). During a drone survey with an RGB camera, the ground is photographed several times from different points of view, and each image is tagged with coordinates. Thus, photogrammetry software can be used to create a 3D model of the project area (Fig. 1.11) [15]. Unlike aircraft and satellites, UAVs can perform flight at a lower altitude, providing high-quality data in a shorter period of time than other vehicles. Also, the costs and maintenance of these activities are significantly reduced. [16]. The main tasks of survey operations are related to cartography, urban planning, and landing management. Using remote sensing cameras is also possible to detect water in dry areas and to find water leaks in underground water pipes.


(a) *UAV employed in surveying.*

(b) *UAV used in construction industry.*

Figure 1.10: Two UAVs employed in surveying operations


(a) *Picture collected from UAVs for mapping areas.*

(b) *UAV's image used for urban planning.*

Figure 1.11: UAVs' photography for mapping areas (a) and urban planning (a)

Agriculture

UAV technology has allowed for an optimized approach to various farming tasks, such as field mapping, plant stress detection, biomass estimation, weed management, inventory counting, and chemical spraying (Fig. 1.12) [17]. In the case of precision agriculture, it is necessary providing UAVs with additional devices such as multispectral cameras, thermal cameras, RGB cameras, and LIDAR systems. UAVs are also required to fly according to way-points, in order to maintain the hover condition and perform the obstacle avoidance [18]. Thanks to their capability of adjusting their altitude and following flight paths, UAVs can use their sensors to identify areas where performing crop spraying quickly and with great precision. Indeed, UAVs perform crop spraying five times faster than regular machinery.
Crop monitoring is another application of UAVs in agriculture: satellite imagery is very costly and data quality is largely affected by weather conditions. UAVs can perform crop monitoring more accurately and frequently providing higher quality data. To reduce excessive water usage, UAVs are also equipped with infrared and thermal sensors able to identify areas receiving too much or too little water, allowing the irrigation only to parts of a field which really need [19].

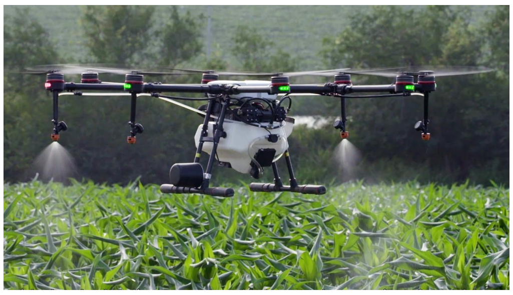
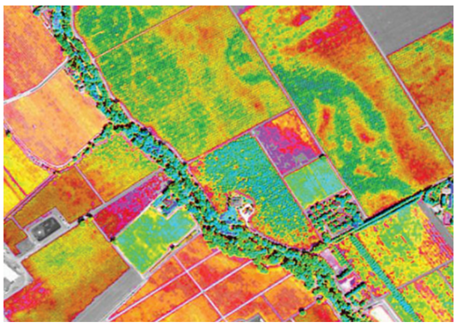
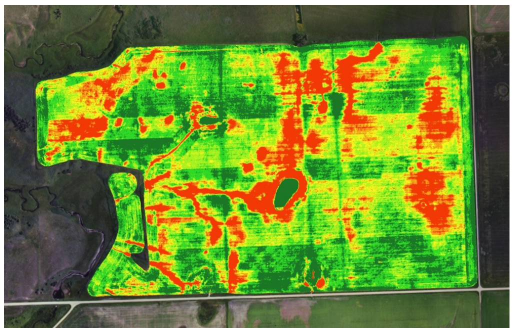


Figure 1.12: UAV employed in weed management



(a) *Image from UAV's thermal sensors for crop monitoring*



(b) *Normalized Difference Vegetation Index (NDVI) image to determine corp stress*

Figure 1.13: Imagery collected from UAVs in agricolture applications

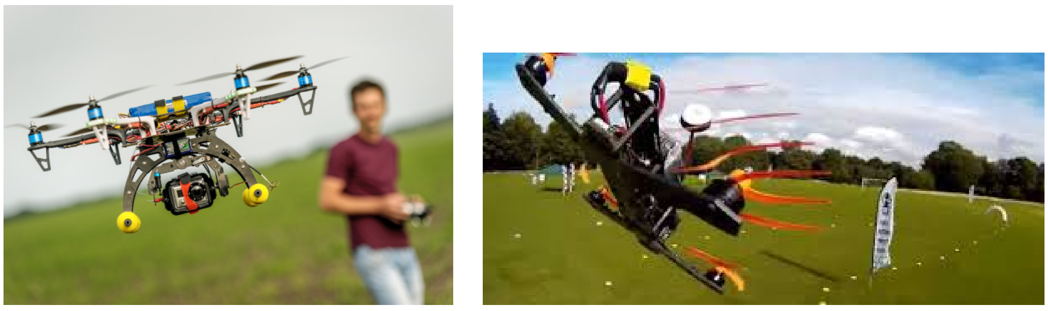
Cinematography

With the advent of stable and powerful quadrotors, and high-quality cameras mounted on controllable gimbals, quadrotors are becoming more and more useful cinematographic devices for both professional and amateur film-makers (Fig. 1.14) [20]. In the cinema industry, movie-makers use drones to produce shots with complex camera motion, and sometimes they represent the only way to reach viewpoints that would be inaccessible to other camera devices [21].

Figure 1.14: Quadcopter UAV on a cinematography set

Recreational flying

Several companies sell UAVs for hobbyists and recreational use (Fig. 1.15). From economic to more professional ones, UAVs are available on the market providing satisfactory performance for people who use them for filming and photographing, or drone racing. Even Do It Yourself (DIY) UAVs is a spread leisure activity that involves hobbyists who want to build up and customize their drones [22]. In compliance with the rules established from every country, it is possible to fly UAVs for personal uses: EASA and FAA respectively rule UAVs flying in European countries and USA [23] [24].



(a) *DIY for recreational use* (b) *Racing drone*

Figure 1.15: Two examples of drones for recreational flying

1.3 State-of-the-Art of Flight Controllers for UAVs

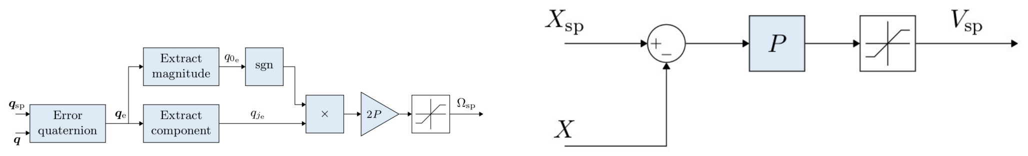
A Flight Controller for UAVs is the hardware component that ensures flight stability and makes Remote Control (RC) or autonomous flight possible. Its functions consist of obtaining UAVs' states (position, attitude, velocity, etc.) from sensors, converting radio commanded signals into actuator pulses, and ensuring proper flight performance while following signals or maintaining states (i.e. altitude and flight path).

Flight Controllers can be distinguished into two main categories: closed-source and open-source software. Closed-source software can not be modified in code and appear as "black boxes" to customers. To overcome this issue, some companies developed open-source Flight Controller just to enable customers of modifying software on their own and according to their purpose. Accessing to Flight Controller's software, industries and universities can collaborate on solving challenges related to UAVs platform, such as functionality, reliability, fault tolerance, and endurance, which all are tightly linked to the UAVs' Flight Controller hardware and software [25].

A brief description of open and closed-source Flight Controllers developed by some companies will follow. The main task of this section is to introduce Flight Controller's hardware and software this thesis takes inspiration from. Firstly, some open-source products for academic and some industrial applications are described. Afterward, more advanced Flight Controllers for professional applications are introduced.

1.3.1 PX4 series

The Pixhawk Flight Controller project started at the Computer Vision and Geometry Lab of ETH Zurich, before becoming an independent Open Source Hardware (OSH) platform project. Pixhawk collaborates with several partners, including the Linux Foundation DroneCode project. These Flight Controllers are based on the PX4-Flight Management Unit (FMU) and multiple versions have been developed. PX4 is used in a wide range of applications, from consumer drones to industrial applications. It is also the leading research platform for drones and has been successfully applied to underwater vehicles and boats.
As regards multicopter control algorithm (Fig. 1.16), PX4 makes use of quaternion for attitude control [26] and the desired angular rate is obtained through a PID controller. Position pass through a simple Proportional Controller which gives the desired velocity as output. Therefore, a PID converts velocity in a commanded acceleration.



(a) *PX4 Angle Controller.* (b) *PX4 Position Controller.*

Figure 1.16: PX4 control scheme

PX4 FMUv5

PX4 hardware evolved to FMUv5 at the end of 2017, when the processor was updated to STM32F7 with double-precision Floating Point Unit (FPU) and higher accuracy attitude calculations based on the Bosch BMI055 IMU [25].
The hardware consists of two accelerometer/gyroscope sensors for redundancy, a magnetometer, a barometer, and a Global Positioning System (GPS) module. It also offers from 8 to 16 PWM channels for Remote Control RC.
PX4 can be applied to several frame configurations, from different types of conventional take-off and landing aircraft to multicopters (quadcopter, octocopter, tricopter) and hybrid configurations, too.

Holybro PX4

It is based on the Pixhawk-project FMUv5 OHD and it is optimized to run PX4 version 1.7. It is suitable for academic and commercial developers, and it features more computing power (two times the RAM than previous versions), additional ports for better integration and expansion, new sensors, and integrated vibration isolation.

mRo Pixracer

This is a very small and light autopilot optimized for First Person View (FPV) racers (Fig. 1.17). It can be the best solution for any small frame that requires no more than 6 PWM outputs, and it has in-built Wi-Fi. The Pixracer is designed to use a separate avionics power supply. This is necessary to avoid current surges from motors or ESCs. Indeed, some currents could flow back to the Flight Controller and disturb its delicate sensors.

1.3.2 Paparazzi

Paparazzi UAV is an open-source drone hardware and software project that provides autopilot systems for multicopters/multirotors, fixed-wing, helicopters, and hybrid aircraft. The project is developed at Ecole Nationale de l'Aviation Civil (ENAC) UAV Lab.

(a) *PX4 FMUv5.*

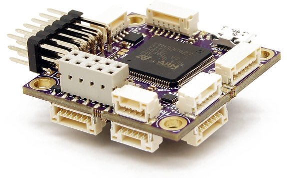

(b) *mRo Pixracer.*

Figure 1.17: PX4 series

Unlike other systems, Paparazzi UAV was designed with autonomous flight as the primary focus and manual flying as the secondary one. Actually, from the beginning, it was designed to control multiple aircraft within the same system.
Paparazzi also features a dynamic flight plan system that is defined by mission states and uses waypoints as variables. This makes it possible to carry on very complex and fully automated missions without the operators' intervention. The hardware consists of IR sensors, inertial sensors, complete IMU heads, voltage regulators, GPS receivers, converters, adapters, and programmers of all sorts [27]. In March 2017, ENAC Lab released a new autopilot named *Chimera* in collaboration with Paparazzi which is based on STM32F7 MCU [25]. In Fig. 1.18 *Chimera* circuit board is shown.

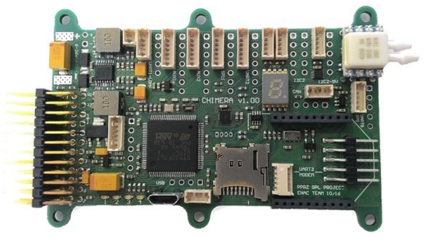


Figure 1.18: Chimera Autopilot

1.3.3 Ardupilot Mega (APM)

The ArduPilot project provides an advanced, full-featured, and reliable open source autopilot software system. The first ArduPilot open code repository was created in 2009. Since then, it has been developed by different teams of professional engineers, academics, and computer scientists.
It is capable of controlling almost any vehicle system: conventional and VTOL airplanes, gliders, multirotors, helicopters, sailboats, powered boats, submarines, ground

vehicles, and even balance-bots. Also, the supported vehicle types are frequently updated to extend its use to new and novel platforms [28].
The APM is an Arduino Mega-based autopilot, which has been developed by the DIY Drones community. The software was originally developed for 8-bit ARM-based MCUs, and to run on its own board ArduPilot. This board was however replaced by ArduPilot Mega (APM), and the software was updated to be run on 32-bit ARM-based MCUs. Nevertheless, the controller can run under Linux, enabling it to be used on a large class of electronics, such as single-board computers and all full PC systems. ArduPilot has a Geographic Coordinate System (GCS) desktop for mission planning, calibration, and vehicle set-up for Windows, Linux, and Mac OSX [28][25]. *Ardupilot Mega* is shown in Fig. 1.19.

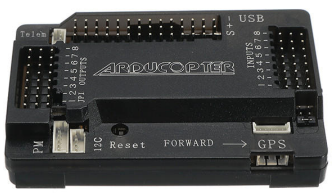


Figure 1.19: Ardupilot Mega (APM)

1.3.4 TopXGun

TopXGun is a leader company in Flight Controller development and manufacture established in 2009 and formally registered in 2015. TopXGun products are widely used for agricultural purposes, security, delivery, survey, and mapping.

T1 Flight Controller

Especially for short-range activities, it's considered one of the best alternatives in agricultural application. T1 Flight Controller (Fig. 1.20) is suitable for multi-rotor UAVs, and it supports point-to-fly mode and waypoint planning (up to 128 rout points). A failsafe function is implemented, so the aircraft will return to the take-off point in case of signal loss. T1 Flight Controller supports quadcopter, hexacopter, and octocopter configuration [29].

M2 Flight Controller

M2 (Fig. 1.21) is a professional Flight Controller specially designed for the application of the UAV industry. It adopts TopXGun hardware architecture of second-generation: accuracy of the gyroscope is 20 times higher than the one of the previous generation,

Figure 1.20: T1 flight controller

and the seismic capacity is increased by 10 times. IMU has a full temperature range calibration, combined with a new sensor integration algorithm, and the accuracy of attitude measurement can reach within 0.5 degrees. Thanks to its open Software Development Kit (SDK), customers can customize the drone's functions according to their needs. The M2 flight controller can communicate with Apollo onboard computer, which is designed to fulfill more complicated tasks, as recognizing special shapes, objects, or colors. M2 and Apollo are widely used in aerial surveying and mapping, pipe network inspection, security police, and logistics and transportation industries [29].

Figure 1.21: M2 flight controller

1.3.5 DJI

DJI is a leader company in drone sector. They are involved in drone and UAVs' component production for several applications, from hobby and professional aerial photography to agricultural and business services.

N3 Flight Controller

N3 Flight Controller (Fig. 1.22) supports quadcopter, hexacopter and octocopter configurations. It provides a dual IMU redundancy and its flight control algorithm can detect IMU failures during flight. N3 is equipped with a vibration dampening system, providing better flight performances and reliability. Its robust flight control algorithm is ideal for controlling a broad range of industrial and DIY multirotor aircraft, providing professional stability [30].

Figure 1.22: N3 flight controller

A3 Flight Controller

A3 (Fig. 1.23) provides Software Development Kit (SDK) and dedicated hardware interfaces that allow customers to add actuators and sensors according to their needs. A3 supports D-RTK GNSS for centimeter-level positioning accuracy compared to normal GPS and barometer solutions. Using dual antennas, its heading reference is more accurate than a compass sensor and it can withstand magnetic interference from metal structures. For redundancy, it is equipped with three IMUs and three GNSS units, with additional analytical redundancy and advanced diagnostic algorithms. Thanks to its accurate flight control and optimal flight performances, it is ideal for industrial and cinematic applications [30].

Figure 1.23: A3 flight controller

1.4 Focus and scope of this work

This thesis aims at developing a Flight Controller for a quadcopter UAV. Some of the most critical aspects about the development of a Flight Controller are discussed: data filtering from sensors and sensor fusion are performed to obtain accurate information about quadcopter attitude, and particular attention is given to the software's working frequency. For this purpose, the STM-32 data-sheet is analyzed in detail, and signal frequencies are carefully monitored. Indeed, control algorithms could be inefficient if the software frequency is not high enough to ensure a fast response of motors to disturbances or commands. The same happens if delays due to different frequencies between software and signals occur.
A simulation model of the quadcopter is also developed to preliminary tune controller parameters before testing the Flight Controller on the drone. Quadcopter dynamics, DC motors, and PWM logic are implemented in a Simulink environment, and a linearized model is obtained to set controller parameters.
Additional tuning of controller parameters is performed on a test bench where only rotational dynamics is allowed to the drone. This phase of the work is very useful but can not give reliable results about drone response, because of the asymmetries and frictions introduced by the joint.
To the best of the author's knowledge, most commercial Flight Controllers implement a PID controller, and actually with good results. However, since the growing applications drones are involved in, best-performing controller algorithms are required to increase flight performances and improve drones' reliability. PID controller can not ensure good flight performances for the whole flight envelope, and tuning based on experimental tests can take a long time and can be dangerous for the drone. At the same time, using a PID controller, some issues related to wind-up and noises can occur. Therefore, these issues have to be solved to obtain suitable performances.
In this thesis, a fully accessible Flight Controller is developed using Arduino IDE, which is very simple to use for everyone who has some skills in programming. Code is intentionally made as simple as possible to be easily understandable for the ones who want to test new control algorithms. At the same time, an LQR controller is implemented and it is found to be a valid alternative to the more common PID controller with some advantages: with respect to the PID, an LQR does not introduce wind-up and noises issues and it can be tuned faster than a PID if a linearized model of the drone is available.
Results shown in chapter 6 demonstrate satisfactory performances of the developed Flight Controller. A 400 Hz software refresh frequency seems to be satisfactory to perform the auto-stabilizing, and follow commands. LQR is preliminarily tuned on the simulation model and its parameters are adjusted on the test bench. During flight tests, another tune is performed to obtain a fast response to the commands from the transmitter.

1.5 Flight Controller software architecture

In this section, the software architecture of the developed Flight Controller is introduced. The Arduino STM-32 is chosen as the micro-controller and it is programmed by using Arduino IDE.

The software architecture can be divided into two parts: set-up configurations and main loop operations. Arduino IDE proposes two different sections whenever a new file is opened, and they are *setup* and *loop* sections. Code written in the *setup* section is executed only when the micro-controller turns on, while the code in the *loop* is executed after the *setup* code over and over again. Variables are declared before the *setup* section.

The architecture of the developed software is shown in Fig. 1.24. The set-up section consists of PINs' configuration, timers setting, and all operations that can be executed only one time during the process (gyroscope calibration, controller settings, SD card configuration, and so on). On the other hand, the loop section executes all operations that are needed to perform flight and data logging.

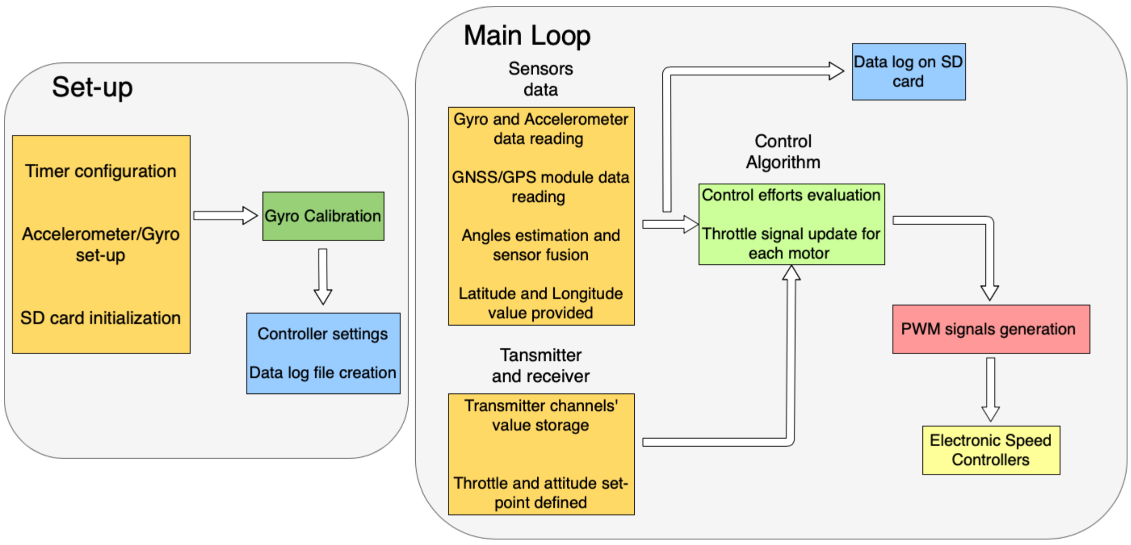

Set-up
Timer configuration
Accelerometer/Gyro set-up
SD card initialization
Gyro Calibration
Controller settings
Data log file creation
Main Loop
Sensors data
Gyro and Accelerometer data reading
GNSS/GPS module data reading
Angles estimation and sensor fusion
Latitude and Longitude value provided
Tansmitter and receiver
Transmitter channels' value storage
Throttle and attitude set-point defined
Data log on SD card
Control Algorithm
Control efforts evaluation
Throttle signal update for each motor
PWM signals generation
Electronic Speed Controllers

Figure 1.24: Flight Controller architecture

1.5.1 Code structure

The developed software mainly consists of a *main* file where functions collected in other schedules are called. Such a configuration is preferred to make the *main* as clean as possible for debugging.
The *main* file is named *FC_main*, and it is necessary to load this file on the micro-controller to execute the whole code. In *FC_main* all variables are declared and set-up configurations are executed. The Fig. 1.25 shows the code architecture and each

functions called from *FC_main*. Below, each schedule is introduced and the task of every function is explained.

- Controller
  1. *PID_calc*: evaluates control efforts according to the PID control algorithm
  2. *LQR_calc*: evaluates control efforts according to the LQR control algorithm
- IMU
  1. *gyro_setup*: starts the communication between STM-32 and gyroscope/accelerometer sensor.
  2. *calibrate_gyro*: evaluates the value to subtract to raw gyroscope data before integrating angular rate.
  3. *gyro_signalem*: accesses gyroscope and accelerometer registers to read data and store them into variables.
  4. *angle_calc* performs angle estimation from gyroscope and accelerometer data and filters them to avoid noises and drifts.
- Print
  1. *SD_print*: prints data collected from sensors (attitude, position) during flight on the file "DATA.TXT" on an SD card.
  2. *Serial_print*: prints variables for debugging if the micro-controller is connected by serial.
- Reveiver
  1. *handler_channel_*1: measures the pulse width from the transmitter and assigns it to some variables managing PPM signals.
- Timer
  1. *timer_setup*: TIM2 and TIM4 are enabled to receive signals from the transmitter and generate PWM signals.

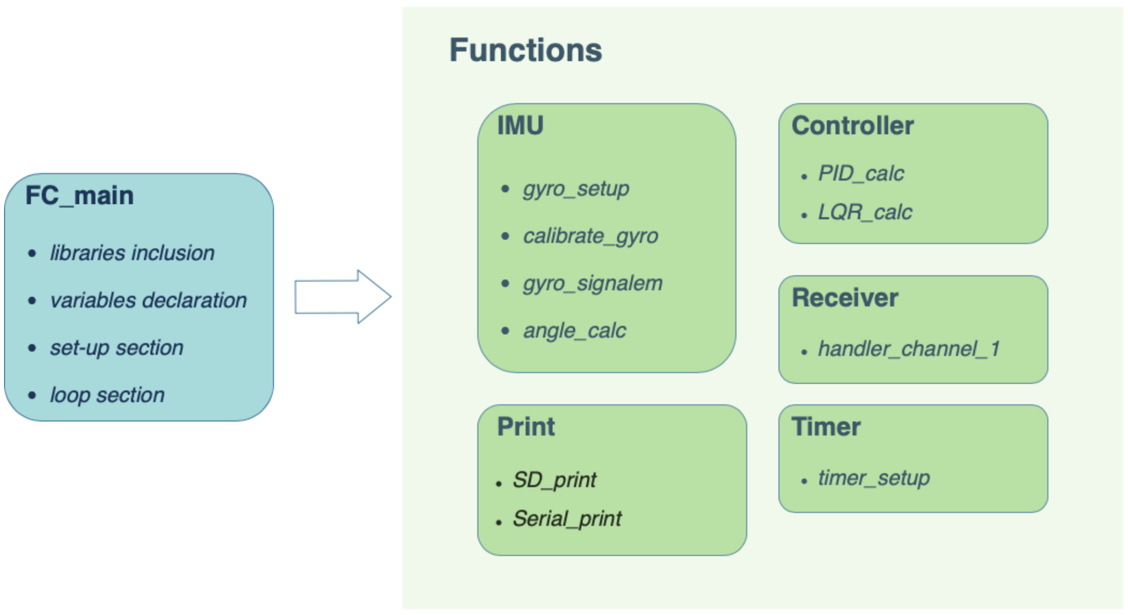


Figure 1.25: Code general structure

Chapter 2

IMU implementation

An Inertial Measurement Unit (IMU) is an electronic device that provides the angular orientation of a body with respect to an inertial reference frame. IMU typically consists of a gyroscope, an accelerometer, and a magnetometer sensor whose data are merged by the software to obtain accurate values about the body's attitude. To develop a Flight Controller, it's necessary to provide the microcontroller with the vehicle's orientation, which is used from the controller algorithm to follow an attitude signal. In the following section, an IMU is developed using a gyroscope and an accelerometer. Then, data are combined to reduce noises and a sensor fusion is performed to have accurate information about the vehicle's attitude.

2.1 Gyroscope

A gyroscope is a sensor that provides angular rates of a frame with respect to the body reference frame. Knowing how much time is passed from one measurement to another, it is possible to integrate the angular rate provided by the gyroscope, finding the body's attitude.

In this work, an MPU-6050 (Fig. 2.1) is used to provide the Flight Controller with the angular rates of the quadcopter. MPU-6050 is a Micro-Electro-Mechanical System (MEMS) that consists of a 3-axis accelerometer and a 3-axis gyroscope. In section 6.1.2, more information about this sensor is provided.

The gyroscope output is a 16 bit signed value and it is converted in to the code in a number from -500 degrees per second (dps) to 500 dps. It means that the maximum angular rate provided by the gyroscope will be 500 dps. According to needs and uses, it is possible to modify this scale directly acting on the code. However, a 500 dps full scale is more than enough considering quadcopter dynamics. Looking at the MPU-6050 datasheet Fig. 2.2 [31], it is found that to a 500dps full scale corresponds an output of 65.5 when the gyroscope is rotating at 1 dps. So it becomes easy to find an angular rate from the gyroscope in degrees per second.
When data from the gyroscope are collected and graphed, it is found out that gyroscope output is not zero, despite it is not moving. The gyroscope needs to be calibrated, and

in the developed software it is done by collecting 2000 measures and subtracting the mean value to the gyroscope outputs (Algorithm 1). The gyroscope calibration starts every time the code runs on the micro-controller, and it lasts a couple of seconds. During calibration, the gyroscope should not be moved to avoid affecting the mean value measured during the process. When the calibration process is finished, the gyroscope output is very close to zero and the gyroscope can provide accurate angular rates.
To obtain the quadcopter's attitude from gyroscope data, angular rates need to be integrated by the software. This is done by adding the traveled angle to the actual one during the time passed from one lecture to another. Since the refresh frequency of the developed software is equal to 400 Hz, the gyroscope provides data every 2500 us. It means that the traveled angle in degree can be obtained by the following relation:

$$traveled\_angle = \frac{gyro\_output}{refresh\_frequency * 65.5} \tag{2.1}$$

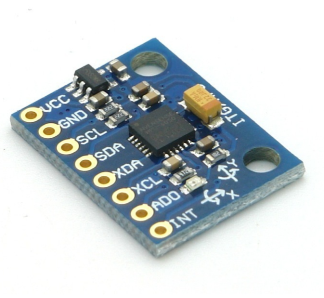
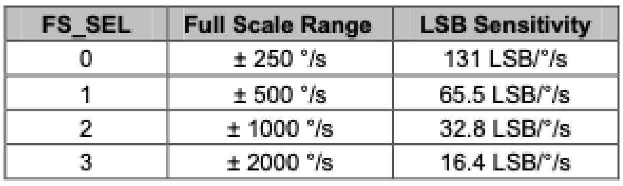


Figure 2.1: MPU-6050 gyroscope/accelerometer MEMS



FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

Figure 2.2: MPU-6050 datasheet to read gyroscope full scale range

Algorithm 1 Gyroscope Calibration

```
1: cal_int = 0;
2: if (cal_int != 2000) then
3:     for (cal_int = 0; cal_int < 2000; cal_int ++) do
4:         gyro_signalen();
5:         $\omega_{xcal}$ += $\omega_x$;
6:         $\omega_{ycal}$ += $\omega_y$;
7:         $\omega_{zcal}$ += $\omega_z$
8:     end for
9:     $\omega_{xcal}$ /= 2000;
10:    $\omega_{ycal}$ /= 2000;
11:    $\omega_{zcal}$ /= 2000;
12: end if
```

The traveled angle is added to the previous angle evaluated by the code, and in this way roll, pitch, and yaw angles of the drone can be obtained.

2.2 Accelerometer

An accelerometer provides the frame's acceleration along the three body axes. In particular, an accelerometer is typically used to measure gravity acceleration and its components along the three body axes. Knowing gravity accelerometer components, it is possible to find the body's attitude by trigonometry. Considering Fig. 2.3, the following relationships between roll (ϕ), pitch (θ) angle, and gravity acceleration can be found:

$$\begin{cases} \phi = arcsin(\frac{g_y}{g}) \\ \theta = arcsin(\frac{g_x}{g}) \end{cases} \tag{2.2}$$

where g_x and g_y are the gravity acceleration components along X_b and Y_b, and $g = 9.81$ is the module of gravity acceleration vector.

Looking at MPU 6050 datasheet (Fig. 2.4), it is possible to set the accelerometer scale just like the gyroscope. In the developed Flight Controller, it is chosen a scale from -8 g to 8 g and, according to MPU-6050 manual [31], an output of 4096 from the accelerometer will correspond to 1 g. Using Eq. 2.2, roll and pitch angles of the quadcopter can be obtained (Algorithm 3). Since gravity acceleration components don't change when the drone spins around itself, the yaw angle can not be evaluated by the accelerometer. Actually, a magnetometer should be used to provide the yaw angle and combine this information with the one provided by the gyroscope.

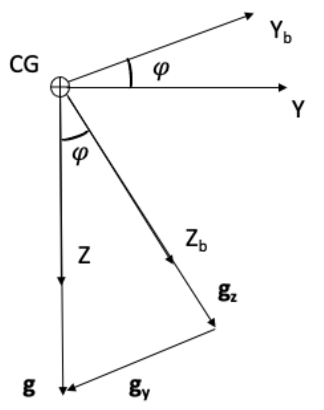
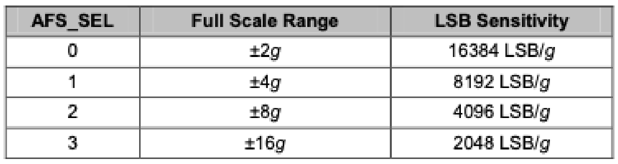


Figure 2.3: Roll angle found by trigonometry



AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

Figure 2.4: MPU-6050 datasheet to read accelerometer full scale range

Algorithm 2 Gyroscope and accelerometer data storage

```
1: HWire.beginTransmission(gyro_address);
2: HWire.write(0x3B);
3: HWire.endTransmission();
4: HWire.requestFrom(gyro_address, 14);
5: $a_x$ = HWire.read() « 8 | HWire.read();
6: $a_y$ = HWire.read() « 8 | HWire.read();
7: $a_z$ = HWire.read() « 8 | HWire.read();
8: $\omega_x$ = HWire.read() « 8 | HWire.read();
9: $\omega_y$ = HWire.read() « 8 | HWire.read();
10: $\omega_z$ = HWire.read() « 8 | HWire.read();
11: $\omega_y$ *= -1;
12: $\omega_z$ *= -1;
```

Algorithm 3 Roll and pitch angle evaluated from accelerometer data

```
1: $a = \sqrt{a_x^2 + a_y^2 + a_z^2}$;
2: $\phi_{acc} = asin\left(\frac{a_y}{a}\right) 57.296$;
3: $\theta_{acc} = asin\left(\frac{a_x}{a}\right) 57.296$;
4: if (!first_angle) then
5:     $\theta = \theta_{acc}$;
6:     $\phi = \phi_{acc}$;
7:     first_angle = true;
8: end if
```

2.3 Complementary filter

Through the gyroscope and the accelerometer, two measurements of roll and pitch angles are available. Actually, none of them is enough accurate to provide the Flight Controller with the quadcopter's attitude, because of several errors introduced by both sensors.
Angles calculated by integrating gyroscope signals are very useful to know how attitude is changing over time, but they do not provide accurate information when the body is not moving. This is because the software integrates angular rate, and sometimes measures can drift from the true angle due to noises.
On the other hand, the accelerometer is very sensitive to vibrations, so angles evaluated from Eq. 2.2 can be unreliable, especially considering the number of vibrations on a quadcopter.

To overcome these issues, a complementary filter is used to combine the two measurements obtained from the gyroscope and the accelerometer. A complementary filter corrects angles estimated from the gyroscope providing angles estimated by the accelerometer when the body is not moving. On the other hand, gyroscope angles are used to estimate angles in the short term. This is because angles from the gyroscope are less noised thanks to the integration. In practical terms, a complementary filter acts as a Low Pass Filter (LPF) for angles obtained from the accelerometer and as a High Pass Filter (HPF) for angles obtained from the gyroscope. A LPF (Eq. 2.3) and HPF (Eq. 2.4) first-order transfer function, can be written as follows:

$$HPF = \frac{1}{\tau s + 1} \tag{2.3}$$

$$LPF = \frac{\tau s}{\tau s + 1} \tag{2.4}$$

where s is the complex variable from the Laplace transformation, and tau is the $cut -$

off frequency. In Fig. 2.5 and 2.6, the bode diagrams of the magnitude of a first-order LPF and HPF with $\tau = 1$ are shown. Looking at the LPF bode diagram, before a cut-off frequency, the transfer function's magnitude is almost constant with respect to the frequency. When the frequency increases, and cut-off frequency τ is overcome, the transfer function's magnitude is significantly reduced. The HPF bode diagram behaves exactly in the opposite way. In the next sections, it will be shown how to combine these two filters to perform sensor fusion between gyroscope and accelerometer data, obtaining an accurate estimation of the drone's attitude.

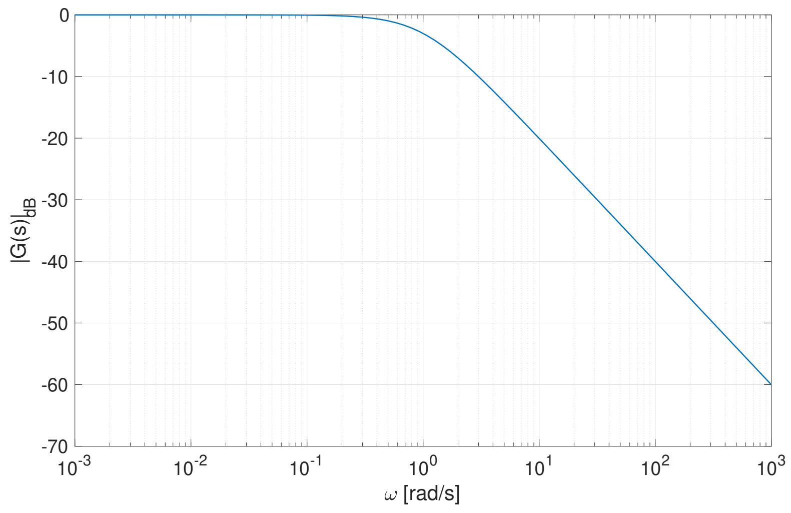


Figure 2.5: Low Pass Filter bode diagram for $\tau = 1$

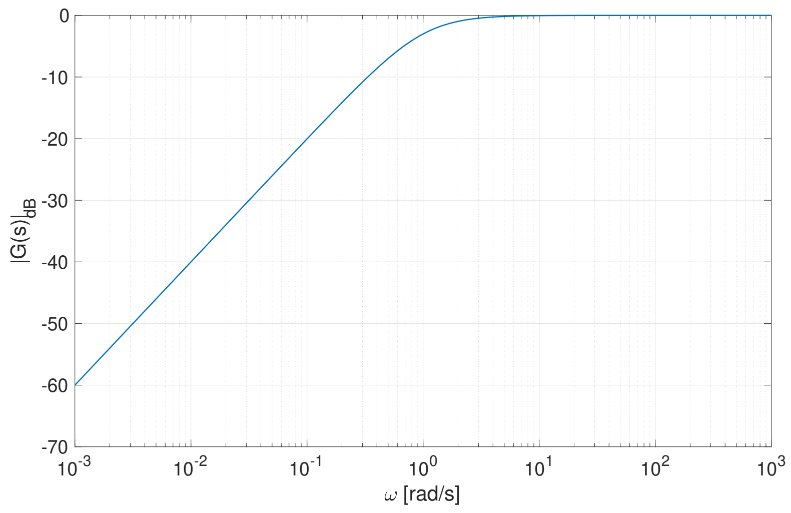


Figure 2.6: High Pass Filter bode diagram for $\tau = 1$

2.3.1 Complementary filter implementation: continuous and discrete form

As shown in Fig. 2.7, the angular rates from the gyroscope are integrated and then a High Pass Filter (HPF) is used. On the contrary, accelerometer angles are filtered by a Low Pass Filter (LPF). Defining:

$$G(s) = \frac{1}{\tau s + 1} \tag{2.5}$$

the first order LPF for acceleration angles, and

$$1 - G(s) = \frac{\tau s}{\tau s + 1} \tag{2.6}$$

the first-order HPF for gyroscope angles, it is possible to reduce the block diagram to Fig. 2.8. By adjusting the τ value, the filter can be tuned properly.
 Since the continuous form of a complementary filter can not be implemented on a

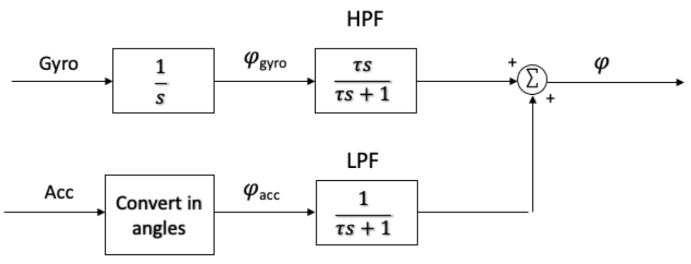


Figure 2.7: Block diagram continuous implementation of a Complementary Filter for roll angle

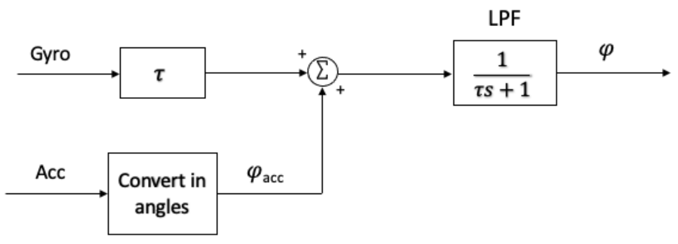


Figure 2.8: Reduced block diagram of a Complementary Filter for roll angle

microcontroller, the discrete form is introduced in the block diagram of Fig. 2.9. A fixed fraction of one information and the complementary fraction of the other are combined to obtain the estimation of the angle. Thus, this value is feedback at the next time step. Looking at Fig. 2.9, $K \in \mathbb{R}$ and $0 \leq \mathrm{K} \geq 1$ is the fixed fraction, Δt is the elapsed time and $\frac{1}{z}$ block indicates that the output variable of the block is the input of the previous time step.

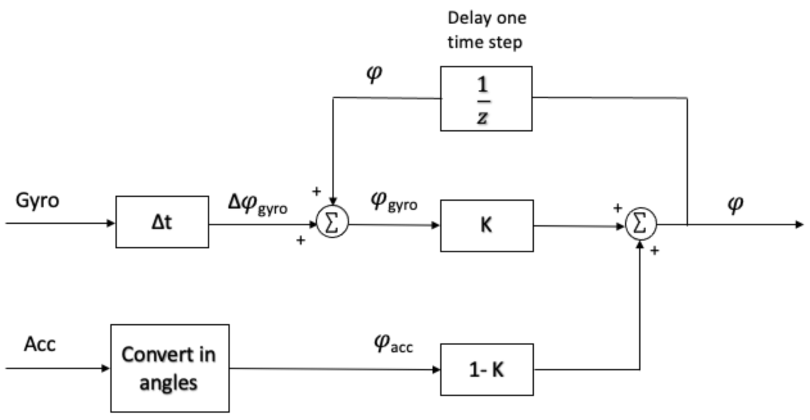


Figure 2.9: Block diagram of a discrete Complementary Filter for roll angle

2.3.2 Results

In Fig. 2.10 and Fig. 2.11, roll and pitch angles estimated by the gyroscope, the accelerometer, and the complementary filter are shown and compared. Angles estimated by the gyroscope are far away from the true angle of the drone but they are less affected by noises if compared to accelerometer angles. Angles estimated by the accelerometer seem to be closer to the true angles, but they are affected by a great number of noises. Looking at angles estimated by the complementary filter, pitch and roll angles estimation are combined and an accurate IMU is obtained.

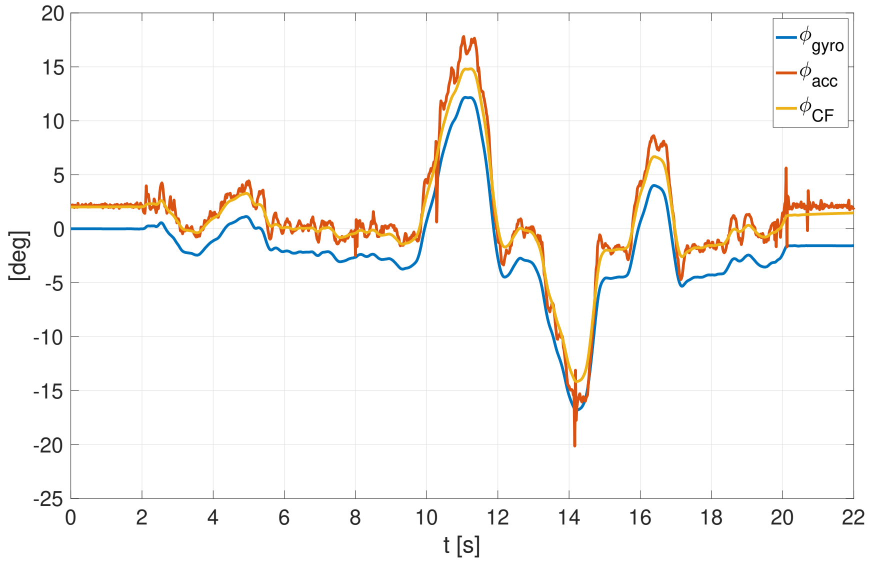


Figure 2.10: Comparison between roll angle estimated by the gyroscope, the accelerometer and by the complementary filter

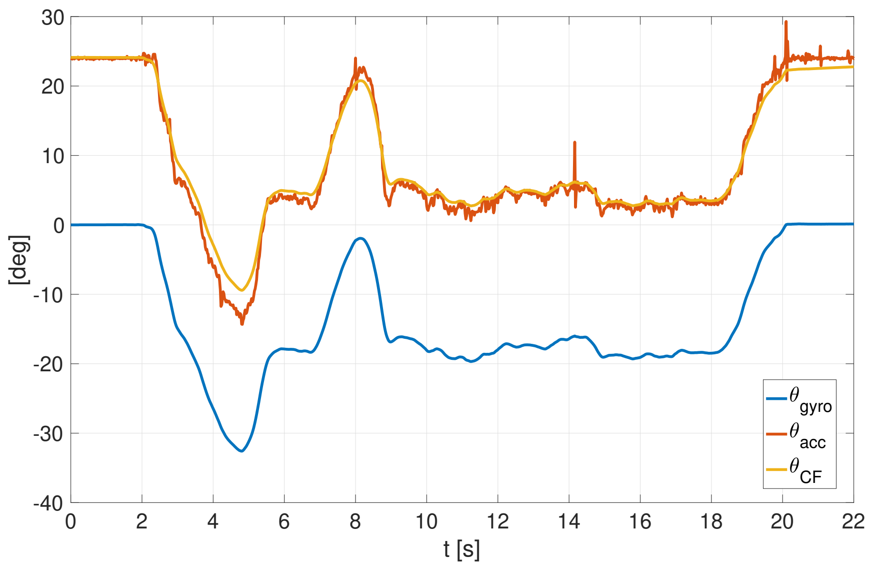


Figure 2.11: Comparison between pitch angle estimated by the gyroscope, the accelerometer and by the complementary filter

Chapter 3

Transmitter and Receiver

A transmitter and a receiver are used to send signals to the Flight Controller. In this section, Pulse Width Modulated (PWM) and Pulse Position Modulated signals are introduced, and the STM-32 board is enabled to manage signals from the transmitter by communicating with the receiver. STM-32 data-sheet is used to obtain information about how to manage PWM signals properly and obtain desired PWM frequency directly acting on the code.

3.1 Pulse Width Modulation

The pulse width modulation is a modulation technique that generates variable-width pulses to represent the amplitude of an analog input signal [32]. According to the PWM, the voltage is distributed through a series of pulses rather than a continuously varying signal. Due to the duty cycle and high frequency of the signal, the voltage sent to actuators is the average voltage of the signal. By increasing or decreasing the pulse width, the average voltage can be increased and decreased as well (Fig. 3.1).

A PWM signal can be generated using a comparator as shown in Fig. 3.2. A modulating signal forms one of the inputs to the comparator, while the other input is fed with a non-sinusoidal wave (or sawtooth wave), which operates at the carrier frequency. The comparator compares the two signals and generates a PWM signal as its output waveform. If the value of the Sawtooth triangle signal is more than the modulation signal, then the PWM output signal is in the "High" state, else it's in the "Low" state. Thus, the value of the input signal magnitude determines the comparator output which defines the width of the pulse generated at the output [33].

In this thesis, the width of the pulse is sent from a transmitter to a receiver, which is connected to the STM-32 board PINs. Thus, the signal can be stored in some variables and elaborated by the software which sends it to the motors through other board's PINs. Therefore, a PWM requires as many board PINs as any channel of the transmitter. It would result in at least four wirings (throttle, roll, pitch, and yaw signals) and four board's PINs employed, which are too much if it's needed to add some hardware components to the Flight Controller. To overcome this issue, another modulation

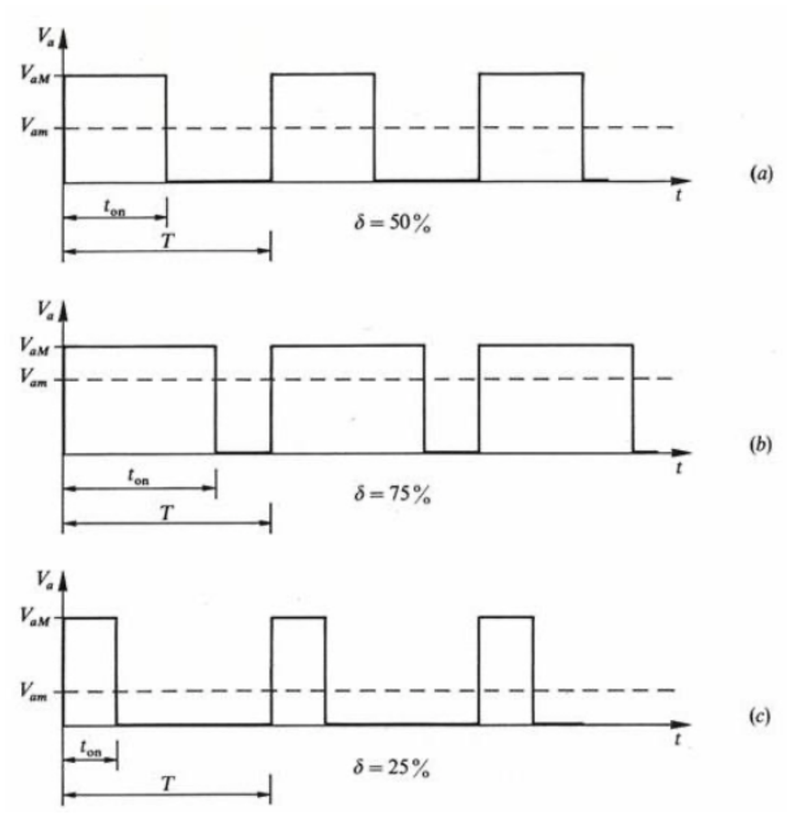


Figure 3.1: Duty cicle of a PWM signal with respect to the signal amplitude

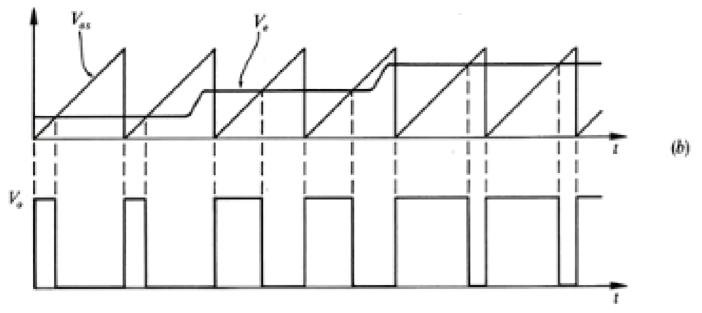


Figure 3.2: PWM generation: comparison of the Sawtooth triangle signal with the reference signal

technique based on PWM is introduced and implemented on the developed software.

3.1.1 Pulse Position Modulation

The Pulse Position Modulation (PPM) is a modulation technique where the position of the pulsed carrier is varied according to the modulating signal. Actually, PPM is based

on PWM: the position of the pulse in PPM signals depends on the trailing edge of the pulses in PWM signals. As it is shown in Fig. 3.3 the length of the pulse is always the same, but the position changes according to the width of the PWM signal.
By switching the output of the transmitter from PWM to PPM signal, such a modulation requires only one PWM signal from the board to manage all channels of the transmitter. This means that only one PIN of the board is required, with the advance of reducing wirings.

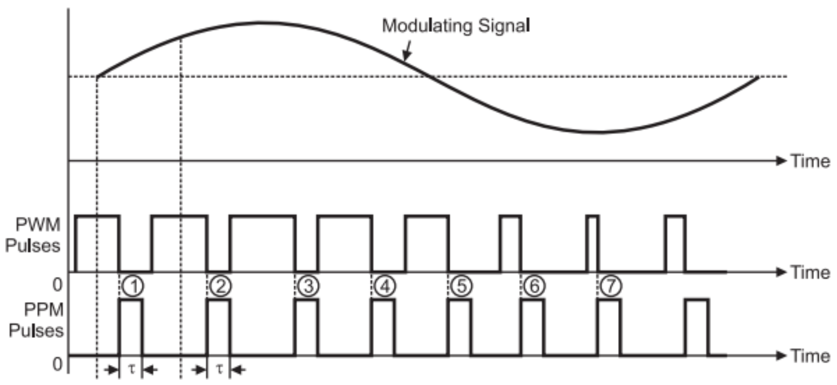


Figure 3.3: PPM signal from the trailing edge of PWM pulses

3.2 Input capture mode

In the previous section, it is stated that a PWM signal needs to be generated from the STM-32 board, and changing the pulse length of the signal by the transmitter or the software itself, quadcopter's motors can be controlled. Moreover, there are many requirements that the PWM signal generated by the STM-32 PINs needs to satisfy. Therefore, in this section, the STM-32 data-sheet is analyzed in detail to find the more efficient way to generate a PWM from the STM-32 PINs.

3.2.1 Interrupt

To generate a PWM signal, a microcontroller has to calculate the pulse length coming from the transmitter. When the rising edge of the pulse is detected, the microcontroller measures the start time. Then, when the falling edge is detected, the microcontroller measures the end time and calculates the time of the pulse. Therefore, the microcontroller needs to interrupt the currently executing code to deal with the calculation of the pulse length. Such a response of the processor is called *interrupt*, and it occurs every time a command is sent from the transmitter to the microcontroller. In Fig. 3.4 an example about an *interrupt* is shown: while the code is calculating PID inputs, an interrupt occurs and the start time of a pulse is measured. After that, the software continues to run the main code and, when the falling edge of the pulse is detected,

another interrupt occurs to calculate the pulse length.
The microcontroller communicates with the transmitter through some PINs connected to the receiver. These PINs have to be enabled to manage with PWM signals and generate an interrupt when the falling or the rising edge of a signal is detected. Thus, the STM-32 datasheet is studied to configure properly these PINs.

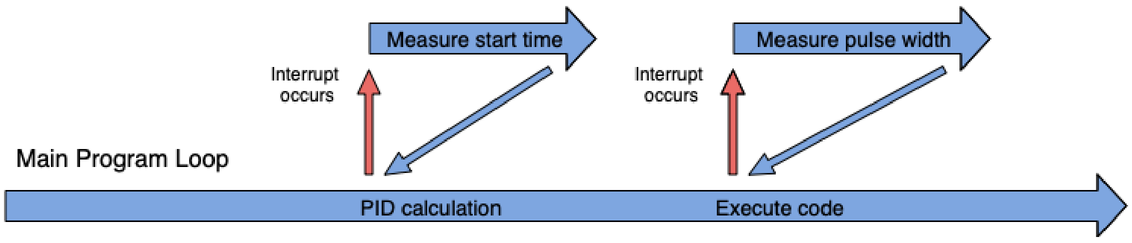


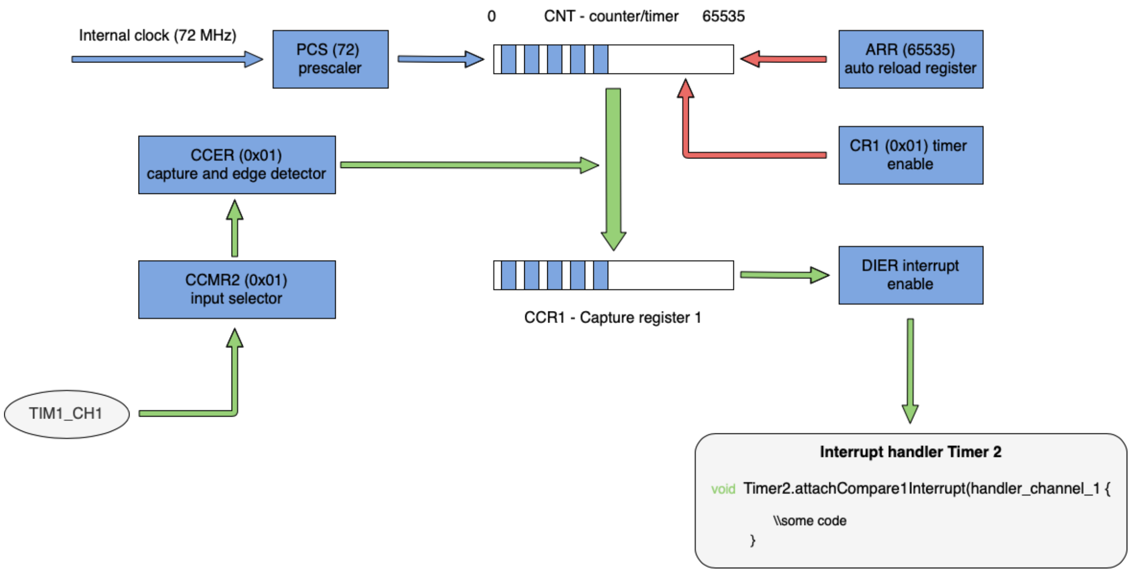
Figure 3.4: An example of how interrupt occurs

3.2.2 STM-32 timers

The STM-32 datasheet introduces four timers (TIM2, TIM3, TIM4, TIM5) which consist of a 16-bit auto-reload counter driven by a programmable pre-scaler. They may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture mode) [34]. A block diagram is purposed by the data-sheet to explain how these timers work. In Fig. 3.5, the working principle it is resumed in a reduced block diagram. A 16-bit timer (CNT) that counts from 0 to the value of an auto-reload register (ARR), fixed at 65535 (the maximum value of a 16 bit counter). When a signal from the connected input (TIM2_CH1) changes from low to high, the rising edge is detected and the timer value of the counter (CNT) is stored in the capture register (CCR1). When this happens, an interrupts is generated and some code can be executed to store the rising edge time of the pulse. This is enough to generate a PPM signal, as explained in section 3.1.1.

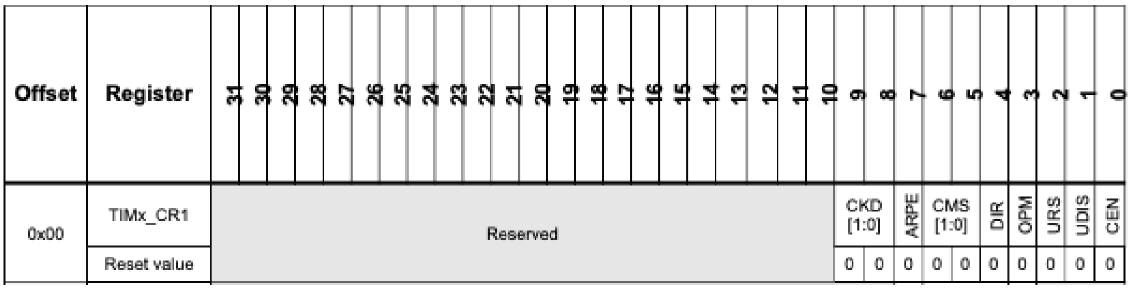
3.2.3 Code implementation

Following the example proposed in the datasheet, the TIM2 of the STM-32 is configured. The STM-32 timers consist of some registers that are pre-defined 32-bit variables fixed in the memory of the board. By setting the individual bits of a register, the specific function of the timer can be set. Fig. 3.6 shows the register map for a STM-32 timer. It should be noted that some bits of the register are reserved, and it is not allowed to access them. However, the bits of the register that have to be set for input capture mode are accessible.


Internal clock (72 MHz)
PCS (72) prescaler
0
CNT - counter/timer
65535
ARR (65535) auto reload register
CR1 (0x01) timer enable
CCER (0x01) capture and edge detector
CCMR2 (0x01) input selector
TIM1_CH1
CCR1 - Capture register 1
DIER interrupt enable
Interrupt handler Timer 2

```
void  Timer2.attachCompare1Interrupt(handler_channel_1 {
          \\some code
      }
```

Figure 3.5: Input Capture block diagram



Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	TIMx_CR1	Reserved																						CKD [1:0]		ARPE	CMS [1:0]		DIR	OPM	URS	UDIS	CEN
	Reset value																							0	0	0	0	0	0	0	0	0	0

Figure 3.6: Register map of a STM-32 timer

In the code below (Algorithm 4), some bits of the TIM2 registers are set to enable a specific function and perform input capture mode properly:

- $Timer2.attachCompare1Interrupt(handler\_channel\_1)$: every time an interrupt occurs, the sub-rutine $handler\_channel\_1$ is executed

- $TIMER2\_BASE- > CR1 = TIMER\_CR1\_CEN$: enables the timer counter

- $TIMER2\_BASE- > DIER = TIMER\_DIER\_CC1IE$: enables the timer to trigger an interrupt when a capture occurs

- $TIMER2\_BASE- > CCMR1 = TIMER\_CCMR1\_CC1S\_INPUT\_TI1$: connect the input to the edge detector

- $TIMER2\_BASE- > CCER = TIMER\_CCER\_CC1E$: enable the trigger on the rising edge

- $TIMER2\_BASE- > PSC = 71$: the pre-scalar is set in the corresponding register

- $TIMER2\_BASE- > ARR = 0xFFFF$: the auto-reload register is set in the corresponding register

The other registers are unused and they are set back to 0.

Algorithm 4 TIM2 set-up and input capture mode configuration

```
Timer2.attachCompare1Interrupt(handler_channel_1);
TIMER2_BASE->CR1 = TIMER_CR1_CEN;
TIMER2_BASE->CR2 = 0;
TIMER2_BASE->SMCR = 0;
TIMER2_BASE->DIER = TIMER_DIER_CC1IE;
TIMER2_BASE->EGR = 0;
TIMER2_BASE->CCMR1 = TIMER_CCMR1_CC1S_INPUT_TI1;
TIMER2_BASE->CCMR2 = 0;
TIMER2_BASE->CCER = TIMER_CCER_CC1E;
TIMER2_BASE->PSC = 71;
TIMER2_BASE->ARR = 0xFFFF;
TIMER2_BASE->DCR = 0;
```

3.3 Electronic Speed Controller

An electronic speed controller (ESC) (Fig. 3.7) is an electronic circuit that controls and regulates the speed of an electric motor. ESCs contain a microcontroller that interprets the input signal and appropriately controls the motor using firmware. An ESC receives as input a signal, while the circuit and the firmware provide to send it to motors and to control them.

In the previous section 3.2, the pulse length from a transmitter is measured but a PWM signal should be also generated by the PINs the ESCs are connected to. STM-32 timers introduced in section 3.2.2 are used again and the PWM signal frequency is adjusted from the software to equal the one of the code.

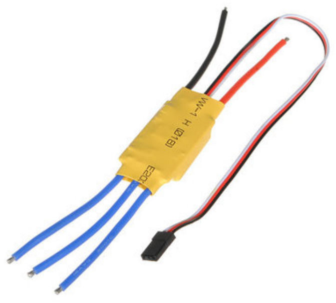


Figure 3.7: Electronic Speed Controller

3.3.1 PWM generation

To describe how timers are used to generate a PWM signal, in Fig. 3.2 a block diagram is shown. The working principle is very similar to the one explained in Fig. 3.5. Referring to Fig. 3.8, a counter timer (CNT) counts from 0 to the value of the auto-reload register (ARR), fixed to 5000 in this case. The channel 1 (TIM4_CH1) is set high whenever the timer value is less than the compare register (CCR1) value. When the timer is greater than the compare register (CCR1) the channel 1 (TIM4_CH1) output is set low. Because the (ARR) register is set to 5000, the pulse frequency will be 200 Hz.
By reloading the counter, it is possible to adjust the PWM frequency according to needs. In this work, a 400 Hz refresh frequency for the software is desired to obtain good flight performance. It means that the developed Flight Controller will calculate controller effort every 2.5 ms, providing a fast response to disturbances and commands as well. The software will provide to reload the auto-reload register (ARR) manually every 2.5 ms through the lines in Algorithm 5. When the first loop starts, the current time is stored in *loop_timer* variable. After assigning the pulse length to each channel of the TIM4, the ARR is set to 5000 and the counter is reloaded. esc_1, esc_2, esc_3, and esc_4 are the variables where the pulse length for each motors (or ESC) is stored in. Those variables are assigned to each channel of the TIM4 from line 1 to line 4 of Algorithm 5: channel 1, 2, 3, and 4 corresponds respectively to PINs B6, B7, B8, and B9 of the STM-32 board. When code operations end, a while loop occurs until the difference between the current time and the *loop_timer* is less the 2.5 ms. Finally, *loop_timer* variable is updated with the current time value and another loop can become. In this way, the timer counter is reloaded every 2.5 ms, and a 400 Hz PWM signal is provided.

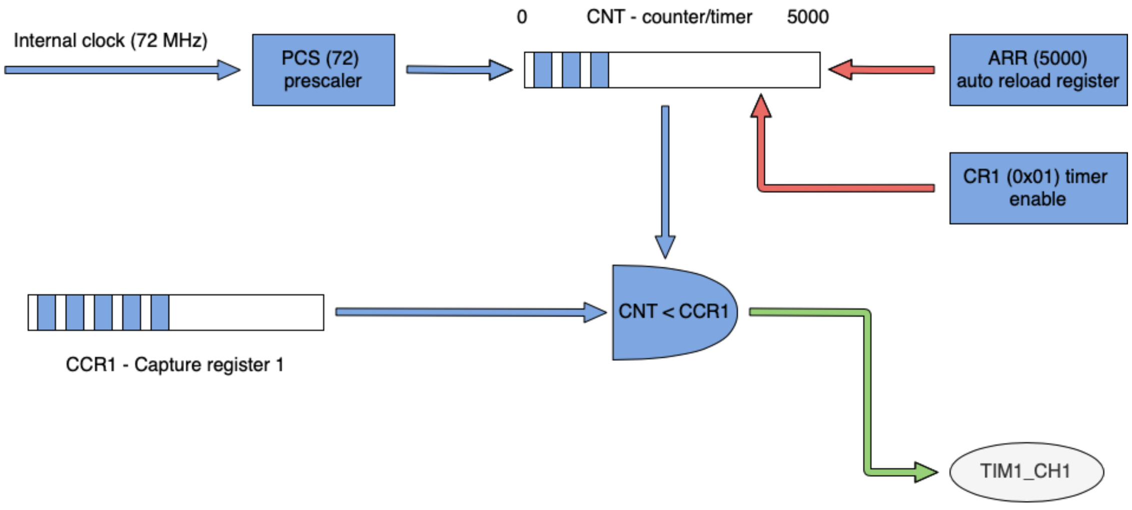


Figure 3.8: TIM4 working pinciple shown in a block diagram

Algorithm 5 Software frequency set at 400 Hz

```
TIMER4_BASE->CCR1 = esc_1;
TIMER4_BASE->CCR4 = esc_2;
TIMER4_BASE->CCR3 = esc_3;
TIMER4_BASE->CCR2 = esc_4;
TIMER4_BASE->CNT = 5000;

while (micros() - loop_timer < 2500) do loop_timer = micros();
end while
```

Chapter 4

GNSS/GPS Module

4.1 Global Navigation Satellite System

The Global Navigation Satellite System (GNSS) is a navigation system that uses satellites for geo-spatial positioning. A GNSS consists of a constellation of satellites orbiting Earth. These satellites continuously transmit signals that enable users to determine their three-dimensional position with global coverage. The working principle of the GNSS is based on an elemental geometric problem, involving the distances of a user to a set of at least four GNSS satellites with known coordinates. The distances between the GNSS receiver and satellites are determined by using signals and navigation data transmitted by the satellites. This will result in an accuracy of several meters about the position of the user. However, more advanced techniques can be used for centimeter-level positioning [35].

A GNSS consists of three main segments:

- The space segment: consists of all the satellites that provide the receiver with signals and navigation data.
- The control segment: is responsible for the operation of the system.
- The user segment: comprises the GNSS module that provides, for instance, the position and the velocity of the user.

The GNSS mainly consists of three main satellite technologies: GPS, Glonass, and Galileo. Below, these satellite constellations are described.

GPS

The Global Position System (GPS) was developed by the United States Department of Defense (DoD) for military purposes. The GPS constellation involves satellites that are positioned in six equally spaced orbital planes surrounding Earth. Each plane contains four ‘slots’, constituting a 24-slot arrangement. This ensures that there are at least four satellites in view from any point of the planet. The satellites are placed in an orbit at an altitude of 20200 Km and an inclination of 55° relative to the equator which is named Medium Earth Orbit (MEO). Orbits are nearly circular, due to their eccentricity of less than 0.02. The semi-major axis measures about 26560 Km and the nominal period is 11 hours, 58 minutes and 2 seconds [35].

Glonass

The nominal Glonass constellation consists of 24 MEO satellites. They are placed in three different orbital planes that accommodate 8 satellites equally spaced. The orbits are roughly circular, at an altitude of 19100 Km and with an inclination of about 64.8°. The nominal period is of 11 hours, 15 minutes and 44 seconds and the geometry is repeated every eight sidereal days. Due to the lack of funding, the number of satellites decreased from 24 to only 6 in 2001. Actually, in August 2001, the Russian government invested new funding to recover the constellation and to modernize the system. A total of 24 operational satellites plus 2 in maintenance were again available in December 2011, and the full constellations were restored.

Galileo

In its Full Operational Capability (FOC), the Galileo constellation consists of 27 operational and 3 spare MEO satellites at an altitude of 23222 Km and with an orbit eccentricity of 0.002. Each of the three orbital planes is occupied by three satellites, and they are inclined at an angle of 56° with respect to the equator. The satellites are spread around each plane with an orbital period of 14 hours, 4 minutes and 45 seconds, repeating the geometry every 17 revolutions (10 sidereal days). This constellation guarantees a minimum of six satellites in view from any point on Earth's surface, with an elevation above the horizon of more than 10°.

4.1.1 GNSS and drones

Since the GNSS was developed, armies have been using it to coordinate military activities. However, nowadays several civilian applications involve GNSS technology, from surveying operations to smart-phones that use it frequently. In particular, GNSS can be very useful if combined with drone technology: some of the applications mentioned in section 1.2 can't be even done without a GNSS module. In fact, the GNSS represents a way to perform the autonomous flight, which is one of the main objects to be achieved to employ drones in more and more applications.
Therefore, It appears clear the importance of providing a Flight Controller for UAVs with latitude and longitude for global positioning, and this is why the developed Flight Controller is enabled to communicate with a GNSS/GPS module.

4.2 u-blox receivers

On the market, several GNSS receivers are available. For this work, an $u-blox$ receiver is chosen. $u - blox$ receivers are fully configurable with UBX protocol configuration messages (UBX-CFG-XXX) that can be sent during normal operation to change the so-called $current\ configuration$. All documentation about how to change an $u - blox$ receiver $current\ configuration$ can be found in [36].

$u-blox$ receivers support a wide range of different GNSS, such as the ones described above (GPS, Glonass, Galileo), and some augmentation systems (SBAS, QZSS, IMES). Sending the message UBX-CFG-GNSS, the users are able to specify which signals should be processed by the receiver. The default settings provide the receiver with a GPS signal, and it is the one used in this work. Actually, as it will be seen later, GNSS module is employed to provide the Flight Controller with latitude and longitude, despite the several and useful information a GNSS module can also provide (timing, speed over the ground). However, latitude and longitude data can be enough to perform the autonomous flight.

4.2.1 Serial communication and u-blox protocols

To transmit GNSS measurements from the $u - blox$ receiver to the serial, Universal Asynchronous Receiver/Transmitter (UART) ports are featured. The serial ports consist of an RX and a TX line, which provides the serial with information about the receiver's status, positioning, timing, and more other data. The $u - center$ software is provided by $u - blox$ to allow the users to read the serial data stream by only using any serial to USB interface. Opening the $u - center$ software serial monitor, data from the communication between the receiver and satellites can be read. They are organized in data blocks that are repeated at a certain frequency using the NMEA protocol. In Fig. 4.1, the NMEA protocol is explained. More information about this protocol can be found in [36].

In Fig. 4.2 a data block printed on the $u-center$ serial monitor is shown. By default settings, $u - blox$ receivers output every data block at 1 Hz frequency. This is way too slow for a Flight Controller to perform position-hold or autonomous flight but, looking at $u - blox$ manual, it is possible to increase the update frequency up to 10 Hz by sending some messages to the serial port.
The baud rate is set to 9600 baud per second (bps), and it is not equal to the one of the Flight Controller software fixed at 57600 bps. Again, looking at $u - blox$ manual, some messages need to be sent to the serial port to change the baud rate. As regards the data stream, some lines in Fig. 4.2 are not needed to provide the flight controller with latitude and longitude. Indeed, GGA and GSA lines contain all information that are needed to evaluate the latitude and longitude of the receiver. All GSV messages depend on the number of satellites in view from the receiver, and they are not needed to the Flight Controller. They can be disabled by sending other messages to the serial port. All these operations are executed from specified code lines, and they will be explained in detail in the next section.

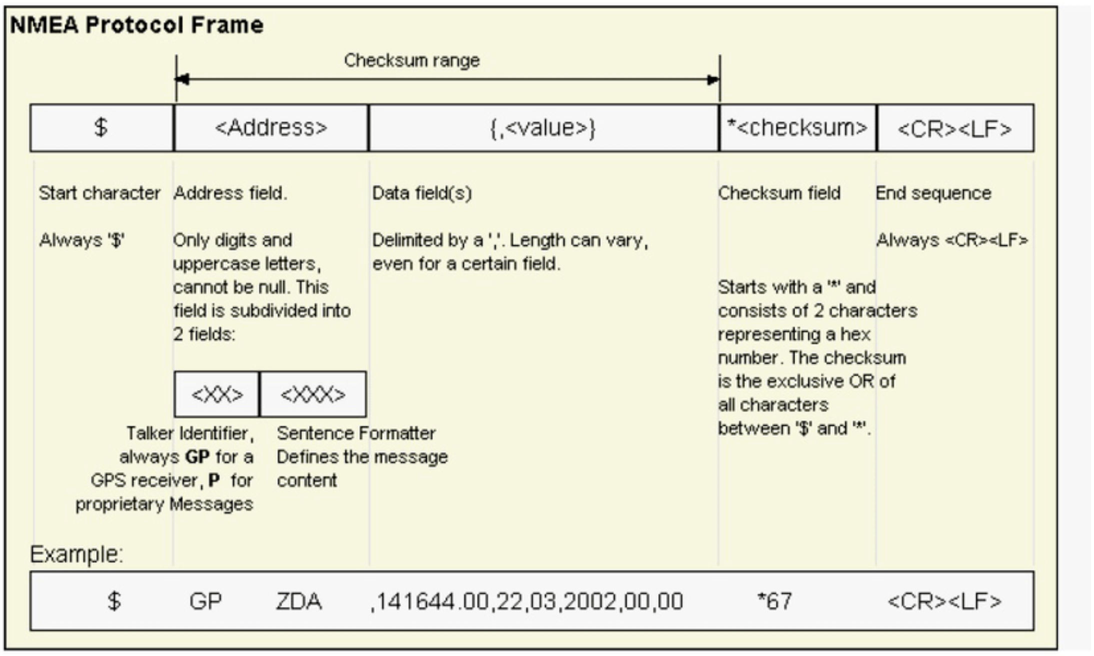
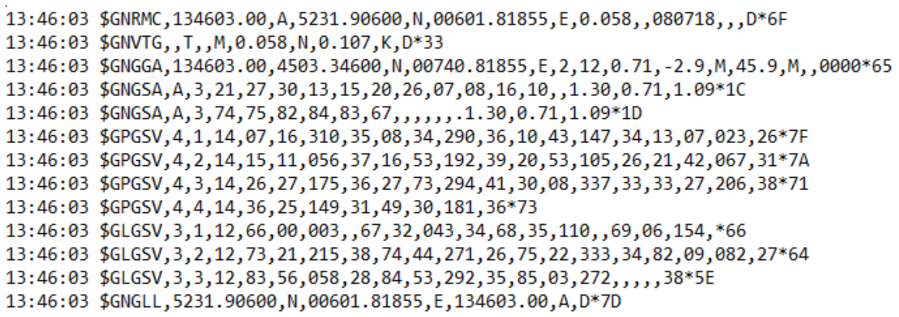

NMEA Protocol Frame
Checksum range
$
<Address>
{,<value>}
*<checksum>
<CR><LF>
Start character
Always '$'
Address field.
Only digits and uppercase letters, cannot be null. This field is subdivided into 2 fields:
<XX>
<XXX>
Talker Identifier, always **GP** for a GPS receiver, **P** for proprietary Messages
Sentence Formatter Defines the message content
Data field(s)
Delimited by a ','. Length can vary, even for a certain field.
Checksum field
Starts with a '*' and consists of 2 characters representing a hex number. The checksum is the exclusive OR of all characters between '$' and '*'.
End sequence
Always <CR><LF>
Example:
$
GP
ZDA
,141644.00,22,03,2002,00,00
*67
<CR><LF>

Figure 4.1: Structure of a NMEA protocol message used by u-blox receivers



```
13:46:03 $GNRMC,134603.00,A,5231.90600,N,00601.81855,E,0.058,,080718,,,D*6F
13:46:03 $GNVTG,,T,,M,0.058,N,0.107,K,D*33
13:46:03 $GNGGA,134603.00,4503.34600,N,00740.81855,E,2,12,0.71,-2.9,M,45.9,M,,0000*65
13:46:03 $GNGSA,A,3,21,27,30,13,15,20,26,07,08,16,10,,1.30,0.71,1.09*1C
13:46:03 $GNGSA,A,3,74,75,82,84,83,67,,,,,,.1.30,0.71,1.09*1D
13:46:03 $GPGSV,4,1,14,07,16,310,35,08,34,290,36,10,43,147,34,13,07,023,26*7F
13:46:03 $GPGSV,4,2,14,15,11,056,37,16,53,192,39,20,53,105,26,21,42,067,31*7A
13:46:03 $GPGSV,4,3,14,26,27,175,36,27,73,294,41,30,08,337,33,33,27,206,38*71
13:46:03 $GPGSV,4,4,14,36,25,149,31,49,30,181,36*73
13:46:03 $GLGSV,3,1,12,66,00,003,,67,32,043,34,68,35,110,,69,06,154,*66
13:46:03 $GLGSV,3,2,12,73,21,215,38,74,44,271,26,75,22,333,34,82,09,082,27*64
13:46:03 $GLGSV,3,3,12,83,56,058,28,84,53,292,35,85,03,272,,,,,38*5E
13:46:03 $GNGLL,5231.90600,N,00601.81855,E,134603.00,A,D*7D
```

Figure 4.2: An example of a data block output following NMEA protocol

4.3 Module implementation

In this section, the GNSS/GPS module implementation is shown. Following the documentation provided by the $u - blox$, some messages need to be sent to the serial port to change the default settings that are not consistent with the developed Flight Controller. Then longitude and latitude in degree are found reading the data block printed by the receiver on the serial monitor according to the NMEA protocol. All code lines to change default settings, and the simple algorithm used to read latitude and longitude are explained in detail, and finally, longitude and latitude values in degrees are stored in some variables.

4.3.1 GNSS module set-up and data reading

In the Algorithm 6, the *gps_setup* is shown. Its function are reading data written by the receiver on the serial port, and setting the $u - blox$ module according to Flight Controller needs. As explained above, three default settings need to be modified in reading GNSS data:

1. Disabling GSV messages that are not needed to know latitude and longitude.

2. Increase data update frequency that is default set at 1 Hz.

3. Increase serial baud rate from 9600 bps to 57600 bps.

Referring to Agorithm 6, line 4 disables GPGSV messages by writing an eleven characters string named *Disable_GPGSV* in the serial port. Line 8 set the update frequency to 5 Hz by writing a fourteen characters string named *Set_to_*$5Hz$ in the serial port. In fact, a 5 Hz frequency is supposed to be enough to perform position-holding. However, checking the $u - blox$ manual, the update frequency can be set up to 10 Hz. Finally, line 12 set the baud rate of GPS data to 57600 bps by writing the twenty-eight characters string named *Set_to_*$57kbps$.

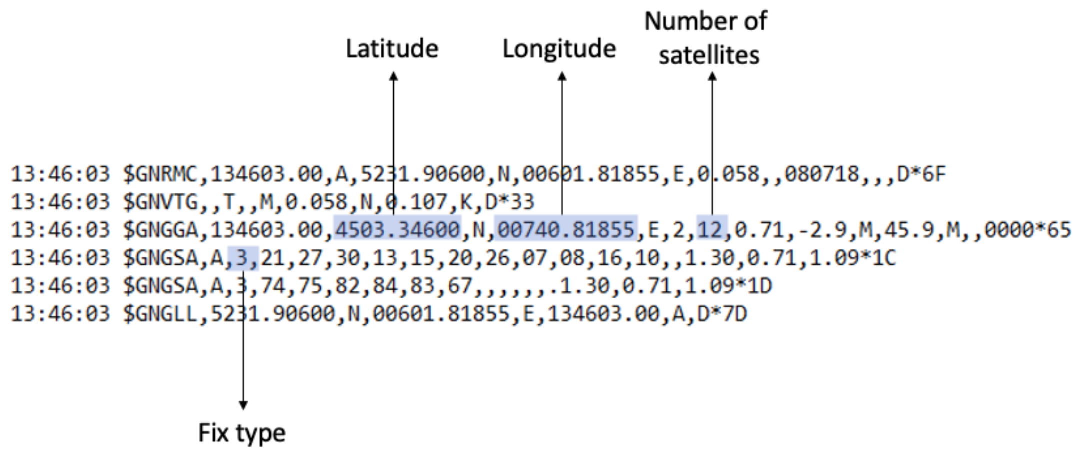
Algorithm 6 GPS data reading from Arduino STM-32

```
 1: Serial2.begin(9600);
 2: delay(250);
 3:
 4: uint8_t Disable_GPGSV[11] = 0xB5, 0x62, 0x06, 0x01, 0x03, 0x00, 0xF0, 0x03,
    0x00, 0xFD,0x15;
 5: Serial2.write(Set_to_5Hz, 14);
 6: delay(350);
 7:
 8: uint8_t Set_to_5Hz[14] = 0xB5, 0x62, 0x06, 0x08, 0x06, 0x00, 0xC8, 0x00, 0x01,
    0x00, 0x01, 0x00, 0xDE, 0x6A;
 9: Serial2.write(Set_to_5Hz, 14);
10: delay(350);
11:
12: uint8_t Set_to_57kbps[28] = 0xB5, 0x62, 0x06, 0x00, 0x14, 0x00, 0x01, 0x00,
    0x00, 0x00, 0xD0, 0x08, 0x00, 0x00,
13: 0x00, 0xE1, 0x00, 0x00, 0x07, 0x00, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE2, 0xE1
14: delay(200);
15:
16: Serial2.begin(57600);
17: delay(200);
```

4.3.2 Latitude and longitude storage in variables

As explained above, once the receiver has written GPS data according to NMEA protocol, it is the task of the software to read these data and filtering latitude and longitude information. In Fig. 4.3, an example of data block updated at 5 Hz frequency is shown. It is possible to note that every line starts with a "$" character. In Fig., four data are highlighted: the latitude, the longitude, the number of satellites used to calculate the position and the fix type for quality indicator. It should be noted, that the number of satellites in GGA messages will have a maximum value of twelve, even when the satellites that are providing signals are more than twelve.

The NMEA protocol ensures that all characters will be always in the same position. Therefore, lines can be filtered easily to access latitude and longitude. In Algorithm 7 the code implemented is shown. When a "$" character is found, a buffer named *incoming_message* is cleared from the data of the previous loop. Every character of each line written by the receiver is stored in this buffer. When a "*" is found the *new_line* variables is set to 1 and some checks are executed to filter latitude and longitude data. The first check is executed to verify if latitude and longitude information are available. Then, another check is executed to verify if the receiver is writing the GGA message which provides latitude and longitude information. The NMEA protocol provides the serial port with latitude and longitude in degree-minute format. For example, if the GPS receiver reports a latitude of 4717.112671 North, this means a latitude of 47 degrees and 17.112671 seconds. Because minutes are not easily usable for position-holding, in lines 34 and 45 the latitude and longitude are converted in degrees. After that, longitude and latitude are stored in some variables and can be used from the controller to perform position-holding.



```
13:46:03 $GNRMC,134603.00,A,5231.90600,N,00601.81855,E,0.058,,080718,,,D*6F
13:46:03 $GNVTG,,T,,M,0.058,N,0.107,K,D*33
13:46:03 $GNGGA,134603.00,4503.34600,N,00740.81855,E,2,12,0.71,-2.9,M,45.9,M,,0000*65
13:46:03 $GNGSA,A,3,21,27,30,13,15,20,26,07,08,16,10,,1.30,0.71,1.09*1C
13:46:03 $GNGSA,A,3,74,75,82,84,83,67,,,,,,.1.30,0.71,1.09*1D
13:46:03 $GNGLL,5231.90600,N,00601.81855,E,134603.00,A,D*7D
```

Figure 4.3: An example of a data block output following NMEA protocol with GPGSV lines disabled

Algorithm 7 Latitude and Longitude data storage

```
while (Serial2.available() and new_line_found == 0) do
    char read_serial_byte = Serial2.read();
    if (read_serial_byte == $) then
        for (message_counter = 0; message_counter <= 99; message_counter ++) do
            incomming_message[message_counter] = '-';
        end for
        message_counter = 0;
    else(message_counter <= 99) message_counter ++;
    end if
    incomming_message[message_counter] = read_serial_byte;
    if (read_serial_byte == '*') then new_line_found = 1;
    end if
end while

if (new_line_found == 1) then new_line_found = 0;
    if (incomming_message[4] == 'L' and incomming_message[5] == 'L' and incomming_message[7] == ',') then
        digitalWrite(STM32_board_LED, !digitalRead(STM32_board_LED));
        l_lat_gps = 0;
        l_lon_gps = 0;
        lat_gps_previous = 0;
        lon_gps_previous = 0;
        number_used_sats = 0;
        Serial.println("Errore");
    end if
end if
if (incomming_message[4] == 'G' and incomming_message[5] == 'A' and (incomming_message[44] == '1' or incomming_message[44] == '2')) then
    lat_gps_actual = (incomming_message[19] - 48)*10000000;
    lat_gps_actual += (incomming_message[20] - 48)*1000000;
    lat_gps_actual += (incomming_message[22] - 48)*100000;
    lat_gps_actual += (incomming_message[23] - 48)*10000;
    lat_gps_actual += (incomming_message[24] - 48)*1000;
    lat_gps_actual += (incomming_message[25] - 48)*100;
    lat_gps_actual += (incomming_message[26] - 48)*10;
    lat_gps_actual /= 6;
```

Algorithm 7 Latitude and Longitude data storage (continued)

```
35:     lat__gps__actual += (incomming__message[17] - 48)*100000000;
36:     lat__gps__actual += (incomming__message[18] - 48) *10000000;
37:     lat__gps__actual /= 10;
38:     lon__gps__actual = (incomming__message[33] - 48)*10000000;
39:     lon__gps__actual += (incomming__message[34] - 48)*1000000;
40:     lon__gps__actual += (incomming__message[36] - 48)*100000;
41:     lon__gps__actual += (incomming__message[37] - 48)*10000;
42:     lon__gps__actual += (incomming__message[38] - 48)*1000;
43:     lon__gps__actual += (incomming__message[39] - 48)*100;
44:     lon__gps__actual += (incomming__message[40] - 48)*10;
45:     lon__gps__actual /= 6;
46:     lon__gps__actual += (incomming__message[30] - 48)*1000000000;
47:     lon__gps__actual += (incomming__message[31] - 48)*100000000;
48:     lon__gps__actual += (incomming__message[32] - 48)*10000000;
49:     lon__gps__actual /= 10;
50: end if
```

Chapter 5

Simulation model

In this chapter, a simulation model of the quadcopter is developed to test control algorithms and preliminary tune controller parameters. The translational and rotational dynamics of the drone are simulated using Newton's law and Euler's equation, while DC motors are simulated through first-order transfer functions. PID and LQR control algorithms are implemented to perform attitude control and controller parameters are identified in order to decrease rising times and overshoots.

5.1 Quadcopter Flight Mechanics

A quadcopter is controlled by changing the rotor's speed to generate control torques along the three body axes and allow the quadcopter to roll, pitch, and yaw. Two rotors and propellers are provided for each arm of the quadcopter: two opposite lying rotors spin in one direction, while the other two spin in the opposite direction. This prevents the quadcopter will spin around itself because of the contrast torque produced by motors. Changing the rotor's speed, the thrust of each motor changes itself, and control torques can be generated. In Fig. 5.1 thrust, roll, pitch, and yaw maneuvers are shown with respect to the thrust and angular rate for each motor. Such a configuration allows the quadcopter to perform lateral, backward, and forward movements by rolling and pitching respectively, while altitude is controlled by changing by the same amount the thrust of each motor. It should be noted that every maneuver can be performed separately from each other, thanks to the configuration and rotor's spinning direction. Due to its configuration, a quadcopter is intrinsically unstable: if the rotor's speed can not be changed, the quadcopter will be unable to perform the auto-stabilizing flight. This means that when a disturbance occurs, the quadcopter will lose its stability immediately. Therefore, a control algorithm to control motors is needed to have a statically and dynamically stable vehicle.

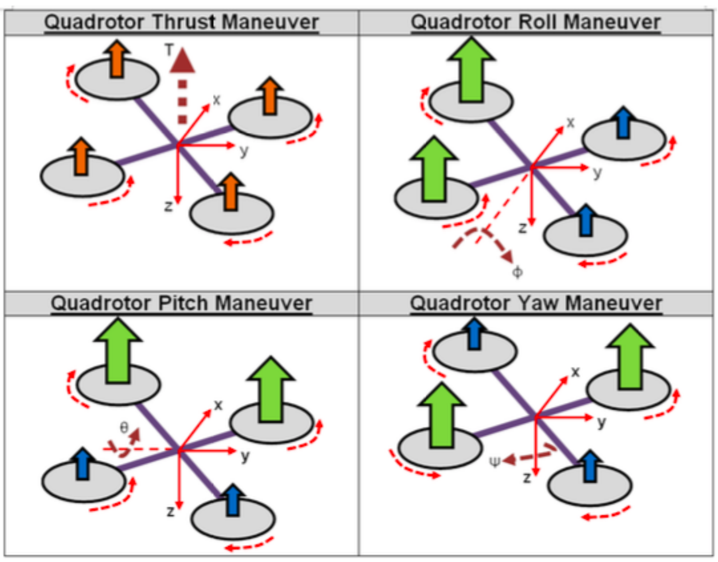


Figure 5.1: How quadcopters perform manoeuvres changing the thrust of each motor

5.2 Mathematical Model

Considering a quadcopter as a rigid body, its dynamics is described by Six Degrees Of Freedom (6DOF), which are three relative to the rotational dynamics and three relative to the translational one. Newton's law and Euler's equation provide respectively linear velocities and angular rate of a rigid body in the body frames. To obtain the position and the attitude of the rigid body, kinematics equations need to be implemented and reference systems have to be introduced. In this section, a mathematical model of quadcopter dynamics and kinematics is developed and a linearization of the model is performed. The state-space form of the linear model is shown: such a representation plays a key role in developing the control algorithm and making considerations about the stability and dynamics of the quadcopter.

5.2.1 Kinematics Equations

Before developing the mathematical model, it is necessary to define reference systems. North-East-Down (NED), defined as in [37] is used as the inertial reference system, while the aircraft body reference system is chosen as shown in Fig. 5.2. The NED frame is identified by $x - y - z$ axes, and $\hat{\boldsymbol{e}}_{\boldsymbol{x}}$, $\hat{\boldsymbol{e}}_{\boldsymbol{y}}$, $\hat{\boldsymbol{e}}_{\boldsymbol{z}}$ are the three unit vectors in the three axes. The body frame is identified by $x_b - y_b - z_b$, and $\hat{\boldsymbol{e}}_{\boldsymbol{1}}$, $\hat{\boldsymbol{e}}_{\boldsymbol{2}}$, $\hat{\boldsymbol{e}}_{\boldsymbol{3}}$ are the three unit vectors in the three axes.

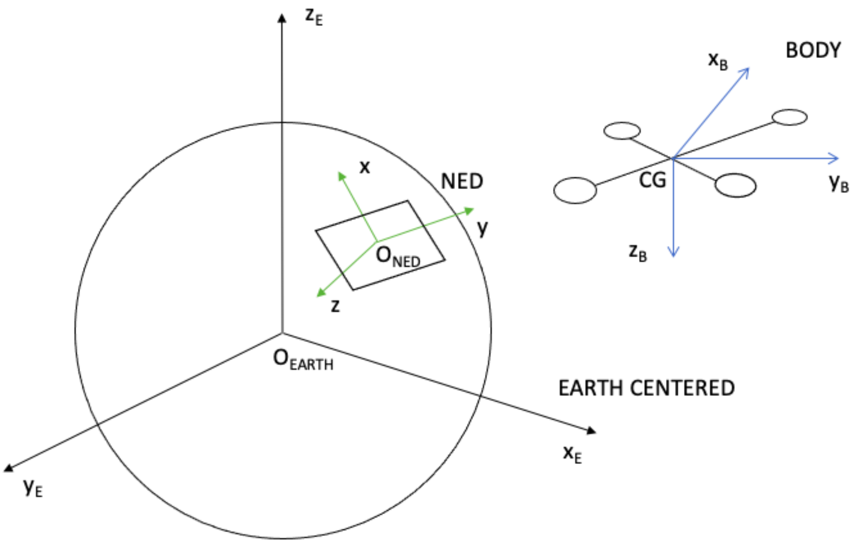


Figure 5.2: Earth Centered, NED and body frame

Euler angles are introduced in Fig. 5.3 to describe the orientation of a rigid body with respect to a fixed coordinate reference system. A rotational matrix $\boldsymbol{R}$ can be used

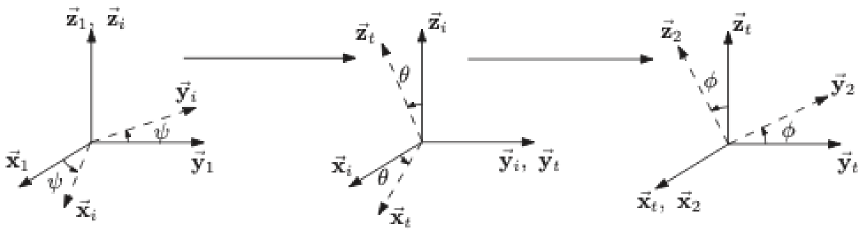


Figure 5.3: A 3-2-1 Euler rotation sequence to obtain a vector in another reference system

to find a vector known in body coordinates, in NED reference system. It can be found as in [38]:

$$\boldsymbol{R} = \begin{bmatrix} c(\theta)c(\psi) & s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) \\ c(\theta)s(\psi) & s(\theta)s(\phi)s(\psi) + c(\phi)c(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix}$$

Defining $\boldsymbol{V} = [\dot{x}, \dot{y}, \dot{z}]^T$ the velocity of the aircraft in the inertial frames and $\boldsymbol{V_B} = [u, v, w]^T$ the velocity of the aircraft in the body frames, it is possible to write:

$$\boldsymbol{V} = \boldsymbol{R} \cdot \boldsymbol{V_B} \tag{5.1}$$

As in [37], considering the angular rate of the quadcopter in NED frame $\boldsymbol{\omega} = \left[\dot{\phi}, \dot{\theta}, \dot{\psi}\right]^T$ and the angular rate of body frame with respect to NED frame $\boldsymbol{\omega_B} = [p, q, r]^T$, it is possible to write:

$$\boldsymbol{\omega} = \boldsymbol{T} \cdot \boldsymbol{\omega_B} \tag{5.2}$$

where

$$\boldsymbol{T} = \begin{bmatrix} 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\phi) \\ 0 & \frac{s(\phi)}{c(\theta)} & \frac{c(\phi)}{c(\theta)} \end{bmatrix}$$

Eq. 5.1 and Eq. 5.2 are the kinematics equations: the Eq. 5.1 expresses the velocity of the aircraft in the inertial frame, while the Eq. 5.2 expresses the angular rate in the body frame in terms of Euler angles. The kinematic equations can be written as follows:

$$\begin{cases} \dot{x} = w\left[s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta)\right] - v\left[c(\phi)s(\psi) - s(\phi)c(\psi)s(\theta)\right] + u\left[c(\psi)c(\theta)\right] \\ \dot{y} = v\left[c(\phi)c(\psi) + s(\phi)s(\psi)s(\theta)\right] - w\left[c(\psi)s(\phi) - c(\phi)s(\psi)s(\theta)\right] + u\left[c(\theta)s(\psi)\right] \\ \dot{z} = w\left[c(\phi)c(\theta)\right] - u\left[s(\theta)\right] + v\left[c(\theta)s(\phi)\right] \\ \dot{\phi} = p + r\left[cos(\phi)tan(\theta)\right] + q\left[s(\phi)t(\theta)\right] \\ \dot{\theta} = q\left[c(\phi)\right] - r\left[s(\phi)\right] \\ \dot{\psi} = r\frac{c(\phi)}{c(\theta)} + q\frac{s(\phi)}{c(\theta)} \end{cases} \tag{5.3}$$

5.2.2 Dynamics Equations

The Newton's law (Eq. 5.4) and the Euler's equation (Eq. 5.5) provide a mathematical model for a 6DOF rigid body. The Newton's law provides the translational dynamics of a rigid body, while the Euler's equation provides the rotational one. The two equations are written as follows:

$$\boldsymbol{F_B} = m\left(\boldsymbol{\omega_B} \wedge \boldsymbol{V_B} + \dot{\boldsymbol{V}}_B\right) \tag{5.4}$$

$$\boldsymbol{M_B} = \boldsymbol{I} \cdot \dot{\boldsymbol{\omega}}_{\boldsymbol{B}} + \boldsymbol{\omega_B} \wedge (\boldsymbol{I} \cdot \boldsymbol{\omega_B}) \tag{5.5}$$

where $\boldsymbol{F_B} = [F_x, F_y, F_z]^T$ is the vector containing the total force applied to the aircraft in body frames, m is the mass of the aircraft, $\boldsymbol{I}$ is the diagonal inertia matrix, and $\boldsymbol{M_B} = [M_x, M_y, M_z]^T$ is the vector containing the total torque applied to the body in body frame. The $\boldsymbol{I}$ matrix is defined as follows:

$$\boldsymbol{I} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}$$

where I_x, I_y, and I_z are the moment of inertia along the three body axes. Note that $\boldsymbol{I}$ matrix is assumed to be diagonal due to the symmetry of the quadcopter's frame with respect to the body reference system.

The 6DOF quadcopter dynamics can be obtained by the following system:

$$\begin{cases} F_x = m\left(\dot{u} + qw - rv\right) \\ F_y = m\left(\dot{v} + pw + ru\right) \\ F_z = m\left(\dot{w} + pv - qu\right) \\ M_x = \dot{p}I_x - qrI_y + qrI_z \\ M_y = \dot{q}I_y + prI_x - prI_z \\ M_z = \dot{r}I_z - pqI_x + pqI_y \end{cases} \tag{5.6}$$

It should be noted that all quantities in Eq. 5.6 are expressed in the body frame. To express the position and the attitude of the quadcopter in the inertial reference system (NED), kinematics equations in Eq. 5.3 have to be considered.

5.2.3 Forces and Moments

The vector $\boldsymbol{F_B} = [F_x, F_y, F_z]^T$ contains the total forces applied to the quadcopter along the three body axes and it is given by:

$$\boldsymbol{F_B} = mg\boldsymbol{R}^T \cdot \boldsymbol{\hat{e}_z} - F_t\boldsymbol{\hat{e}_3} \tag{5.7}$$

where $\boldsymbol{\hat{e}_z}$ is the unit vector in the inertial z-axis, g is the gravitational acceleration, and F_t is the module of the total thrust provided by the four motors. Note that the forces due to the wind or other disturbances are neglected in this treatment. The module of the total thrust provided by the four motors can be written as:

$$F_t = T_1 + T_2 + T_3 + T_4 \tag{5.8}$$

where T_1, T_2, T_3, T_4 are the values of the thrust of each motor.

The vector $\boldsymbol{M_B} = [M_x, M_y, M_z]^T$ contains the total torques applied to the quadcopter in the body frames and it is given by:

$$\boldsymbol{M_B} = \boldsymbol{\tau_B} + \boldsymbol{g_m} \tag{5.9}$$

where $\boldsymbol{\tau_B} = [\tau_x, \tau_y, \tau_z]^T$ is the vector containing the control torques provided by the quadcopter's motors, and $\boldsymbol{g_m} = [g_{mx}, g_{my}, g_{mz}]^T$ is the vector containing the gyroscopic torques caused by the combined rotation of the four rotors and the aircraft body. Note that torques due to the wind or other disturbances are neglected in this treatment. The vector $\boldsymbol{\tau_B}$ is found referring to Fig. 5.4:

$$\begin{cases} \tau_x = (T_1 - T_2 - T_3 + T_4)Lsin(\theta) \\ \tau_y = (T_1 + T_2 - T_3 - T_4)Lcos(\theta) \\ \tau_z = -C_1 + C_2 - C_3 + C_4 \end{cases} \tag{5.10}$$

where L is the length from the rigid body's center of gravity to the motors, θ is defined

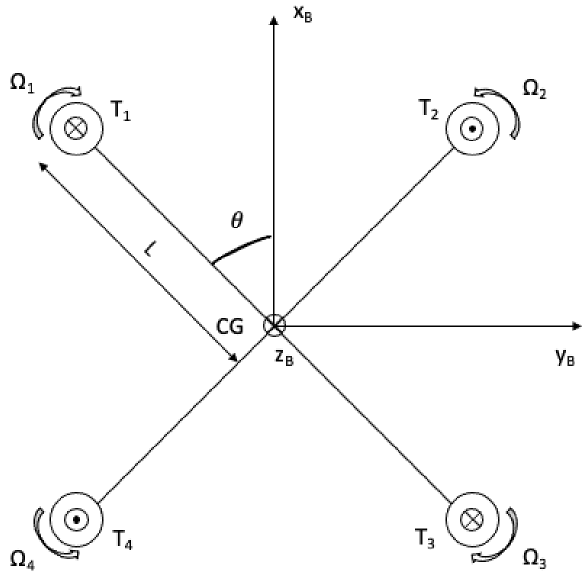


Figure 5.4: Top view of the quadcopter

as in Fig. 5.4, and C_1, C_2, C_3, C_4 are the contrast torques provided by each rotor.

The vector $\boldsymbol{g_m}$ is given by the following relation:

$$\sum_{i=1}^{4} J_p \left(\omega_B \wedge \hat{e}_3\right) (-1)^{i+1} \Omega_i \tag{5.11}$$

where J_p is the moment of inertia of the propellers, $\hat{e}_3$ is the unit vector in the body z-axis, and Ω_i is the angular rate of the i^{th} rotor.

5.2.4 Linearized model

The mathematical model of the quadcopter can be descried by the following non-linear system:

$$\begin{cases} \dot{x} = w\,[s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta)] - v\,[c(\phi)s(\psi) - s(\phi)c(\psi)s(\theta)] + u\,[c(\psi)c(\theta)] \\ \dot{y} = v\,[c(\phi)c(\psi) + s(\phi)s(\psi)s(\theta)] - w\,[c(\psi)s(\phi) - c(\phi)s(\psi)s(\theta)] + u\,[c(\theta)s(\psi)] \\ \dot{z} = w\,[c(\phi)c(\theta)] - u\,[s(\theta)] + v\,[c(\theta)s(\phi)] \\ \dot{\phi} = p + r\,[cos(\phi)tan(\theta)] + q\,[sin(\phi)tan(\theta)] \\ \dot{u} = rv - qw - gsin(\theta) \\ \dot{v} = pw - -ru + gsin(\phi)cos(\theta) \\ \dot{w} = qu - pv + gcos(\theta)cos(\phi) - F_t \\ \dot{\theta} = q\,[cos(\phi)] - r\,[sin(\phi)] \\ \dot{\psi} = r\frac{cos(\phi)}{cos(\theta)} + q\frac{sin(\phi)}{cos(\theta)} \\ \dot{p} = \frac{I_y - I_z}{I_x} rq + \frac{\tau_x}{I_x} \\ \dot{q} = \frac{I_z - I_x}{I_y} pr + \frac{\tau_y}{I_y} \\ \dot{r} = \frac{I_x - I_y}{I_z} pq + \frac{\tau_z}{I_z} \end{cases} \tag{5.12}$$

where the first six equations are kinematic relations, while the other six equations describe the translational and rotational dynamics.

To develop a controller algorithm, a linearized model is needed to easily obtain the eigenvalues that give information about the stability and the dynamics of the system. A linearized model can be used as well to obtain the frequency-domain form of the system to analyze its dynamics through the Bode diagrams.

Assuming small Euler angles and linearizing around an equilibrium point, the system in Eq. 5.12 can be written in this form:

$$\begin{cases} \dot{x} = u \\ \dot{y} = v \\ \dot{z} = w \\ \dot{\phi} = p \\ \dot{\theta} = q \\ \dot{\psi} = r \\ \dot{u} = -g\theta \\ \dot{v} = g\phi \\ \dot{w} = -\frac{F_t}{m} \\ \dot{p} = \frac{\tau_x}{I_x} \\ \dot{q} = \frac{\tau_y}{I_y} \\ \dot{r} = \frac{\tau_z}{I_z} \end{cases} \tag{5.13}$$

Note that to linearize the model, it is assumed that M_x, M_y, and M_z are due only to torques generated by motors along the three body axes.

Once a linear model of the quadcopter dynamics is obtained, the system can be written in the state-space form as follows:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}(t) \end{cases} \tag{5.14}$$

where $\boldsymbol{x}(t) = [x, y, z, \phi, \theta, \psi, u, v, w, p, q, r]^T$ is the state vector, $\boldsymbol{u}(t)$ the control signal, $\boldsymbol{y}(t)$ the controlled output, $\boldsymbol{A}$ the state matrix, $\boldsymbol{B}$ the input matrix and $\boldsymbol{C}$ the output matrix. This form is useful to obtain informations about quadcopter dynamics, and tune controller parameters (see 5.3 section).

Before defining $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$, some considerations have to be done. As discussed in chapter 3 the micro-controller is able to perform control actions by assigning to STM-32 board's PINs some variables which store the pulse width of the PWM signal. This allows motors to change the rotors' speed according to control efforts. Therefore, the control signal $\boldsymbol{u}(t)$ of the state-space representation (Eq. 5.14) should be the pulse length of the PWM signal itself. It should be noted that the linearized model introduced in Eq. 5.13 needs as input the control torques provided by propellers and not the pulse length of the signal. Such a model is useless to identify controller parameters for the developed Flight Controller. Thus, a model able to convert the PWM signal for each motor into torques applied to the quadcopter is needed, so that the controller parameters can be found uniquely among the simulation model and the Flight Controller software.

Considering Fig. 5.4, it is possible to find a relation between the control torque components τ_x, τ_y, τ_z, and the variation of the thrust vector defined as $\boldsymbol{\Delta T} = [\Delta T_x, \Delta T_y, \Delta T_z]$:

$$\begin{cases} \Delta T_x = \frac{\tau_x}{4Lsin(\theta)} \\ \Delta T_y = \frac{\tau_y}{4Lcos(\theta)} \\ \Delta T_z = \frac{\tau_z}{4} \end{cases} \tag{5.15}$$

The variation of the thrust vector $\boldsymbol{\Delta T}$ has as first component (ΔT_x) the variation of the thrust required by each motor to provide the quadcopter with a control torque along the x-body axis (τ_x), as second component (ΔT_y) the variation of the thrust required by each motor to provide the quadcopter with a control torque along the y-body axis (τ_y), and as third component (ΔT_z) the variation of the thrust required by each motor to provide the quadcopter with a control torque along the z-body axis (τ_z).
Afterwards, a relation between the pulse width $\boldsymbol{\Delta PWM} = [\Delta PWM_x \Delta PWM_y \Delta PWM_z]$ and the thrust variation vector $\boldsymbol{\Delta T}$ needs to be found to finally have as control input of the system the pulse width of the PWM signal. For this purpose some experimental tests are carried out to find the thrust provided by one of the motors with respect to the pulse width of the PWM signal. After collecting these data, a linear interpolation can be performed and a linear

relation can be found:

$$\boldsymbol{\Delta T} = n\boldsymbol{\Delta PWM} \tag{5.16}$$

where n is the slope of the linear relation between the thrust of each motor T and the pulse width of the PWM signal ΔPWM. Thus, the control torque components can be found with respect to the pulse width as follows:

$$\begin{cases} \tau_x = 4nLsin(\theta)\Delta PWM_x \\ \tau_y = 4nLcos(\theta)\Delta PWM_y \\ \tau_z = 4n\Delta PWM_z \end{cases} \tag{5.17}$$

Substituting the Eq. 5.17 in the linearized system of Eq.
refeq:lin, the rotational dynamics of the quadcopter can be written with respect to the pulse width variation as follows:

$$\begin{cases} \dot{x} = u \\ \dot{y} = v \\ \dot{z} = w \\ \dot{\phi} = p \\ \dot{\theta} = q \\ \dot{\psi} = r \\ \dot{u} = -g\theta \\ \dot{v} = g\phi \\ \dot{w} = -\frac{F_t}{m} \\ \dot{p} = \frac{4nLsin(\theta)\Delta PWM_x}{I_x} \\ \dot{q} = \frac{4nLcos(\theta)\Delta PWM_y}{I_y} \\ \dot{r} = \frac{4n\Delta PWM_x}{I_z} \end{cases} \tag{5.18}$$

Now the state-space model can be defined properly:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{4nLsin(\theta)}{I_x} & 0 & 0 \\ 0 & 0 & \frac{4nLcos(\theta)}{I_y} & 0 \\ 0 & 0 & 0 & \frac{4n}{I_z} \end{bmatrix}$$

and $\boldsymbol{C} = \boldsymbol{I}_{12}$, $\boldsymbol{u}(t) = [F_t, \Delta PWM_x, \Delta PWM_y, \Delta PWM_z,]^T$ is defined as the control input as desired. $\boldsymbol{I}_{12} \in \mathbb{R}^{12,12}$ and it is the identity matrix.

5.3 Control Algorithms

Due to intrinsically instability of the quadcopter configuration, a control algorithm needs to be implemented to control the rotor's speed, allowing the quadcopter to perform the auto-stabilizing flight or following signals from a transmitter. The quadcopter dynamics is simulated through the mathematical model developed in the last section and two different controllers are tested. In this section the Proportional-Integral-Derivative and the Linear Quadratic Regulator controller are both introduced and in the next section they will be implemented and compared.

5.3.1 PID controller

The Proportional-Integrative-Derivative controller is the most common feedback controller for several applications. It is described by the following equation:

$$\boldsymbol{u}(t) = K_P\boldsymbol{e}(t) + K_I\int_0^t \boldsymbol{e}(t)dt + K_D\frac{d\boldsymbol{e}(t)}{dt} \tag{5.19}$$

defining $y(t)$ as the measured process variable and $y_{sp}(t)$ as the reference variable, $u(t)$ is the control signal and $e(t) = y_{sp}(t) - y(t)$ is the control error. The reference variable is often called the set-point and represents the signal to be tracked by the plant. The control signal is thus a sum of three terms: the P-term (which is proportional to the error), the I-term (which is proportional to the integral of the error), and the D-term (which is proportional to the derivative of the error) (Fig. 5.5) [39]. The controller parameters are the proportional gain K_P, the integral gain K_I, and the derivative gain K_D.

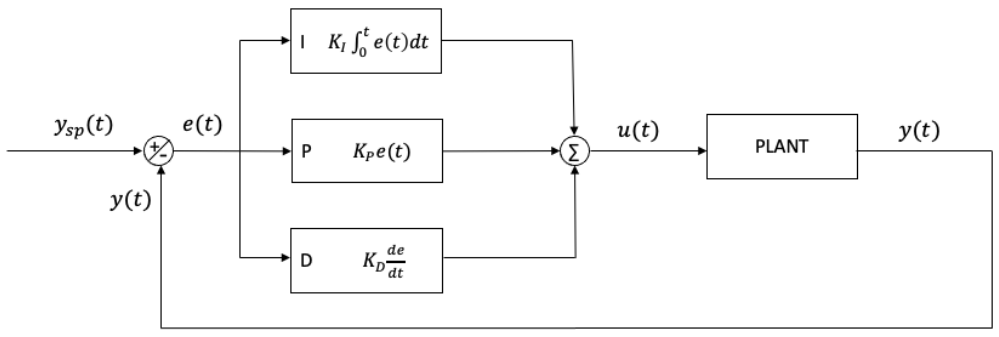


Figure 5.5: PID controller scheme

5.3.2 Wind-up and high-frequency noises

Some issues are introduced by the integral and derivative component of a PID controller and they have to be fixed to obtain a stable response to disturbances and commands. Wind-up is a non-linear phenomenon that occurs when actuators saturate. In PID controllers, the wind-up phenomenon occurs when the amount of the error increases, so the integral contribute increases as well. When this happens, the plant operates in open-loop dynamics, because actuators that are saturated can not perform control action. This could be very dangerous, especially for a quadcopter that is intrinsically unstable in open-loop dynamics. Another issue introduced by PID controllers is related to the derivative contribute. Since the derivative contribution is related to the variation of the error, high-frequency noises from sensors would largely increase the derivative contribution, introducing instability in the response. This appears clear looking at the Bode diagram of the s transfer function (Fig. 5.6) which is used to differentiate the error signal: at high frequencies the module goes to infinity, explaining how sensitive the differentiator is at high frequency. There are several ways to overcome these issues and they consist of algorithms for wind-up issues and filtering for noises. There are several anti-wind-up methods such as back-calculation, observer approach, and many others described in [40]. They mainly consist of algorithms that set to zero integrative contribution when a condition is verified. Also, adaptive methods have been implemented for some applications.

As concerns noises, input signals to the differentiator use to be filtered by a low pass filter avoiding high frequency noises. Typically, a first-order low pass filter is employed and it is more than enough for most applications.

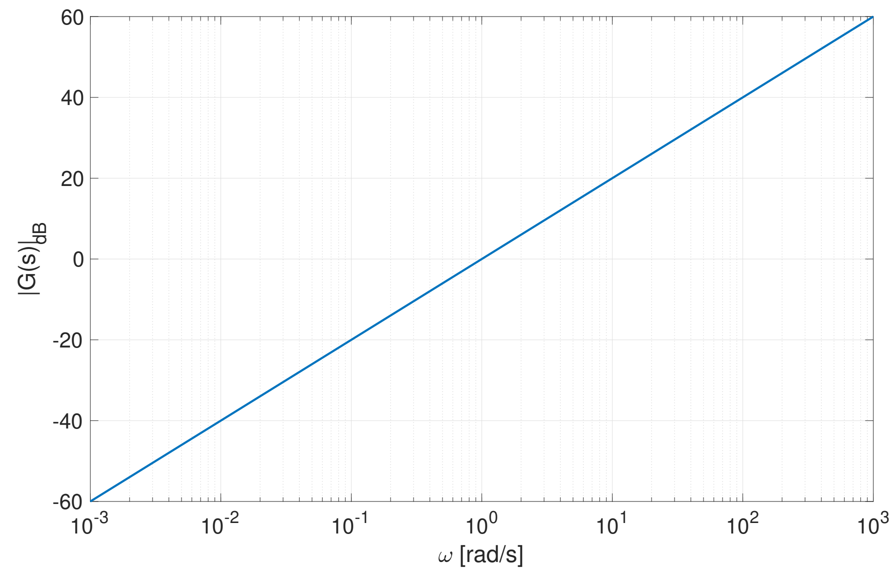


Figure 5.6: Bode magnitude of $G(s) = s$ to show differentiator high sensitivity at high frequencies

5.3.3 LQR controller

The Linear Quadratic regulator controller is based on an algorithm to find a control signal $\boldsymbol{u(t)}$ that minimizes a cost function. The problem consists of finding a linear control law of the type,

$$\boldsymbol{u(t)} = -\boldsymbol{Kx(t)} \tag{5.20}$$

where $\boldsymbol{K}$ is the feedback gain-matrix. To find the control signal $\boldsymbol{u(t)}$ that minimizes the cost function, the performance index (PI) is introduced:

$$J(\boldsymbol{x},\boldsymbol{u}) = \frac{1}{2}\int_0^\infty \left(\boldsymbol{x}^T\boldsymbol{Qx} + \boldsymbol{u}^T\boldsymbol{Ru}\right)dt \tag{5.21}$$

Substituting Eq. 5.20 into Eq. 5.21 yields:

$$J(\boldsymbol{x},\boldsymbol{u}) = \frac{1}{2}\int_0^\infty \boldsymbol{x}^T\left(\boldsymbol{Q} + \boldsymbol{K}^T\boldsymbol{RK}\right)dt \tag{5.22}$$

The feedback gain matrix $\boldsymbol{K}$ has the form,

$$\boldsymbol{K} = \boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} \tag{5.23}$$

where P is the solution of algebraic Riccati equation given in Eq. 5.24:

$$\boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{PA} + \boldsymbol{Q} - \boldsymbol{PBR}^{-1}\boldsymbol{B}^T\boldsymbol{P} = 0 \tag{5.24}$$

where $\boldsymbol{Q} \geq 0$, $\boldsymbol{R} > 0$, $\boldsymbol{P} \geq 0$ are symmetric, positive definite and semi-positive matrices respectively defined as the state and control weighting matrices,

$$\boldsymbol{Q} = diag\left[Q_1, Q_2, ..., Q_{ns}\right] \tag{5.25}$$

$$\boldsymbol{R} = diag\left[R_1, R_2, ..., R_{na}\right] \tag{5.26}$$

ns is the number of the states while na is the number of actuators.
$\boldsymbol{K}$ matrix is found in MATLAB by the *lqr* function that solves algebrical Riccati Eq. 5.24,

$$\boldsymbol{K} = lqr(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}, \boldsymbol{R}) \tag{5.27}$$

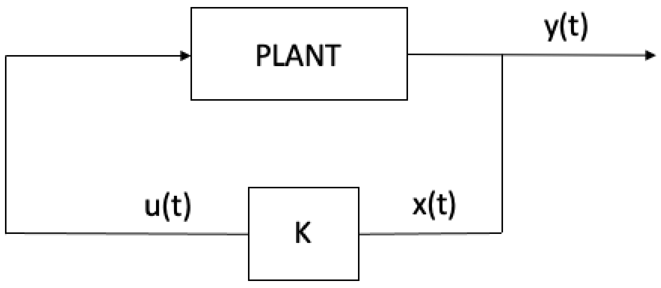


Figure 5.7: LQR controller scheme

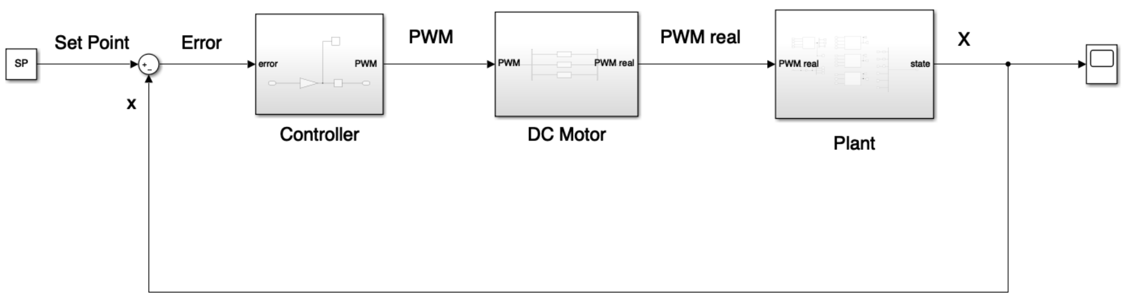


Figure 5.8: Implementation of the model in Simulink environment

5.4 Model Implementation

MATLAB and Simulink are used to implement the simulation model based on quadcopter dynamics described above (Fig. 5.8).

It mainly consists of the following blocks:

- Controller: provides the motors with the PWM signal to follow the set-point signal from the receiver.
- DC motor: introduces DC motor's dynamics.
- Plant: simulates quadcopter dynamics.

The Controller and Plant blocks have been discussed in previous sections 5.3 5.2. DC motors are simulated by a first order transfer function to introduce in the model delays due to DC motors dynamics:

$$PWM_{real} = \frac{1}{\tau s + 1} PWM \tag{5.28}$$

where τ is the time constant assumed to be 0.02 s.

The thrust coefficient n and the moments of inertia of the drone are evaluated experimentally in App. A and they are given as input to the simulation model.

The linearized model shown in Eq. 5.18 and its state-space representation is used to set controller parameters and find the gain matrix K of LQR controller introduced in 5.3.3.

5.5 Simulation results

In this section simulation results about roll, pitch, and yaw response to commands are shown. PID and LQR controller parameters are tuned to obtain a fast response to commands with a low overshoot and a zero steady-state error. In the Table 5.1, controller's performance are compared. The controller parameters used for PID and LQR design are shown in App. B. Both the controllers perform a satisfactory step

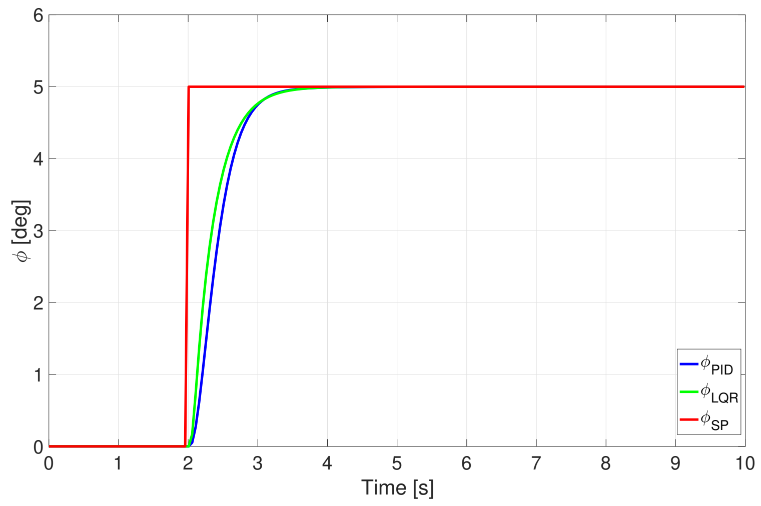


Figure 5.9: Roll response to a step command

response for the quadcopter. This controller design shows that the LQR controller performs quite better than the PID, because of the lower rising and settling time for all three axes.

Note that, as opposed to the roll and pitch angle, the yaw attitude is controlled by acting on the yaw rate r. This is done because a quadcopter is usually controlled by commanding a yaw rate from the transmitter. Since a quadcopter often needs to spin around itself, it is more intuitive to the pilot to command a yaw rate than an angle from the stick of the transmitter. Actually, some quadcopters are driven commanding an angular rate even for the roll and pitch angle. However, in this work, it is preferred

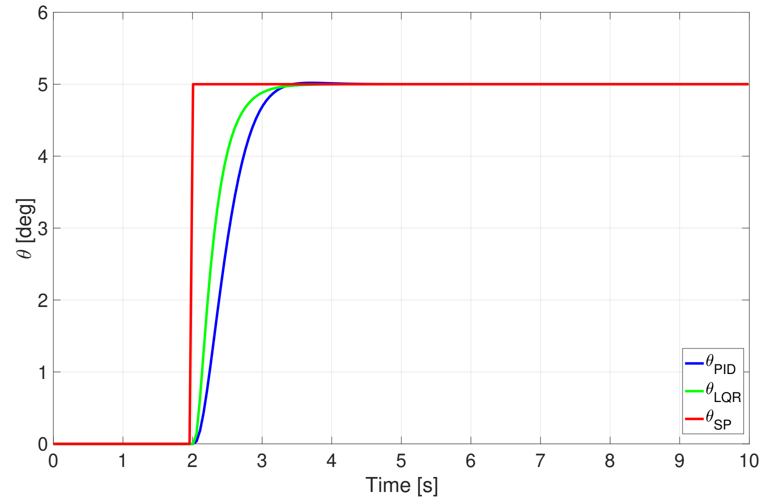


Figure 5.10: Pitch response to a step command

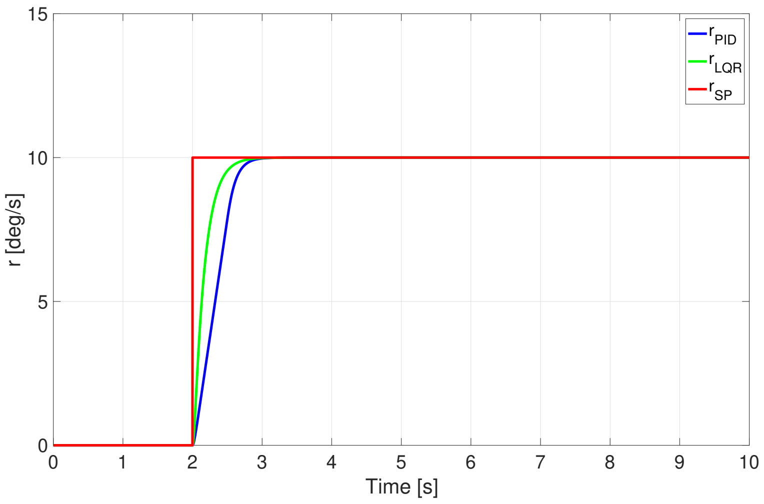


Figure 5.11: Yaw rate response to a step command

to drive the drone by commanding angles for the roll and pitch dynamics, and yaw rates for the yaw.

In Fig. 5.12 and Fig. 5.13, commands sent to the drone from a transmitter during a flight test are given as input to the model. These results confirm that the LQR controller enables the drone to perform roll and pitch maneuver properly. Controller parameters used in the simulation environment are tested during flight tests with good results. Although during flight tests another tuning has been performed to obtain better performance, the controller parameters identified by the simulation model results to be very useful for the final design.

Time Specifications	**Roll**	**Pitch**	**Yaw**
Rising Time PID	0.71 s	0.75 s	0.51 s
Rising Time LQR	0,66 s	0.58 s	0.34 s
Settling Time PID	0.98 s	1.04 s	0.67 s
Settling Time LQR	0.97 s	0.81 s	0.48 s
Overshoot PID	0%	0.01%	0%
Overshoot LQR	0%	0%	0%

Table 5.1: Roll, pitch, and yaw: step response performance

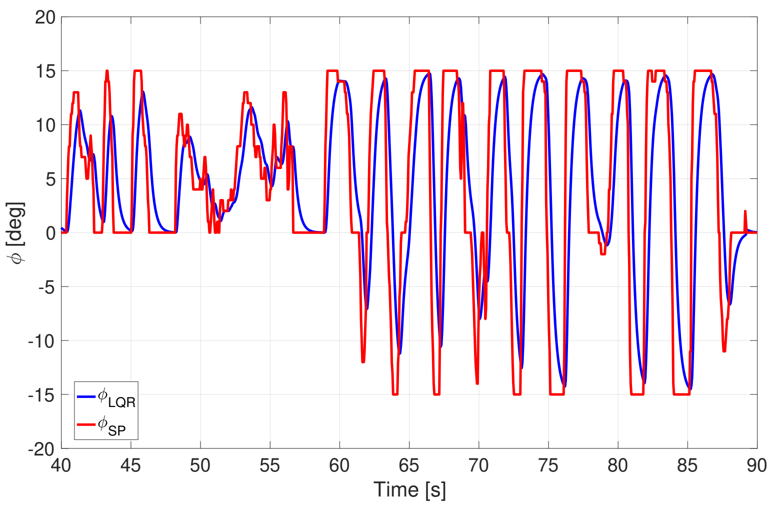


Figure 5.12: Roll response to commands

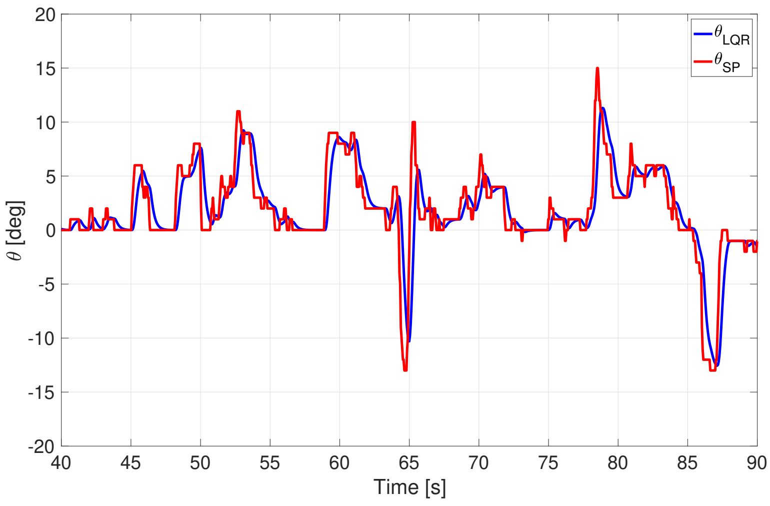


Figure 5.13: Pitch response to commands

Chapter 6

Hardware components

In this chapter, hardware components built on the developed Flight Controller and their specifications are introduced. After soldering all components, the Flight Controller is realized and a schematic shows all connections and components arrangement on the PCB.

6.1 Introduction

To develop a Flight Controller for a rotary-wing UAVs, a micro-crontroller is programmed to elaborate information from sensors and manage signals from the transmitter. Also, a micro SD card is used for data logging during flight and an SD card adapter allows to write flight data on the card. Hardware components implemented on the developed Flight Controller are:

- Arduino STM-32F103C8T6 microcontroller
- MPU-6050 gyroscope/accelerometer
- FlySky Fs-i6 transmitter and receiver
- GNSS/GPS module
- SD card adapter

6.1.1 Arduino STM-32

The STM32F10xxx is a family of microcontrollers with different memory sizes, packages, and peripherals [34]. As seen in section 1.3, STM-32 series of micro-controller is largely employed in commercial Flight Controller. For this reason, it is chosen to develop the Flight Controller on this board.
The STM-32F103C8T6 (Fig. 6.1), aslo known as *blue pill*, houses a 5V to 3.3 V voltage regulator. The MCU operates at 3.3 V but most of its GPIO PINs are 5 V tolerant. It is a 32 bit ARM architecture and "F103" stands to indicate the ARM Cortex M3 architecture. In Table 6.1 its specifications are shown [34].

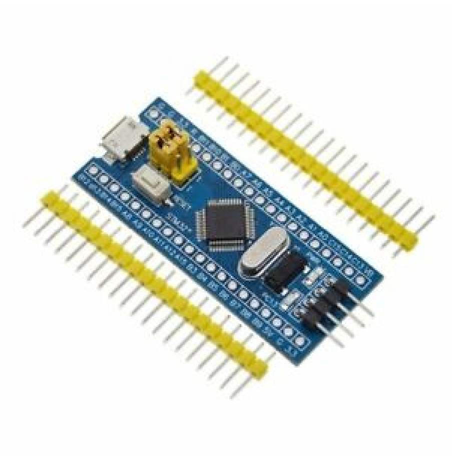


Figure 6.1: STM-32F103C8T6 board

STM-32F103C8T6	
CPU Frequency	72 MHz
Number of GPIO PINs	32
Number of PWM pins	12
Analog input PINs	10 (12-bit)
USART Peripherals	3
I2C Peripherals	2
SPI Peripherals	2
Flash Memory	64 KB
RAM	20kB

Table 6.1: STM-32F103C8T6 specifications

To program the STM-32 from Arduino IDE, an Ft232rl FTDI adapter is used (Fig. 6.2). In Fig. 6.3 the STM-32 pinout is shown [41].

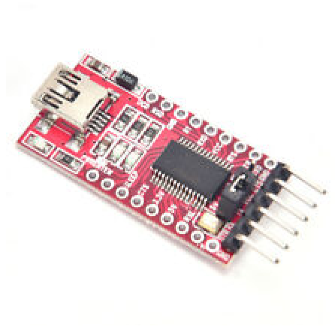


Figure 6.2: Ft232rl FTDI adapter

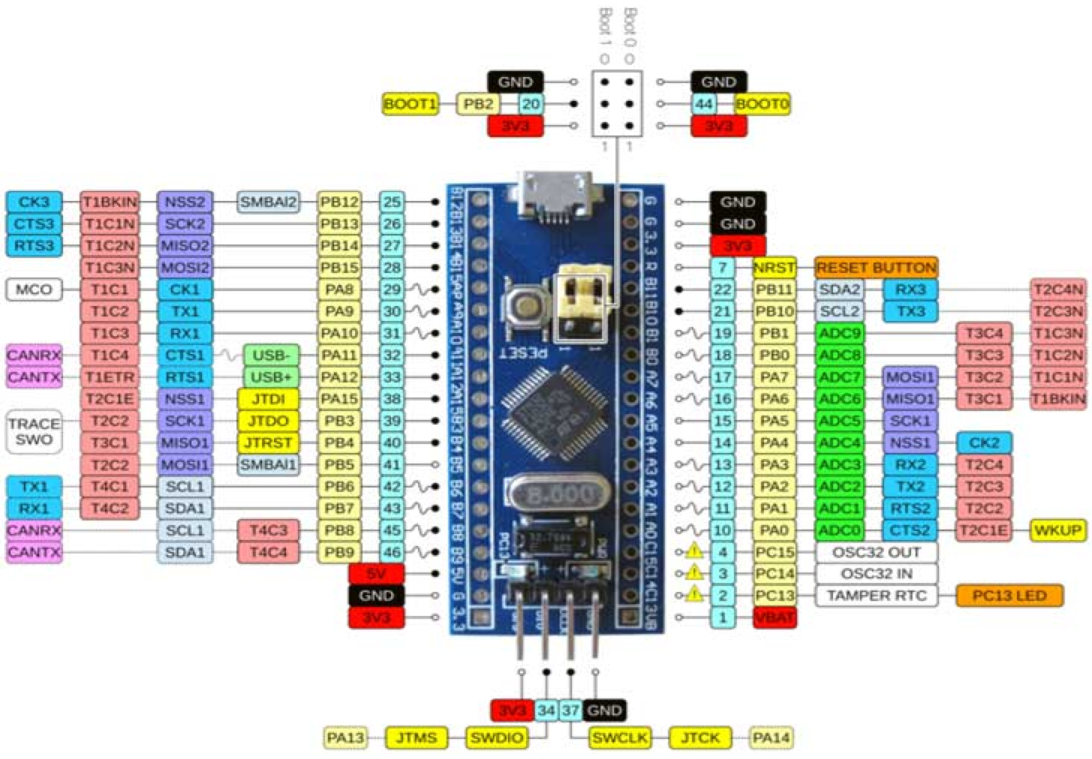


Figure 6.3: STM-32 Pinout

6.1.2 MPU-6050

The MPU-6050 (Fig. 6.4) [42] is a sensor based on Micro Electro Mechanical Systems (MEMS) technology. Both accelerometer and gyroscope are included in the sensor which makes use of the I2C communication protocol to interface microcontroller, such as Arduino boards. It provides also a temperature sensor and it can be easily interfaced with a magnetometer. Its PINs are shown in Fig. 6.4.

- SCL: is the I2C serial clock.
- SDA: is the I2C serial data pin.
- XDA: is the I2C master serial data pin. This pin is used for connecting external sensors.
- XCL: is the I2C master serial clock. This pin is used for connecting external sensors.
- AD0: is the I2C slave address LSB pin.
- INT: is the interrupt digital output pin

As it will be shown in Fig. 6.9, only SCL, SDA, and power supply is needed to enable the communication between the MPU-6050 and STM-32 board. MPU-6050 specifications are summarised in Table 6.2 [31].

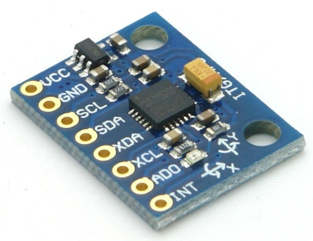


Figure 6.4: MPU-6050 gyroscope/accelerometer MEMS

MPU-6050	
ADC	16 bits
FIFO buffer	1024 byte
Gyroscope range	±250, ±500, ±1000, ±2000°/sec
Accelerometer range	±2g, ±4g, ±8g,16g
Communication protocol	400 KHz fast mode I2C
Operating power supply	3-5 V

Table 6.2: MPU-6050 specifications

6.1.3 FlySky Fs-i6

In this work, a FlySky Fs-i6 transmitter and receiver (Fig. 6.6) [43] is employed to send commands and drive the drone. The transmitter has six channels and four switches and a power input of 6 V DC. The receiver is able to manage PPM signals if the transmitter's settings are changed from the default ones. The transmitter also provides a low voltage alarm in case the voltage is less than 4.2 V. In Tables 6.3 and 6.4, transmitter and receiver specifications are summarized.

Figure 6.5: FlySky Fs-i6 transmitter and receiver

FlySky Fs-i6 transmitter	
Channels	6
RF range	2.405-2.475GHz
Bandwidth	500KHz
Band	142
RF power	less than 20dBm
Sensitivity	1024
Weight	392g

Table 6.3: FlySky Fs-i6 transmitter specifications

FlySky Fs-i6 receiver	
Channels	6 Channels PWM PPM i-Bus
Frequency range	2.4–2.48GHZ
Band width number	140
Transmitting power	not moer than 20dBm
RX sensitivity	-105dBm
Antenna length	26mm
Input power	4.0-6.5V DC
Dimension	40 * 21 * 7mm
Weight	6.4g

Table 6.4: FlySky Fs-i6 receiver specifications

6.1.4 GNSS/GPS module

An $u - blox$ GNSS module from the NEO-M8 series is used in this work. As said in section 4.2, this module is able to provide concurrent receptions of up to three GNSS signals (GPS, Galileo, GLONASS, BeiDou) and it also features a compass module. The module used in this work is the NEO-M8M which is cost-effective, while other modules such as the NEO-M8N and NEO-M8Q provide better performance and easier RF integration. Some of the main features of the NEO-M8M module are collected in Table 6.5.

Figure 6.6: ublox NEO-M8M module

ublox NEO-M8M	
Sensitivity	-167 dBm
Update rate single GNSS	up to 18 Hz
Update rate 2 concurrent GNSS	up to 10 Hz
Position accuracy	2.0 m
Memory	ROM
Oscillator	Crystal
Supported Antennas	Active and passive
Power supply	1.65 V to 3.6 V
Operating temp.	-40 to +85

Table 6.5: u-blox NEO-M8M specifications

6.1.5 SD card adapter

A micro SD card is used to log data during flight. The STM-32 board is enabled to create a test file by communicating with an SD card adapter shown in Fig. 6.7 [44]. Such an SD card adapter makes use of Serial Peripheral Interface (SPI) communication to interface the STM-32 board. A pins overview follows:

- MISO: is SPI output from the Micro SD Card Module.

- MOSI: is SPI input to the Micro SD Card Module.

- SCK: accepts clock pulses which synchronize data transmission generated by Arduino.
- CS: is used by Arduino (Master) to enable and disable specific devices on SPI bus.

It is used a 2 GB SD card (Fig. 6.8), which is more than enough to collect flight data. Then, graphics can be obtained post-processing these data though a Matlab script.

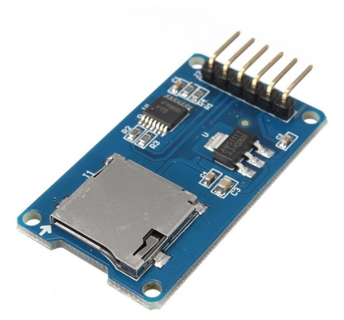


Figure 6.7: SD card adapter used to interface STM-32 board

Figure 6.8: 2 GB micro SD card used to log filgh data

6.2 Hardware connections

In this section, a detailed schematic of the developed Flight Controller is provided (Fig. 6.9). Thick lines represent low current wirings while thick lines represent high current ones. Then, dots are used to represents connections between two or more wirings. Finally, all hardware components are soldered on a Printed Circuit Board (PCB).

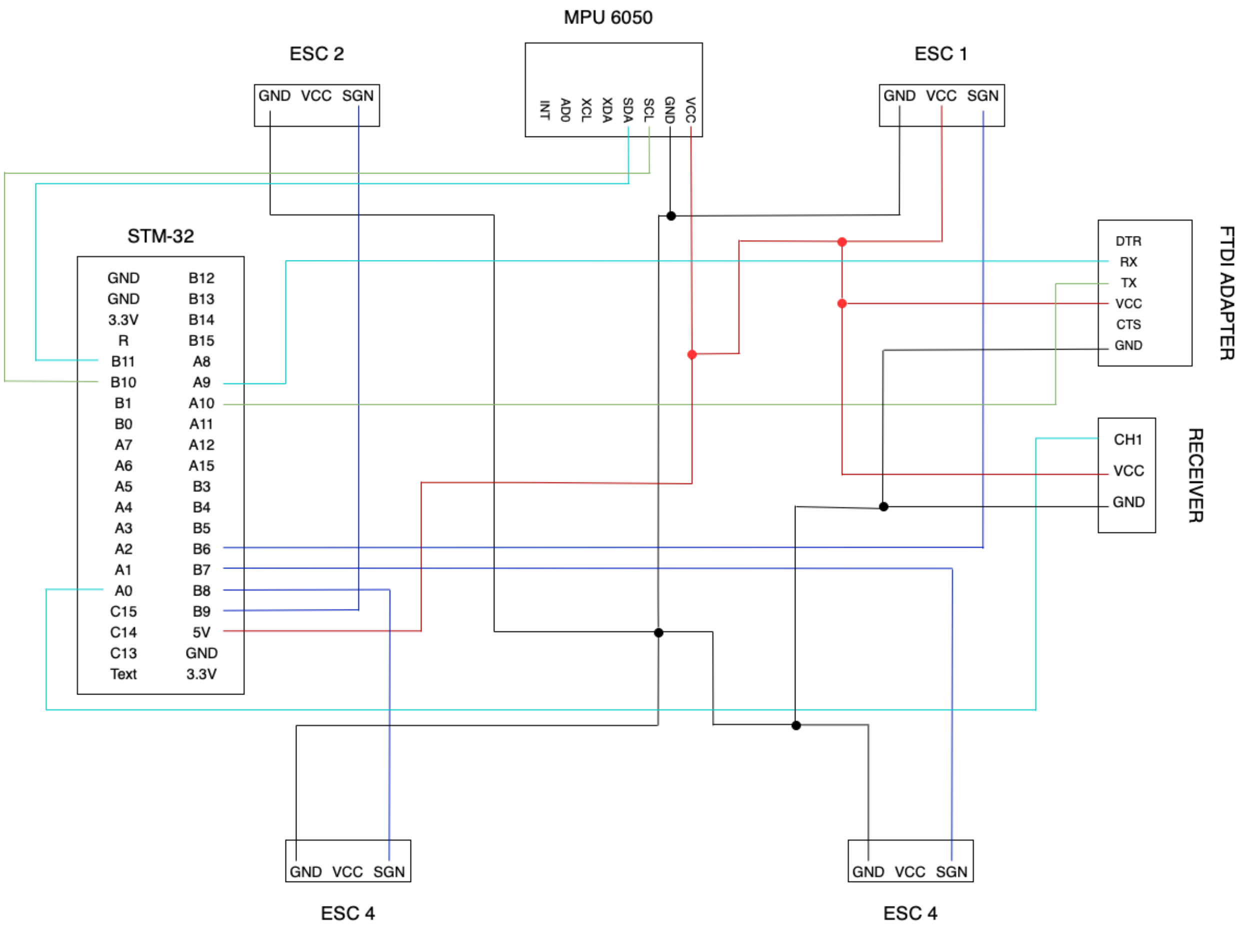


Figure 6.9: Flight Controller schematic

Chapter 7

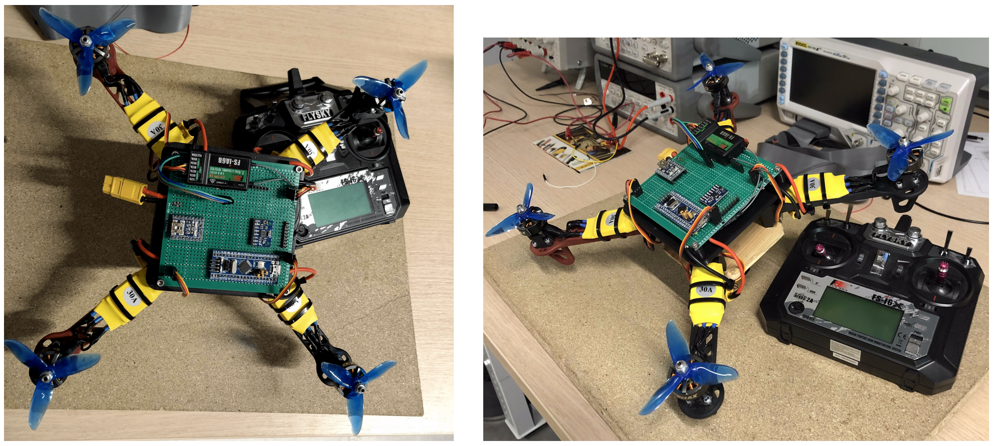
Experimental results

In this chapter, the test bench used to test the developed Flight Controller is shown and its limitations introduced. Therefore, experimental results about roll and pitch responses to commands collected on the bench and during flight tests are shown. Finally, a comparison between results obtained from the simulation model developed in section 5, and experimental data collected during flight is shown.

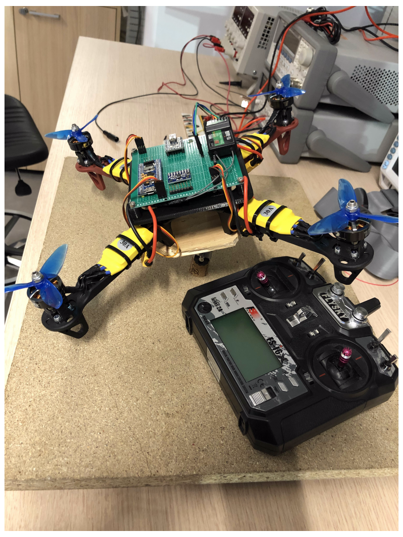
7.1 Test bench

Before testing the Flight Controller during flight, a test bench is realized to tune controller parameters and verify that the auto-stabilizing flight was performed properly. This phase, allowed to reduce tuning times and avoided damaging the drone or the flight controller itself during first tests.

In Fig. 7.1 the test bench is shown: it consists of a joint fixed on a plywood plate where the drone can be engaged on. Therefore, the drone is allowed by the joint to perform only pitch, roll, and yaw, so that tests about the auto-stabilizing can be performed properly and safely. Since tests are carried out in a laboratory, small propellers shown in Fig. 7.1 are used to be safe during the first tests. Despite the small size of propellers, the Flight Controller can perform the auto-stabilizing properly and, after some tunings, a good reaction to disturbances is obtained. However, the test bench results to be unbalanced and asymmetric: this introduces a large disturbance for the drone, which is not able to balance itself with such small propellers when maneuvers are commanded. It should be taken into account also the frictions introduced by the joint, which could be not negligible especially for the yaw control. Another effect to take into account is the ground effect: it is surely of another order of magnitude with respect to the other effects introduced by the joint, but since small propellers are used, it could affect the response quite enough to introduce instability. Therefore, propellers used in flight tests are built on the drone, and controller parameters are tuned again. In this phase, a small amount of current is given to motors to be safe, so again results can not be reliable.



(a) (b)



(c)

Figure 7.1: Pictures of the quadcopter on the test bench and the FlySky-fsi6 transmitter

7.1.1 Auto-stabilizing

In Fig. 7.2, results about the auto-stabilizing performed on the test bench described above are shown.
The throttle is fixed at approximately half of the maximum value of the transmitter, and the LQR controller performs the auto-stabilizing. As explained above, tests on the bench are performed with small propellers to be safe. Therefore, when small errors between attitude signals from sensors and references occur, the thrust of each motor can't be enough to bring the drone back to exactly 0° for both roll and pitch. However, the error is quite small: approximately 0.5° for roll and 0.7° for pitch.

As discussed before, the joint on the bench introduces some effects that disturb a lot of the drone's dynamics, such as frictions, asymmetries, and ground effects. It should be considered that in this phase data have been collected from the serial port and a wire was necessary to connect the Flight Controller to the USB port of a PC. The wire introduces some other disturbances especially for the yaw control, because of the small control torque that can be generated by the motor's spin. Telemetry for wireless communication has been considered in this phase but it was rejected because of the small baud rate that can be achieved. Actually, it becomes computationally expensive writing variables on the serial port and this will increase of an order of magnitude the software working frequency. Therefore, results wouldn't be reliable as the software will be executed at 400 Hz frequency during flight. Despite all disturbances introduces by the test bench, results about the auto-stabilizing are satisfactory.

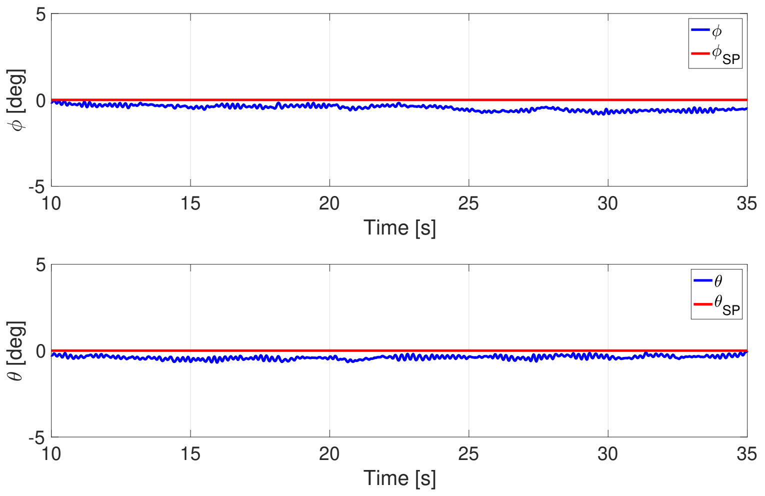


Figure 7.2: Test on the bench: roll and pitch angle during auto-stabilizing without commands or disturbances

7.1.2 Response to commands

Some tests about roll and pitch response to commands are also performed, and results are shown in Fig. 7.3.
These results are more and more affected by disturbances due to the joint and, during pitch and roll maneuvers, the drone encounters some issues in controlling yaw angle too. This is caused by the asymmetries introduced by the joint that unbalance the drone. Thus, motors are required to provide more thrust, and when saturation occurs, they will not be able to control the drone's attitude anymore. Of course, it is also caused by small propellers that are not capable to control the drone if such a disturbance due to joint asymmetries occurs. However, results about pitch and roll angles demonstrate a fast response of the drone to the commands: despite some overshoots, the drone results to be controllable thanks to its fast response to commands. These results encourage to start flight tests and collect flight data without all disturbances introduced by the

joint and wirings. Also, bigger propellers are used during flight tests, so another tuning of the controller parameters is performed.

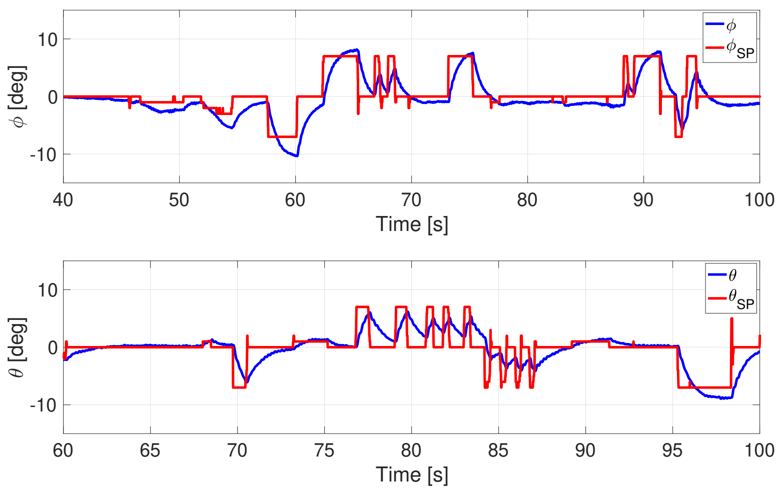


Figure 7.3: Test on the bench: roll and pitch response compared to reference signal (set-point)

7.2 Flight tests and results

In this section results collected during flight tests are shown. Using as a starting point the design tested on the simulation model, LQR controller parameters are tuned again to obtain better performances. Two flight tests are performed and two different LQR designs are implemented on the Flight Controller. Referring to the Table B.3, the graph of Fig. 7.5 refers to "Flight Test 1", while the graphs of Fig. 7.6 and Fig.7.7 refer to "Flight Test 2". The two designs differ on the first component of the $\boldsymbol{Q}$ matrix: this is done to increase the rising time of the roll response and to make the drone more controllable.

Due to the presence of the battery, pitch maneuvers are less accurate than roll ones, especially as regards the steady-state error. Indeed, the battery increases the moment of inertia with respect to the y_b axis and can introduce asymmetries in the mass distribution of the quadcopter.

However, results are satisfactory and the command is tracked properly by the quadcopter. The rising time is estimated to be about 0.6 s for both roll and pitch angles, as expected from the simulation model (see Table 5.1).

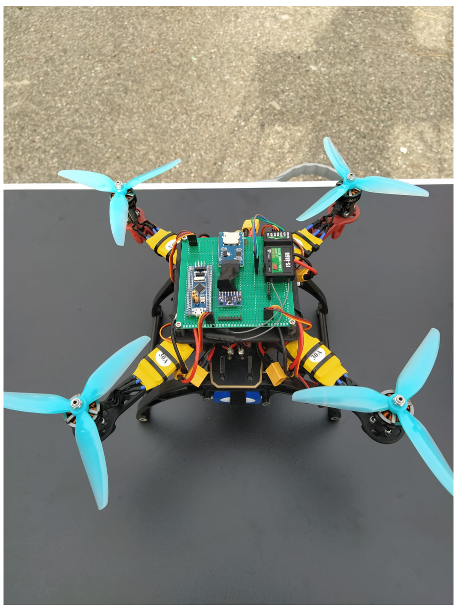


Figure 7.4: Picture of the quadcopter and Flight Controller before flight tests

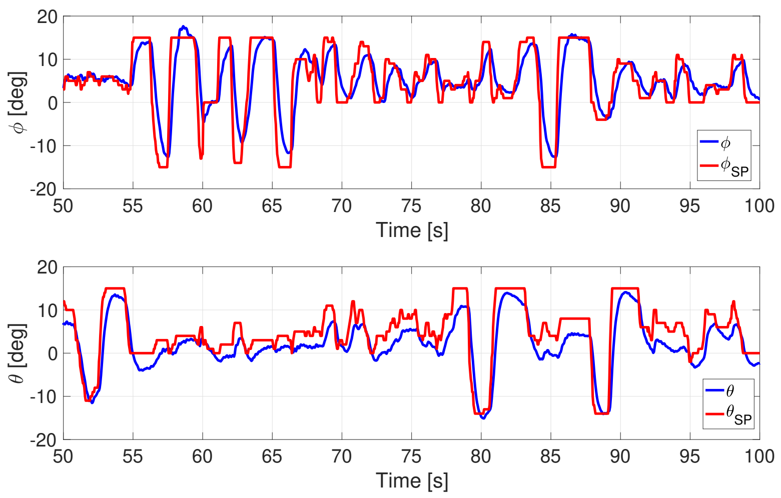


Figure 7.5: Flight test: roll and pitch response compared with commands

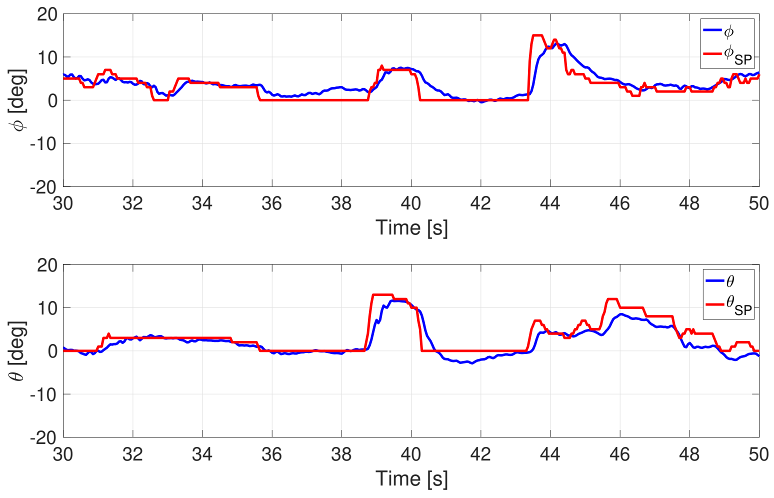


Figure 7.6: Flight test: roll and pitch response compared with commands

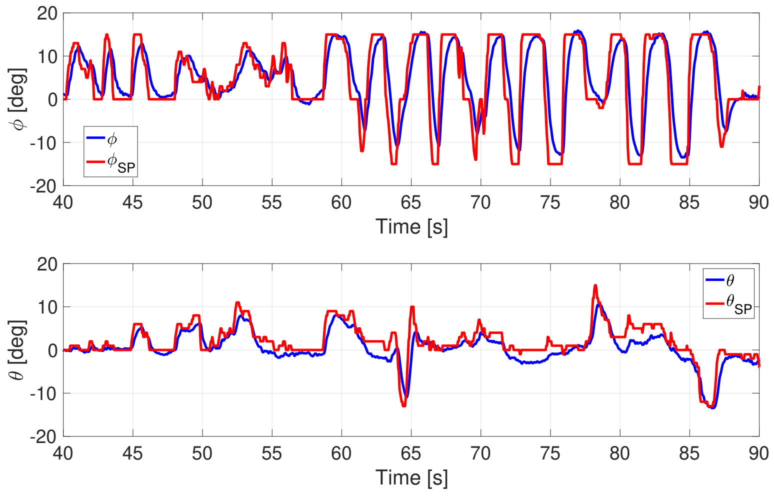


Figure 7.7: Flight test: roll and pitch response compared with commands

Figure 7.8: A picture of the drone during flight tests

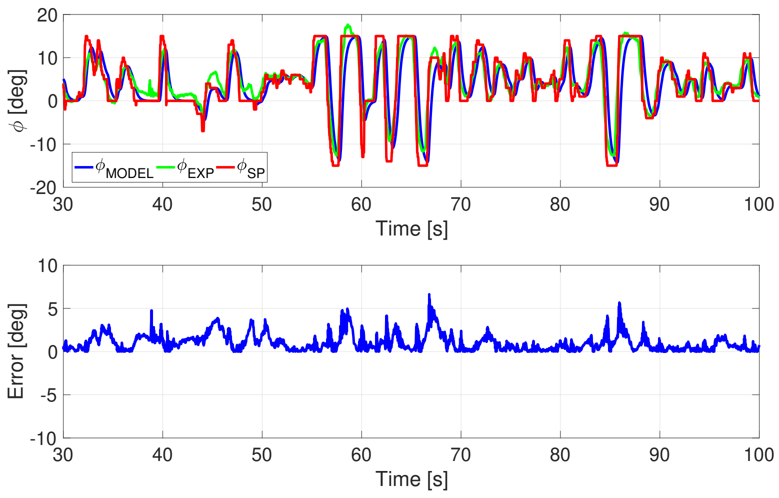
7.3 Comparison between flight tests and simulation results

In this section results obtained from the simulation model are compared to flight tests data. Commands given from the transmitter during flight tests are stored in a Matlab vector, which is given as input to the simulation model.

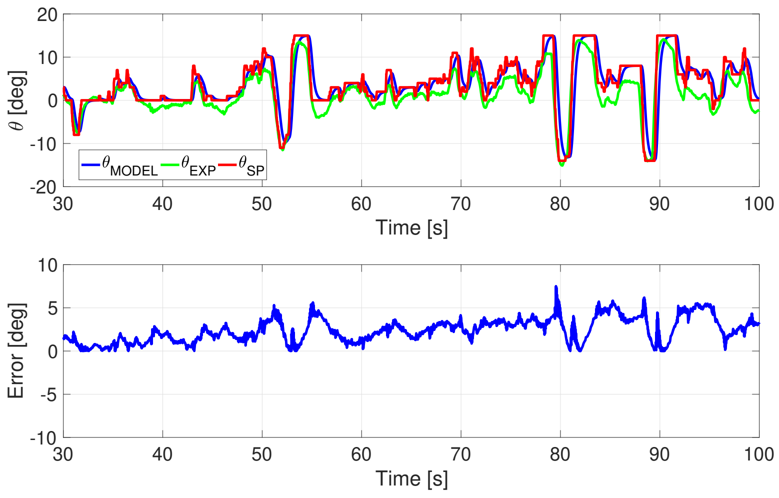
In Fig. 7.9, the experimental and simulation data are compared with satisfactory results. As regards to roll angle, the error between the simulation results and the flight data is minor than 3° for almost the whole flight test. The error increases when fast and great amplitude commands are sent to the drone, but it remains minor than 5°, except for some peaks. It can be seen that the roll angle matches better the experimental data than the pitch angle: the reason can be identified considering that probably the IMU is not built on the drone at exactly 0°, hence the true pitch angle could be closer to the simulation results. Also, it is supposed that the battery is not placed exactly symmetrically with respect to the drone's body axes, instead of it is assumed in the simulation model. However, the matching remains satisfactory, and the developed model can be useful to preliminary set controller parameters and test the controller's performance in a simulation environment. In Table 7.1, the rising time for roll and pitch angles evaluated from simulation results and flight tests is shown. All other step response performance, such as overshoot, steady-state error, and settling time are not evaluated: in fact, appropriate tests have to be carried out to evaluate these quantities properly. However, as regards to quadcopters, the rising time represents the most demanding time specification to make the drone controllable. Even with a non-zero steady-state error, the drone would perform maneuvers properly and it would be easily controllable. From this comparison, the model is found to be accurate for the rising time approximation: with respect to the predictions of the model, the experimental data show an error of about 0.1% for roll and 0.05% for pitch dynamics.

	Simulation Results	**Flight Tests**
Rising Time Roll	0.69 s	0.62 s
Rising Time Pitch	0.6 s	0.57 s

Table 7.1: PID parameters used in the simulation model



(a) *Roll angle*



(b) *Pitch angle*

Figure 7.9: Comparison between flight data and simulation results

Chapter 8

Conclusions and future works

UAVs are getting more and more involved in several applications, and many companies are investing in drones to improve their activities. These vehicles play already a key role in agriculture activities, police service, and surveying, and in less than ten years UAVs could be largely employed for delivering too.

A Flight Controller is the hardware component that ensures flight stability and allows UAVs to perform maneuvers receiving data from sensors and controlling motors. The control algorithm implemented in the Flight Controller's software drives motors in order to follow a reference signal given from a transmitter or a ground station. Flight performance are associated with the control algorithm efficiency and many researchers have been working for last years to improve UAV's flight stability by testing more and more advanced control strategies.

Open-source Flight Controllers are platforms that can be modified in software and in hardware components too. They represent the best solution for the research activity because their software is fully accessible by the users and it can be modified according to the costumer's needs. By accessing the code, not only the control algorithm but also navigation and guidance algorithms, sensor fault detection and many other functions can be implemented to improve UAVs' reliability and extend their applications.

In this thesis, the software of an open-source Flight Controller for a quadcopter is developed and tested on an Arduino STM-32 micro-controller. Flight tests are performed, and experimental data show the capability of the developed Flight Controller to perform properly the auto-stabilizing and the maneuvers during flight.

An IMU is developed to provide the Flight Controller with the drone's attitude and a complementary filter is used to perform the sensor fusion between a gyroscope and an accelerometer. Thanks to the 400 Hz refresh frequency of the software, the Flight Controller provides a fast response to disturbances and commands. To identify the controller design, the most common procedure for the quadcopters is based on experimental tests. For this reason, the PID controller is largely used due to its easy implementation and tuning. Moreover, tuning controller parameters during flight tests could be dangerous for the drone and the hardware components on board. Also, a model-based approach is preferred when more advanced control algorithms have to be implemented. Instead of the most common PID controller, an LQR control algorithm is implemented and tested with satisfactory results. A simulation model of the quadcopter is provided, and experimental tests are carried out to find the thrust provided by each propeller and the drone's moments of inertia. From the comparison of the simulation results and the data collected during flight tests, the simulation model is found to be enough accurate to preliminary design a control algorithm in a simulation environment. Especially as regards the rising time, the model approximates the experimental data with an error no larger than 0.1% for the roll and pitch dynamics. In light of this, a model-based approach can be followed in future works to test new control algorithms and improve flight performance. In fact, once the controller has been tested in the simulation model, a C/C++ code can be generated and implemented directly on the Flight Controller. In order to improve the reliability of the model, other quantities can be experimentally measured in future works (i.e. rotor's torque, and motor's constant).

The developed Flight Controller is also capable of communicating with an $u - blox$ GNSS/GPS module, even if the receiver is not yet included in the Flight Controller's circuit. However, in future works, the GNSS/GPS module will be connected to the Flight Controller and a control algorithm could be implemented to perform position control and autonomous flight.

With the task of making the quadcopter as autonomous as possible during flight, the altitude hold or the obstacle avoidance function can be implemented in future works, accessing the code and adding new sensors to the PCB.

Appendix A

Experimental measurements

In chapter 5, a simulation model of the quadcopter is introduced: the model aims at simulating the quadcopter dynamics, thus some experimental values such as moments of inertia of the drone and the thrust provided by the propellers need to be evaluated. In this appendix, two different test benches are used to evaluate the thrust provided by one of the propellers and the quadcopter's moments of inertia.

The thrusts of the propellers is evaluated through the software provided by RCBenchmark company, while the moments of inertia are measured with a procedure involving a pendulum built in the laboratory.

A.1 Propeller's thrust measurements

Experimental data about the propellers are collected and interpolated to find a relation between the PWM pulse width and the thrust provided by the propeller. Referring to Eq. 5.16, a linear relation is supposed. Experimental tests are carried out to:

- verify that a linear relation can be assumed with a good approximation
- find the n value of Eq. 5.16 to perform the conversion from PWM pulse width to thrust.

The RCBenchmark Series 1520 Thrust Stand (Fig. A.1) is a small size propeller test stand, and it is used in this thesis to collect data about the quadcopter's propellers [45]. By communicating with an USB interface, this device is able to record the thrust provided by the propeller, the RPM, and other quantities (i.e. voltage and current to motors) with respect to the pulse width of the PWM signal. RCBenchmark provides dedicated software to collect these data with respect to the PWM pulse width, which can be changed directly by the software. Data are automatically stored in an Excel file and can be analyzed properly.

As said above, the stand is used to obtain the propeller's thrust, so other data are not collected. In Fig. A.2 the test bench is shown: the motor is fixed on the stand and an ESC is connected to the three PINs of the RCBenchmark. Tests have been performed outside to be safe and to reproduce similar atmospheric conditions during flight tests.

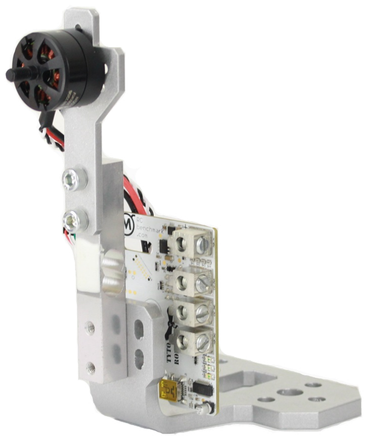
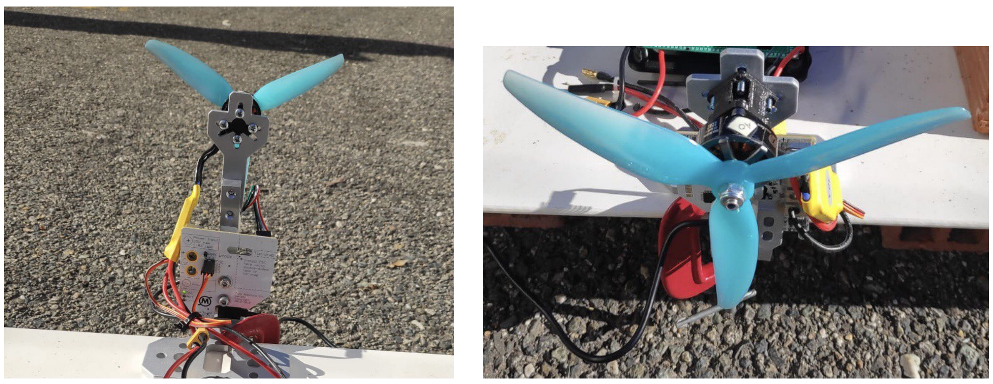


Figure A.1: RCbenchmark Series 1520 Thrust Stand [45]



(a)

(b)

Figure A.2: Pictures of the thrust stand to collect experimental data abut propellers' performance

Data collected during experimental tests are shown in Fig. A.3. A linear relation is found to be a satisfactory approximation for the collected data. Data are interpolated by a first-order polynomial using $polyfit$ and *polyval* MATLAB function. Finally, the linear relation between PWM pulse width and the thrust of one propeller is found:

$$T = nPWM + q \tag{A.1}$$

where $n = 0.010757\ \frac{N}{\mu s}$ and $q = -10.784300\ N$. Therefore, the n value obtained from the interpolation of the experimental data can be substituted in Eq. 5.16 to convert a PWM pulse width variation in a thrust variation, as explained in section 5.2.4.

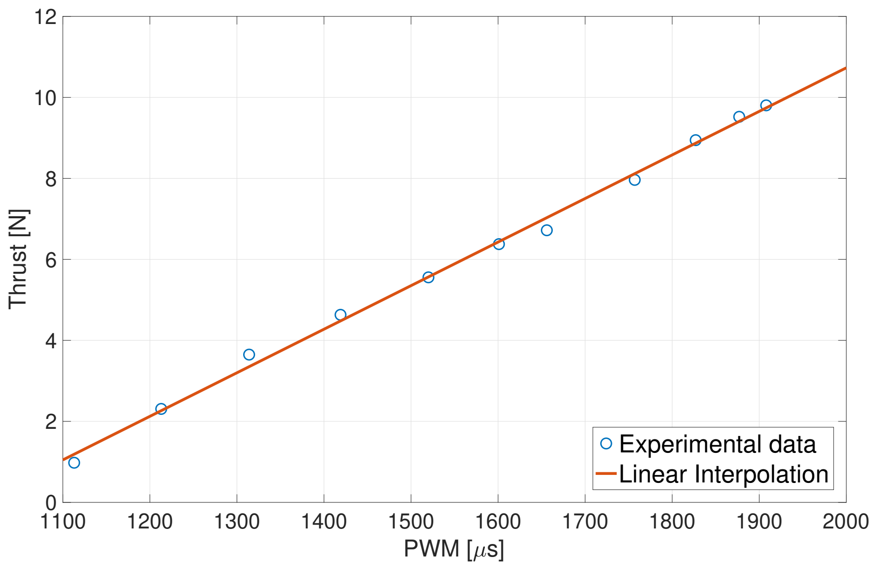


Figure A.3: Interpolation of experimental data

A.2 Moments of inertia measurements

The methodology described in [46] is applied to measure the quadcopter's moments of inertia with respect to x_b and y_b axes, that are respectively identified by I_x and I_y. I_z is not evaluated in these tests because the model aims to give a model-based design approach only for pitch and roll dynamics, which are actually the most critical ones. Since other measurements should be taken for yaw dynamics, the yaw control design is found experimentally by flight tests. However, in future works, physical quantities related to yaw dynamics will be measured too.

The test bench consists of a pendulum where the drone is built on at the extremity (Fig. A.4). By referring to [46], the kinetic and potential energy of the pendulum can be found, and applying the Lagrangian equation the moment of inertia of a body built on the extremity of the pendulum can be found by Eq. A.2:

$$I = \frac{T^2}{4\pi^2}\left(m_1 g\frac{l_1}{2} + mg(l_1 + d)\right) - \frac{m_1 l_1^2}{4} - m\left(l_1 + d\right)^2 - I_{rod} \tag{A.2}$$

where I is the moment of inertia of the drone, T is the period of oscillation, m_1 is the rod mass, l_1 is the distance between the joint and the Center of Gravity (CoG) of the rod, d is the distance between the CoG of the bar and the CoG of the drone, and I_{rod} is the rod's moment of inertia with respect to the rotation axis of the pendulum. The period of oscillation is measured by changing the orientation of the quadcopter: to

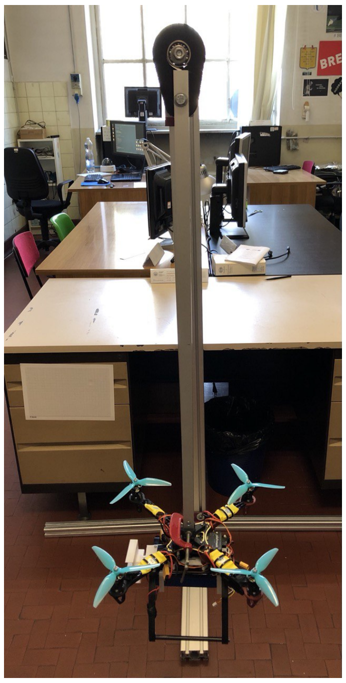


Figure A.4: Pendulum used to measure I_x and I_y

measure I_x the x_b axis of the drone has to be perpendicular to the rod, and the same can be said for I_y and I_z. The measured periods are: $T_{Ix} = 1.880\ s$ for I_x and $T_{Iy} = 1.876\ s$ for I_y. By using Eq. A.2, the drone's moments of inertia are found: $I_x = 0.0231\ Kgm^2$ and $I_x = 0.0282\ Kgm^2$. The small difference between the two moments of inertia is caused by the asymmetry of the drone: in fact, the battery and some hardware devices built on the drone increase I_y.

Appendix B

Controller parameters

In this appendix controller parameters for PID and LQR design are shown. Although numerical methods exist to determine PID and LQR parameters, in this work a trial and error approach is preferred. In fact, the simulation model used to preliminary tune the controller parameters have been validated after performing flight tests. It is worth noting that, as explained in previous chapters, this thesis doesn't follow a model-based approach. An experimental approach has been preferred and the simulation model is developed just to identify approximately the controller parameters to use during flight.

B.1 PID design

The PID controller is tuned by changing the value of the proportional K_P, the derivative K_D, and the integrative component K_I. In Table B.1, the effect of each parameter on the response is shown: referring to this table, the PID parameters are identified following a trial and error approach.

The PID controller is tested only on the simulation model developed in section 5. The parameters used to obtain results shown in section 5.5 are collected in Table B.2 Roll and pitch parameters are different because the moments of inertia I_x and I_y

	Rise Time	Overshoot	Settling Time	Steady State Error	Stability
$K_P \uparrow$	decrease	increase	small increase	decrease	degrade
$K_I \uparrow$	small decrease	increase	increase	large decrease	degrade
$K_D \uparrow$	small decrease	decrease	decrease	minor change	improve

Table B.1: How PID parameters affect the plant dynamics

are different as well. The derivative component K_D of the pitch dynamics is higher than roll to damp more the pitch response: in fact, because of the higher moment of inertia due to the battery, the pitch response would show an overshoot with $K_D = 30$. It should be noted that the integrative component K_I is zero for all angles. In fact,

	Roll	**Pitch**	**Yaw**
K_P	70	70	130
K_D	30	34	0
K_I	0	0	0

Table B.2: PID parameters used in the simulation model

the K_I component is used to have a zero steady-state error, but in the simulation model this can be obtained even with $K_I = 0$. Therefore, it is more correct to say that a PD controller is used for roll and pitch dynamics and a P controller for the yaw. Indeed, a proportional controller for the yaw is enough to obtain a fast response without overshoot and a zero steady-state error.

B.2 LQR design

The LQR controller is tuned by adjusting the value of $\boldsymbol{R}$ and $\boldsymbol{Q}$ matrices. In literature, some methods to identify R and Q matrices have been studied. However, as done for the PID controller, a trial and error approach is followed. In the following table, the LQR parameters used for simulation results, and flight tests are shown. With *diag*[...] a diagonal matrix is defined, and the values inside the square parenthesis are the ones of the main diagonal of the matrix. The controller parameters tested on the simulation

Test	**Q**	**R**
Simulation model	$diag\,[0.05, 0.07, 0.07, 0.001, 0.001, 0.001]$	$diag\,[1, 1, 1]\,10^{-6}$
Flight Test 1	$diag\,[0.05, 0.07, 0.1, 0.001, 0.001, 0.1]$	$diag\,[1, 1, 1]\,10^{-6}$
Flight Test 2	$diag\,[0.08, 0.07, 0.1, 0.001, 0.001, 0.1]$	$diag\,[1, 1, 1]\,10^{-6}$

Table B.3: LQR parameters

model are very close with the ones used during flight tests to obtain results shown in section 7.2, beyond the third and sixth components of the Q matrix. This can be explained considering that the contrast torque provided by motors is not experimentally found but it is assumed from similar works. For this reason, the simulation model cannot be reliable on yaw response, thus the LQR parameters for the yaw control are identified directly during flight tests.

Bibliography

[1] Consortiq, ed. *A Not-So-Short History of Unmanned Aerial Vehicles (UAV)*. URL: https://consortiq.com/short-history-unmanned-aerial-vehicles-uavs/.

[2] The Bureau of Investigative Journalism Search The Bureau Website, ed. *History of drone warfare*. URL: https://www.thebureauinvestigates.com/explainers/history-of-drone-warfare.

[3] Patricia Zambrano-Robledo, Luis Amezquita-Brooks, and Octavio Garcia-Salazar 1 Alan G. Escobar-Ruiz, Omar Lopez-Botello, Luis Reyes-Osorio, "Conceptual Design of an Unmanned Fixed-Wing Aerial Vehicle Based on Alternative Energy". In: *Hindawi, International Journal of Aerospace Engineering* (2019), p. 1. URL: https://doi.org/10.1155/2019/8104927.

[4] Commercial drone experts Coptrz, ed. *A guide to fixed wing drones*. URL: https://www.coptrz.com/a-guide-to-fixed-wing-drones/.

[5] Gaurav Singhal, Babankumar Bansod, Lini Mathew. *Unmanned Aerial Vehicle classification, Applications and challenges: A Review*. Report. 2018.

[6] M. A. Boon, A. P. Drijfhout, S. Tesfamichael. *Comparision of a fixed-wing and multi-rotor UAV for environmental mapping applications: a case study*. Report. 2018.

[7] Maite Arteta Fernaéndez. "Assembly, Modeling, Simulation and Control of a Quadcopter for Application to Solar Farm Inspection". In: (2015), p. 7.

[8] Anna Ga´szczak, Toby P.Breckon, Jiwan Han. *Real-time People and Vehicle Detection from UAV Imagery*. Report. Cranfield University, School of Engineering, United Kingdom.

[9] Eduard Semsch, Michal Jakob and others. *Autonomous UAV Surveillance in Complex Urban Environments*. Faculty of Electrical Engineering, Czech Technical University in Prague, 2009. URL: https://www.researchgate.net/publication/221156280.

[10] Simran Brar, Ralph Rabbat, Vishal Raithatha, George Runcie, Andrew Yu. *Drones for Deliveries*. Berkeley, University of California, 2015.

[11] Mustapha Bekhti, Nadjib Achir, Khaled Boussetta and Marwen Abdennebi. “Drone Package Delivery: A Heuristic approach for UAVs path planning and tracking”. In: *EAI Endorsed Transactions* (2017).

[12] Timothy R. Gulden. *The Energy Implications of Drones for Package Delivery*. RAND Corporation, Santa Monica, Calif., 2017.

[13] Leonardo Company, ed. *The sky is NOT the limit, the video of Sumeri: Sets Sail!* 2021. URL: https://www.leonardocompany.com/en/news-and-stories-detail/-/detail/the-sky-is-not-the-limit-the-video-of-sumeri-sets-sail-.

[14] Torino City Lab, ed. *Sumeri: Si Salpa*. 2021. URL: https://www.torinocitylab.it/it/sumeri-si-salpa.

[15] *Wingtra, Official Website*. URL: https://wingtra.com/drone-mapping-applications/surveying-gis/.

[16] Javad Shahmoradi, Elaheh Talebi, Pedram Roghanchi and Mostafa Hassanalian. “A Comprehensive Review of Applications of Drone Technology in the Mining Industry”. In: *MDPI* (2020).

[17] Samuel C. Hassler and Fulya Baysal-Gurel. “Unmanned Aircraft System (UAS) Technology and Applications in Agriculture”. In: *MDPI* (2019).

[18] Pasquale Daponte, Luca De Vito, Luigi Glielmo, Luigi Iannelli, Davide Liuzza, Francesco Picariello and Giuseppe Silano. “A review on the use of drones for precision agriculture”. In: *IOP Science* (2019). URL: https://iopscience.iop.org/article/10.1088/1755-1315/275/1/012022.

[19] Laurent Probst, Bertrand Pedersen, Lauriane Dakkak-Arnoux. *Drones In Agricolture*. Report. 2018.

[20] Julien Fleureau, Fabien Servant, Franois-Louis Tariolle, Philippe Guillotel. *Directing Cinematographic Drones*. Report. Technicolor, Rennes, France, 2017.

[21] J. Fleureau, F. L. Tariolle and P. Guillotel Q. Galvanet. *Automated Cinematography with Unmanned Aerial Vehicles*. Report. Technicolor, France, 2017.

[22] *DIY Drones*. URL: https://diydrones.com.

[23] EASA European Union Aviation Safety Agency, ed. *Easy Access Rules for Unmanned Aircraft Systems (Regulations (EU) 2019/947 and (EU) 2019/945)*. 2020.

[24] FAA Federal Aviation Administration, ed. *Remote Pilot – Small Unmanned Aircraft Systems Study Guide*. 2016.

[25] Emad Ebeid, Martin Skriver, Kristian Husum, Terkildsen Kjeld Jensen, Ulrik Pagh Schultz. “A Survey of Open-Source UAV Flight Controllers and Flight Simulators”. In: (2018), pp. 1–4. URL: https://www.researchgate.net/publication/325134452_A_Survey_of_Open-Source_UAV_Flight_Controllers_and_Flight_Simulators.

[26] Dario Brescianini, Markus Hehn and Raffaello D'Andrea. "Nonlinear Quadrocopter Attitude Control". In: (2013). URL: https://doi.org/10.3929/ethz-a-009970340.

[27] *Paparazzi, the Free Autopilot.* URL: http://wiki.paparazziuav.org/wiki/Main_Page.

[28] *ardupilot.* URL: https://ardupilot.org/index.php/about.

[29] *TopXGun, OfficialWebsite.* URL: https://www.topxgun.com/en/home.html.

[30] *DJI, Official Website.* URL: https://www.dji.com/it.

[31] IvenSense, ed. *MPU-6000 and MPU-6050 Register Map and Descriptions, Revision 4.2.* Data-sheet. 2013. URL: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf.

[32] Magdi S. Mahmoud. *Advanced Control Methods and Renewable Energy System Integration.* Butterworth-Heinemann, 2016. ISBN: 978-0081017531.

[33] Electricalfundablog, ed. *Pulse Width Modulation (PWM) – Generation, Applications and Advantages.* URL: https://electricalfundablog.com/pulse-width-modulation/.

[34] STMicroelectronics, ed. *RM0008, Reference manual.* Data-sheet. 2018.

[35] J. Sanz Subirana, J.M. Juan Zornoza and M. Hernaéndez-Pajares. *GNSS Data Processing.* ESA, 2013.

[36] u blox, ed. *u-blox 8 / u-blox M8l.* Data-sheet. 2021.

[37] Brian L. Stevens, Frank L. Lewis, Eric N. Johnson. *Aircraft Control and Simulation.* John Wiley and Sons, Ltd, 2016. ISBN: 978-1-118-87099-0.

[38] Anton H.J. de Ruiter, Christopher J. Damaren, James R. Forbes. *Spacecraft Dynamcs and Controls.* John Wiley and Sons, Ltd, 2013. ISBN: 9781118342367.

[39] Karl Johan Astrom. *Control System Design.*

[40] Erik Torstensson. *Comparison of Schemes for Windup Protection.* Master Thesis. Lund University, Department of Automatic Control, 2013.

[41] Components101, ed. *STM32F103C8T6 - Blue Pill Development Board.* URL: https://components101.com/microcontrollers/stm32f103c8t8-blue-pill-development-board.

[42] Robot Store, ed. *MODULO GY-521 MPU-6050.* URL: https://www.robotstore.it/Modulo-GY-521-MPU-6050?gclid=CjwKCAiAmrOBBhA0EiwArn3mfNraCa1CB7P3u8eTVU6O59fh2
3vSdOjti3cnmbe8RoCg6UQAvD_BwE.

[43] ElectronicsComp.com, ed. *Fly Sky FS-i6 6-Channel 2.4 Ghz Transmitter and FS-iA6 Receiver.* URL: https://www.electronicscomp.com/flysky-fs-i6-6-channel-2.4ghz-transmitter-receiver.

[44] BanGood, ed. *Micro Scheda SD TF di Memoria Adattatore Modulo Scudo SPI Micro SD per Arduino.*

[45] RCbenchmark, ed. *Series 1520 Thrust Stand.* 2021. URL: https://www.rcbenchmark.com/pages/series-1520.

[46] M. Romano E. Capello, H. Park, B. Tavora Giorgio Guglieri and. “Modeling and experimental parameter identification of a multicopter via a compound pendulum test rig”. In: *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems,* (2020).

[47] Amir Nasir Babiker Hussein. “Autopilot design for a quadcopter”. Thesis. University Of Khartoum, 2017.