Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering

Hardware/software architecture, code generation and control for multirotor UAVs

Advisor: Prof. Marco LOVERA
Co-Advisor: Dr. Simone PANZA
Dr. Davide INVERNIZZI
Eng. Mattia GIURATO

Thesis by:
Gabriele BRESSAN Matr. 874568

Academic Year 2017–2018

A mio Fratello e ai miei Genitori,
che hanno reso possibile questo traguardo.

Acknowledgments

Durante questo percorso ho avuto la fortuna di essere stato affiancato da persone meravigliose.

Il primo ringraziamento è rivolto al Professor Lovera Marco per avermi dato l'opportunità di svolgere un progetto di mio interesse, per le sue competenze e la sua infinita disponibilità.

Il pensiero più grande va Simone, una persona fantastica che non smetterò mai di ringraziare. Il suo supporto, il suo altruismo e la sua disponibilità sono stati fondamentali per la riuscita di questo progetto e non potrò mai dimenticarlo. Un grazie di cuore anche a Davide che si è dimostrato essere una persona veramente umana, paziente e che ha saputo starmi vicino nei momenti più difficili (gli incubi con il geometrico). Ringrazio anche Mattia e Paolo, punti di riferimento in laboratorio, per il loro altruismo e le loro competenze.

A seguire, tengo a ringraziare tutti i miei compagni di corso, in particolare a Riccardo, senza la quale sarei ancora in triennale, a Simone per essere stato un compagno di avventure e di progetti in questi due anni di magistrale, lo Zio Pietro per avermi aiutato nel momento del bisogno, ad Andrea per le sue grandi abilità nelle materie informatiche e Luca per tutti gli appunti passati.

Un ringraziamento speciale va ad Arianna per avermi incoraggiato nei momenti difficili e per aver sopportato i miei sbalzi di umore di questi 5 lunghi anni. Ringrazio anche i miei compagni di vita, Manuel, Davide, Giona, Giacomo, Alice. Grazie a tutti gli amici che hanno creduto in me.

Il ringraziamento più sentito va a mio fratello Emanuele per avermi sempre guidato e sostenuto nelle scelte, ed infine a mia mamma Elisabetta e a mio papà Giorgio per avermi confortato nei momenti più difficili, per tutti i pranzi preparati e per tutti i passaggi in stazione.

Abstract

Nowadays, Unmanned Aerial Vehicles (UAVs), commonly known as drones, are more and more popular thanks to the wide range of applications in which they can be employed. This category of aircraft finds out every day new application areas, from entertainment to professional purposes, to military missions.

There are many projects aimed to the design of multirotors. In particular the contribution of this thesis consists in the enhancement of hardware and software features that can support the research and the development processes on UAV navigation and control systems in an academic environment. The starting point of this project was ANT-1, a drone designed and developed at the Aerospace System and Control Laboratory (ASCL) at Politecnico di Milano.

Starting from a performance review of the ANT-1 hardware and software, which allowed to find out the components that can be significantly improved. The aim of this thesis is twofold: to create an educational drone that has the possibility to implement and support advanced control laws; to demonstrate such capability to test user-defined advanced control laws.

Currently, the implementation of an attitude control law in the PX4 firmware is an expensive operation in terms of time: for this reason one of the main challenges was to develop an application that allows to import an attitude control law implemented in Simulink, into PX4 firmware. Once the code generation tool has been validated through a PID attitude control law, two advanced control laws have been validated in flight: the non linear geometric and the adaptive controllers.

Sommario

Al giorno d'oggi gli Unmanned Aerial Vehicles (UAVs), comunemente noti con il termine di droni, sono sempre più diffusi grazie alle notevoli applicazioni in cui possono essere impiegati. L'utilizzo di questa categoria di velivoli trova di giorno in giorno nuovi campi di applicazione, che spaziano dall'intrattenimento a scopi professionali, fino a missioni di tipo militare.

Sono molti i progetti volti alla realizzazione di multirotori, in particolare il contributo di questa tesi è mirato al potenziamento di funzionalità hardware e software che ne possano supportare il processo di ricerca e sviluppo. Il punto di partenza di questo progetto è stato ANT-1, un drone progettato e sviluppato presso l'Aerospace System and Control Laboratory (ASCL) del Politecnico di Milano.

Lo scopo della tesi è quello di, partendo da un'analisi delle prestazioni hardware di ANT-1, che ha permesso di valutare quali componenti abbiano margine di miglioramento più significativo, creare un drone didattico che abbia la possibilità di implementare e supportare leggi di controllo avanzato.

Attualmente, l'implementazione di una legge di controllo di asseto nel firmware PX4, risulta essere un'operazione costosa in termini di tempo: per questo motivo una delle principali sfide è stata quella di sviluppare un'applicazione che permetta di importare in PX4 una legge di controllo implementata in Simulink. Una volta aver validato il tool di generazione di codice, tramite una legge di controllo di assetto PID, sono state validate in volo due leggi di controllo avanzato: il controllore di assetto geometrico nonlineare e il controllore adattativo.

In conclusione, grazie ai miglioramenti hardware e dell'infrastruttura software, che permette l'integrazione di leggi di controllo definite dall'utente, è stato possibile contribuire allo sviluppo di un drone che si adatta perfettamente allo scopo didattico e di ricerca.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A multicopter is an aerial vehicle whose motion is controlled by speeding or slowing multiple downward-thrusting propeller units. Multicopters are characterized by very fast attitude dynamics and require a Flight Control Unit (FCU) to control the angular speed of each motor, since a human pilot would not capable to control four motors simultaneously. In addition to the FCU, the simplest kind of multirotor is composed by four brushless motors, four Electronic Speed Controls (ESCs) to regulate the speed of brushless motors and four propellers to generate thrust (see [1]).

In order to balance the generated torques, two opposite motors rotate in clockwise while the other two rotate in counterclocwise. The roll and pitch attitude can be controlled by modifying the thrust obtained by speeding two rotors on one side and slowing down the other two. The quadcopter's yaw attitude can be instead controlled by speeding up the motors that are diagonally and slowing down the other two.

There are several types of multirotor aircraft: tricopter, quadcopter, hexacopter and octocopter are used to refer to 3, 4, 6 and 8-rotor helicopters, respectively. The multirotor with more than three rotors can also have two different configurations: the "X" and the "+" configurations. For this thesis purposes a "X" configuration quadcopter is considered.

Due to the high mechanical complexity related to main and tail rotors, multirotors are preferred over helicopters and this is the reason why multirotors are more attractive.

Currently one of the trends for drone manufacturers is to build smaller and lighter aerial vehicle for different reasons: to comply with weight limits imposed by the Italian Civil Aviation Authority (ENAC) and to reduce the damage in case of crashes. Multirotors with reduced mass and geometric dimension, belong to the category of Micro Aerial Vehicles (MAV). A MAV example is the ANT-1 drone, which has inter-axis of 170 mm. The design of vehicles belonging to the MAV category was possible thanks to the strong technological development of smartphones, which allowed to design smaller and smaller integrated circuits with

accelerometers, magnetometer and gyroscopes.

Nowadays, multicopters can operate in different application areas: videography and photography, disaster response, surveillance, security and mapping. Each of these disciplines requires drones with artificial intelligence algorithms and control laws developed and improved every year.

Starting from a performance review of ANT-1 hardware, which allowed to find out the components that can be significantly improved, the purpose of this thesis it to design an educational drone that has the possibility to implement, test and support advanced control laws. The drones designed for research purposes require hardware components with some requirements: low cost, small size, low power consumption. In addition the companion computer (mini pc) mounted on board, must have good performance to ensure the development of future software features. From a software point of view, one of the challenges was to develop an application that allows to import an attitude control law implemented in Simulink, into PX4 firmware. Once the code generation tool has been validated through a PID attitude control law, two advanced control laws have been validated in flight: the nonlinear geometric and the adaptive controllers.

Thesis structure

The thesis is organized as follows:

- Chapter 1 is dedicated to explain what a multirotor is, how it works and how it is composed in terms of electrical and mechanical components.
- Chapter 2 will present the objectives and the state of the art of drones used for educational and research purposes.
- Starting from an already existing drone called ANT-1, Chapter 3 will analyze and discuss all the hardware requirements necessary to improve the MAV performances.
- Chapter 4 is dedicated to explain how the original PX4 firmware was modified in such a way to include an attitude or position control law implemented in Simulink.
- Chapter 5 introduce the attitude control of multirotor UAVs and it also explain the PID regulator implemented in PX4.
- Chapter 6 discuss about geometric and adaptive attitude control laws theory and also the Simulink implementation issues.
- Chapter 7 will shown the experiments result of PID, geometric and adaptive control laws terms of performances and FCU resources consumption.
- Finally the goals achieved and the conclusion of this work is summarized in Chapter 8.

Chapter 2

Objectives and state of the art

Drones have changed how we approach almost every discipline from video or photography to aerial surveillance, from measurement to mapping. This has become more and more evident in the last few years and for this reason it is necessary to research and improve control laws. The starting point of this project was ANT-1, a drone designed and developed at the Aerospace System and Control Laboratory (ASCL) at Politecnico di Milano. The aim of the thesis is, starting from a performance analysis of ANT-1 hardware, which allowed to find out the components that can be significantly improved, to create an educational drone that has the possibility to implement and support advanced control laws. The design of multirotor used for educational and research purposes must take into account some requirements. The hardware components must be easily available on the market, low cost, low weight, small size and in addition it is necessary to find the trade off between performance and energy consumption of the components. Also the use of an open source firmware is a requirement, because it allows the implementation of new features and the modification of existing ones. The best candidate was the PX4 firmware, fully compatible with the PixFalcon FCU and it is constantly under development by an active worldwide community. One of the objectives is develop an application that allows to import an attitude control law implemented in Simulink, into PX4 firmware. Once the code generation tool has been validated through a PID attitude control law, two advanced control laws have been validated in flight: the non linear geometric and the adaptive controllers.

For this type of drones the market offers few solutions. Figure 2.1 shows the DJI Ryze Tello EDU, the educational version of DJI Ryze Tello and powered by DJI and Intel. Thanks to the official Software Development Kit (SDK) it is possible to send commands an receive drone status exploiting Wi-Fi connection. Ryze Tello EDU integrates different sensors such as a front camera and a pair of infrared sensors used for obstacle detection and ground measurement. With these sensors it is possible to realize more functions such as object recognition, tracking, 3D reconstruction through programming and computer vision. From an educational point of view, using artificial intelligence algorithms would be possible

to improve the stability of its original control law but this is a limitation when a new control law needs to be tested (see [2]).

Figure 2.1: Ryze Tello EDU drone

Another solution available on the market is the Parrot Education drone project (Figure 2.2) which was launched to support the drone revolution occurring in academic institutions. The Parrot project include a drone and also supports different programming languages such as Javascript, Pyhton and Swift (see [3]). The interesting feature of this project is the collaboration with MathWorks. In fact, the Simulink Support Package for Parrot was developed and it permits to design and build flight control algorithms for Parrot mini drones. Using Simulink Support Package it is possible to access to the onboard sensors such as, ultrasonic, accelerometer, gyroscope, air pressure sensors, frontal camera. Unlike the DJI Ryze Tello EDU, with the Parrot Education project it is also possible to define a new control law in Simulink and test it on Parrot drone (see [4]).

Figure 2.2: Parrot education drone

An alternative solution proposed is the Quanser Autonomous Vehicle Research Studio, consisting of QDrone (Figure 2.3), a ground station and also vision and safety equipment. The disadvantage of the proposed solution is the "closed" Quanser environment. As mentioned before the purpose of this thesis consist of hardware and software evaluation to create a drone used for teaching and research. For this reason the main idea is to find and evaluate a performing

Figure 2.3: QDrone by Quanser

hardware easily available on the market and an open source firmware in such a way to modify it, if necessary.

Since none of the discussed solutions satisfies all the requirements, it was decided to reach the goal of this thesis starting from a quadcopter developed in the Aerospace System and Control Laboratory (ASCL) called ANT-1. ANT-1 belongs to the MAV category and it is equipped with hardware (companion and FCU) low cost and easily to find on market. As open source firmware was decided to use PX4 which is the most compatible with the PixFalcon FCU.
Similarly to the Parrot firmware, also for PX4 there exists a MathWorks tool name Pixhawk Pilot Support Package (PSP), in which it is possible to generate a C/C++ code from Simulink model specifically tailored for the PixFalcon FCU using the Pixhawk Toolchain (see [5]). Using PSP it is possible to customize algorithms that leverage onboard sensor data and other calculations at runtime. When this thesis project started the available PSP version was 2.1 and it could be used with PX4 version 1.3.4 (currently the last version is 1.8), but now the PSP version 3.04 was released but the limitation is that MAVlink protocol is not supported (the serial communication between companion and FCU can not happen).

Chapter 3

Hardware selection and evaluation

The aim of this chapter is to describe the system architecture of the ANT-1 drone and what the hardware problems are in already existing drone called ANT-1. Indeed, the challenge is to optimize it and to create another drone for educational and research purposes.

During several flight sessions of ANT-1 we found some problems like slow responsiveness using the SSH communication protocol but also time delay to receive position information coming from the motion capture cameras.

Once identified the problems will be necessary find on the market the best alternative in terms of performance, weight, dimensions and consumption.

3.1 Preliminary requirements

Nowadays, multirotor platforms are used in several areas and each of these can require different hardware components. For the purpose of this thesis, we need to find and evaluate a hardware for a drone that will be used in research and educational area. In order to choose an optimal hardware configuration, one has to satisfy some requirements:

- Hardware easily available on the market;
- Low cost;
- Supporting all the necessary features (i.e., the presence of a Wi-Fi module, necessary to communicate with the drone remotely);
- Weight less than 10 g;
- Small size and square shape;
- No more than 5 V and 2 A consumption.

3.2 Description of the system

The already existing system can be divided in three different parts:

- ANT-1 drone;
- Remote controller and receiver
- Motion capture cameras system.
- Ground Station (GS);

3.2.1 ANT-1 drone

During the experimental research in the drone field, it often happens to collect data during an indoor flight session and it could be complicated and unsafe when drones of big dimensions are used. For this reason, it was decided to create a new Micro Aerial Vehicle (MAV) called ANT-1 (Figure 3.1), made with low cost, off-the-shelf components. Another interesting feature, in addition to the small size, it is the suitability for the Italian Civil Aviation Authority (ENAC), which allows outdoor flight without restrictions only for remotely piloted vehicles that weight less than 300 g. Since ANT-1 belongs to the class of MAV, it respects some requirements:

- Maximum take-off weight: less than 300 g;
- Flight time: at least 10 minutes.
- Reduced geometric dimension: an inter axis smaller than 200 mm.

From a components point of view, ANT-1 is composed by: 4 motors, 4 Electronic Speed Controls (ESCs), 4 propellers, 1 Lithium-Ion Polymer battery (LiPo), 1 PixFalcon Flight Control Unit (FCU) and 1 Raspberry Pi Zero W [**?**] as companion.

In the next sections only FCU and companion will be analysed.

Flight Control Unit

The FCU represented in Figure 3.2 is the core of the quadrotor and it's the part of the drone system dedicated to control the RPM (Revolutions per minute) of each motor, since the human is not capable to control four motors simultaneously to perfectly balance a quadcopter in the air. For the FCU it has been decided to use an electronic board called PixFalcon [**?**] (produced by Holybro) which also employs sensors, such as 3-axes accelerometer, 3-axes gyroscope, magnetometer and barometer, to supplement its calculations.
The PixFalcon features are:

Figure 3.1: ANT-1

- Dimensions: 38×42×12 mm
- Weight: 15.8 g;
- Chipset: based on 32 bit STM32F427 Cortex M4 core 168 MHz CPU and 256kB SRAM.

The first version of the PixFalcon board was created in 2008 by a team of students at the ETH university in Zurich and was named PixHawk. During the same time the team also created the MAVLink [**?**] protocol (used for serial communication between FCU and companion), the PX4 firmware [**?**] (which is the firmware supported by PixFalcon) and QGroundcontrol [**?**] (software used for PixFalcon configuration and real time information). All these components are today's most used standards for flight control hardware and autopilot software in the multirotor Unmanned Aerial Vehicle (UAV) industry.

Companion computer

The companion computer (Figure 3.1) is the part of the drone system used to interface and communicate with PX4 on a PixFalcon using the MAVlink protocol. It enables a broad range of functionality such as the possibility to execute processes that require heavy CPU load. During the flight sessions on a closed arena, the companion computer is used to receive drone position information from the Ground Station, which is connected to a motion capture system, and to send the

Figure 3.2: PixFalcon FCU

information received to the FCU through the serial communication. When ANT-1 was designed, the Raspberry Pi Zero W companion was chosen because it was a best market choice, in terms of hardware performance (Table 3.1).

An advantage of Raspberry Pi Zero W is the Raspbian OS [**?**], which is optimized for its hardware and has reached a stable version thanks to the large community of developers and supporters. From a software point of view Robot Operating System (ROS) [**?**] and MAVproxy are installed into the companion:

- ROS: it is used to communicate with Ground Station with ROS messages. In particular Mavros [**?**], a ROS package provides communication driver for PixFalcon autopilot with MAVLink communication protocol. Additionally it provides UDP MAVLink bridge for ground stations control (*e.g.*, QGroundControl);
- MAVproxy: the intent of this package is for a minimalist, portable and extendable Ground Control Station (GCS) for any UAV supporting the MAVLink protocol. Thanks to Mavproxy it is possible to control the PixFalcon from GCS by the MAVLink protocol.

3.2.2 Remote controller and receiver

The Remote Controller (RC) is the main method to control the attitude of the quadrotor. Even if the drone is often controlled through the GS, the RC is used mainly for arm and disarm the motors for safety reasons.
For the experiment described in the next pages, two different radios plus two receivers have been used:

- FrSky Taranis X9D Plus 2.4 GHz ACCST with R-XSR receiver (Figure 3.3a);

Name	Raspberry Pi Zero W
CPU	ARM11 running at 1GHz
RAM	512 MB
Wireless	2.4 GHz 802.11n
Dimensions	$65mm \times 30$ mm
Weight	9 g
Power	5 V - 1 A

Table 3.1: Raspberry Pi Zero W features

- Quanum i8 2.4 GHz with IA8 receiver (Figure 3.3b).

(a) Taranis X9D radio

(b) Quanum i8 radio

Figure 3.3: Radios equipment

3.2.3 Motion Capture system (Mo-Cap)

Motion Capture is a system composed by 16 Infra-Red (IR) sensitive Opti-Track [**?**] cameras (Figure 3.4a) with incorporate IR flood lights. The cameras are mounted on a closed arena and they are fixed at calibrated positions and orientation so that the measurement subject is within the field of view of multiple cameras.

Thanks to markers sensitive to infrared light (Figure 3.4b) mounted on top of the drone, it is possible to make it trackable and define its position into 3D space. Each drone mounts a different marker layout to be uniquely identified when more drones fly at the same time.

To control the motion capture system, the Motive software (installed in GS) is used. Motive is a software platform designed to control motion capture systems for various tracking applications. It not only allows the user to calibrate

and configure the system, but it also provides interfaces for both capturing and processing of 3D data. The captured data can be both recorded or live-streamed to other pipelines.

The frequency with which the position information are sent to the drone (cameras rate), can be changed from a minimum of 30 Hz up to a maximum of 240 Hz through Motive software. The accuracy of the position estimated by the drone depends on the cameras frequency selected.

(a) Infrared camera (b) Infrared markers

Figure 3.4: Motion Capture System

3.2.4 Ground Station

The Ground Station (GS) is the part of the system dedicated to send and retrieve information from the quadcopter during a flight session. The main task of the GS is to read the position information coming from motion capture system and send them to the drone, but it is also possible to define a trajectory to be followed using specific waypoints and view telemetry data in real time.

The GS architecture is divided into two different OS's: the main OS is Windows 10 [**?**] in which Motive is installed. The second OS is Linux OS (more precisely Ubuntu 16.04 [**?**]) which is installed on a virtual machine and is used to execute ROS. The GS architectural division was necessarily done because Motive is a Windows software while ROS integrates better in a Linux environment.

When we need to do a flight session which requires the position information from the motion capture system, some steps are required on GS:

- turn on the motion cameras and calibrate them using Motive software;
- boot the Linux OS on virtual machine;
- launch roscore using `roscore` command on the Ubuntu terminal window;
- launch a ROS motion cameras node using `roslaunch flyart_mocap mocap.launch` command in another Ubuntu terminal window.

Roscore is a collection of nodes and programs that are pre-requisites of a ROS-based system. It is necessary to have a roscore running in order for ROS nodes to communicate.
The mocap ROS node executed on a GS is necessary to publish the real-time position information read from the motion cameras for sending them to the drone which is subscribed to the same ROS topic.

3.3 System analysis

Each time a new flight session starts, some manual operations are required to make ANT-1 ready to fly. In particular, these operations are necessary when the drone has to fly in altitude or position hold mode, since both modalities require the position information coming from the mocap ROS node running on GS.

The position listener ROS node, which receives the position information coming from mocap, needs to be executed on ANT-1. Once the battery is connected to the drone and the Raspberry Pi Zero W has finished the boot, it is possible to connect to the companion using the SSH protocol:

```
ssh pi@<RASPBERRY_IP_ADDRESS>
```

and execute the listener ROS node with the following command line:

```
roslaunch flyart_mavros ant1_px4_serial.launch.
```

For the creation of a new drone some problems must first be solved in the current hardware on ANT-1:

- when ANT-1 is turned on and the companion finishes the boot load, the Raspberry Pi Zero W is very slow to respond to SSH requests made by a GS;
- position set points sent to ANT-1 using MATLAB [**?**] are sometimes lost;
- when ANT-1 flies in altitude or position hold mode, some timeout warning messages appear on the ANT-1 terminal, and the drone does not behave as expected. The PX4 firmware starts to integrate two times the information coming from the accelerometer when they are not received, causing bad flight behaviour.

The timeout message (Figure 3.5) appears when the time difference between two consecutive position information packages received from FCU is greater than 0.2 seconds.

Below are reported some lines of code coming from the PX4 firmware, in which is possible to see when a timeout message is generated.

```
void BlockLocalPositionEstimator::mocapCheckTimeout()
{
  if (_timeStamp - _time_last_mocap > MOCAP_TIMEOUT) {
    if (!(_sensorTimeout & SENSOR_MOCAP)) {
      _sensorTimeout |= SENSOR_MOCAP;
      _mocapStats.reset();
      mavlink_and_console_log_info(&mavlink_log_pub, "[lpe] mocap timeout ");
    }
  }
}
```

Figure 3.5: How timeout is computed in PX4 firmware

3.4 Resources monitor

In order to better understand latencies and timeout problems, two different parts of the system have been monitored as specified in Figure 3.6 :

- time difference between two different packages coming from ROS camera node (Mo-Cap) and received on a PixFalcon;
- resources utilization analysis of the Raspberry Pi Zero W companion.

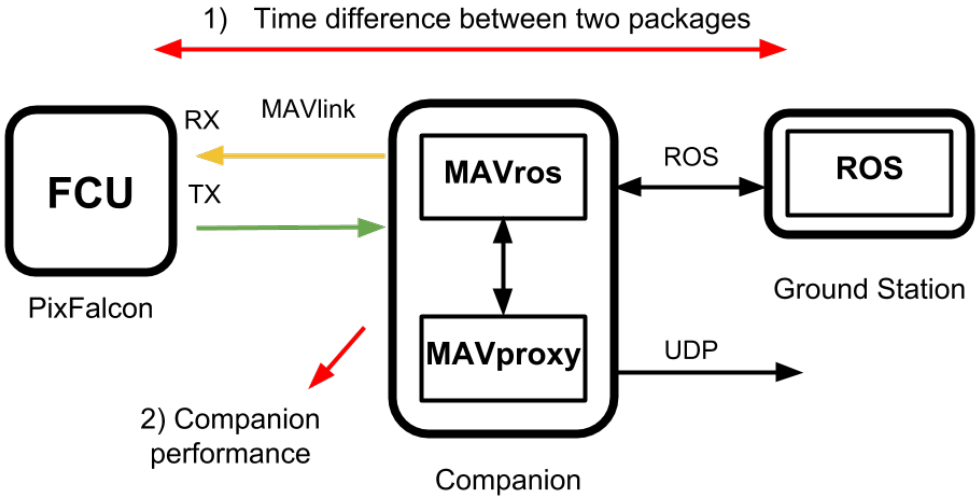


Figure 3.6: System analysis

3.4.1 Time difference between two consecutive packages

Equipment

To better understand why the timeout message appears, the time difference between two consecutive position messages coming from the ground station (in

which the Mo-Cap node runs) and received on the FCU has been measured. For this purpose the PX4 log system was used, because it permits to save all the PX4 topics information (which are running on FCU), in a specified log file. The ATT_POS_MOCAP topic was logged, since it contains the timestamp information about the received position message, in particular there is a field named "timestamp_received".
The setting up of this experiment is composed by two different radio with receivers (as already described in the Section 3.2.2) and two different companion. In particular the Raspberry Pi Zero W and the Intel Edison [**?**] boards are used to verify if the quality of different the Wi-Fi modules can change the experiment results. The two boards are already mounted on two different drones:

- the ANT-1 drone, with the configuration described in the Section 3.2.1;
- the HEXA drone, a hexacopter drone with PixFalcon as FCU and Intel Edison board as companion.

The experiment was composed by three different tests, with the following combination of drone (which contains a specific companion) and RC:

- the first was the ANT-1 drone with FrSky Taranis as radio and R-XSR receiver;
- the second was the HEXA drone with FrSky Taranis as radio and receiver;
- the third was the ANT-1 drone with Quanum i8 radio and IA8 receiver.

Results

In each of the previously defined tests, the drone has been set to be in normal operating conditions (for safety reasons a real flight session has not been performed), but with the RC turned off. After a few seconds the RC was turned on and the results are reported in the following.

The first test was made using the ANT-1 and FrSky Taranis combination which is normally used in the laboratory. The Figure 3.7 clearly shows how the time difference between two consecutive position messages received by FCU changes when the Taranis radio was turned off and on. As mentioned before this increase in time delay was due to the introduction of the Taranis radio signal (when it was turned on), which has the same 2,4 GHz frequency of the Raspberry Pi Zero W Wi-Fi module. For the Wi-Fi communication, the Taranis radio signal was considered as a disturbance and some position information packages were lost.

For the second experiment (Figure 3.8), the idea was to combine the Intel Edison companion and the FrSky Taranis radio. Since the companion was already mounted on a hexacopter (HEXA), the test set up was composed by the HEXA drone and Taranis radio. As before, for the first half of the test, the Taranis radio

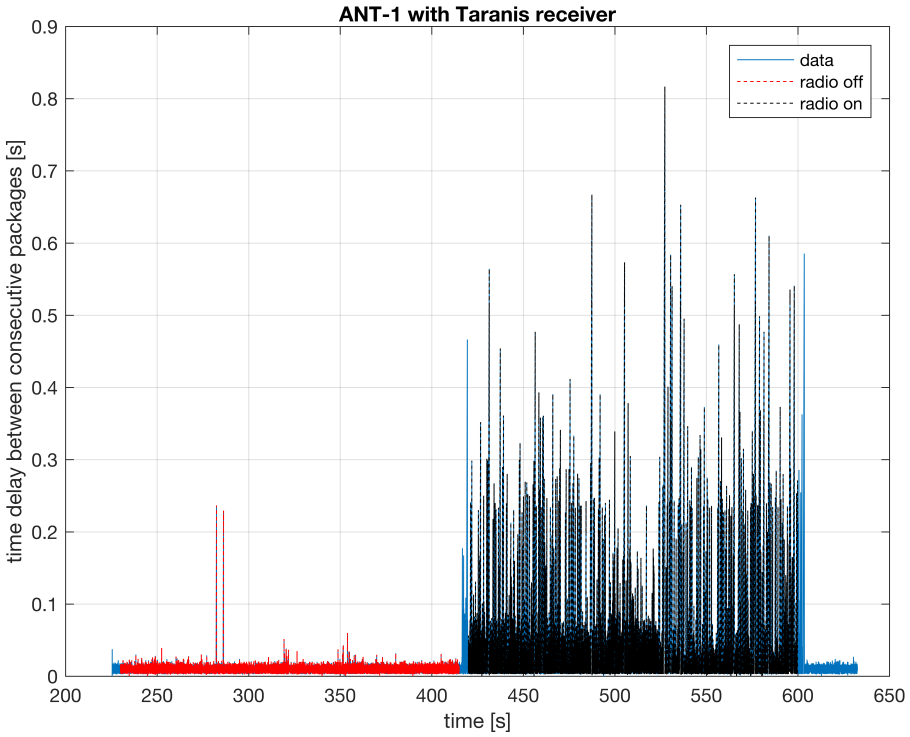


Figure 3.7: ANT-1 with Taranis receiver (R-XSR)

	FrSky Taranis off	FrSky Taranis on
Mean value	0.0102	0.0207
Peak value	0.2365	0.8168
Standard deviation value	0.0045	0.0466

Table 3.2: time delay between consecutive packages using ANT-1 and Taranis receiver

	FrSky Taranis off	FrSky Taranis on
Mean value	0.0103	0.011
Peak value	0.1919	0.2115
Standard deviation value	0.0029	0.0074

Table 3.3: time delay between consecutive packages using HEXA and Taranis receiver

was turned off and then after some seconds it was turned on. The frequency of the Intel Edison Wi-Fi module is 2,4 GHz (as for Raspberry Pi Zero W) but the time delay between two consecutive position messages is lower than the first test. The reason is that, differently form the Raspberry Pi Zero W, the Intel Edison Wi-Fi module uses a dedicated Wi-Fi antenna which improves the signal reception.

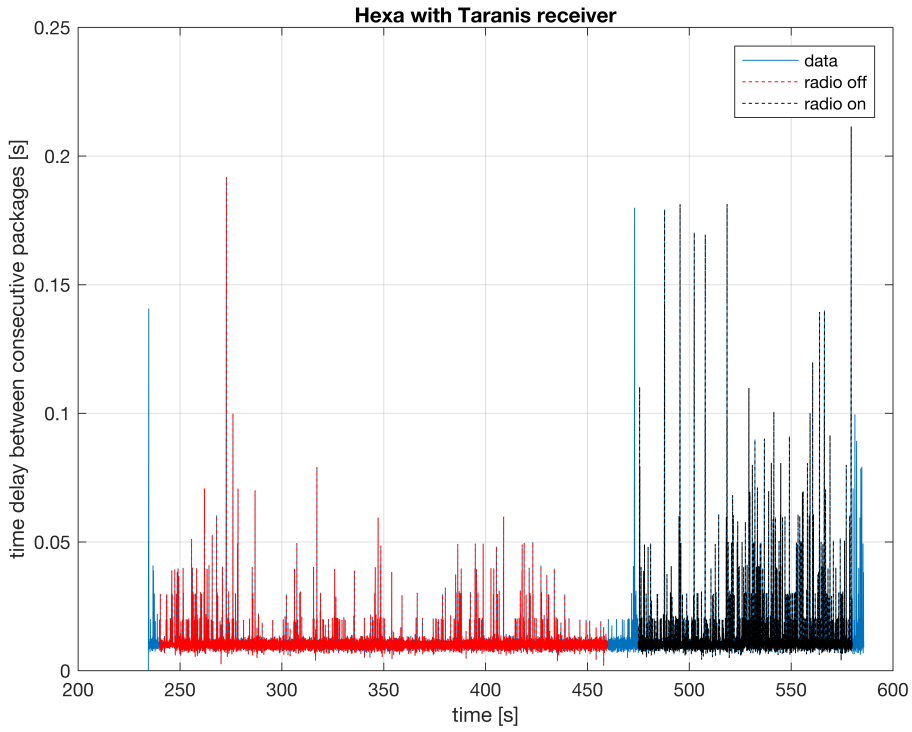


Figure 3.8: Hexa with Taranis receiver (R-XSR)

The last test was made to verify if the time delay between two consecutive position messages is not caused only by the Wi-Fi module quality, but also by the RC communication protocol. For this reason it was decided to combine the Raspberry Pi Zero companion (mounted on the ANT-1 drone) and the Quanum i8 radio. This combination of companion and RC was never used in the laboratory and the results (Figure 3.9) were impressive. Differently from the first test, in which Raspberry Pi Zero W companion and Taranis RC were used, in this case the latency time was drastically reduced also if the Quanum radio frequency is

	Quanum i8 off	Quanum i8 on
Mean value	0.0101	0.0106
Peak value	0.2404	0.2441
Standard deviation value	0.0043	0.0071

Table 3.4: time delay between consecutive packages using ANT-1 and Quanum i8 radio

2,4 GHz.

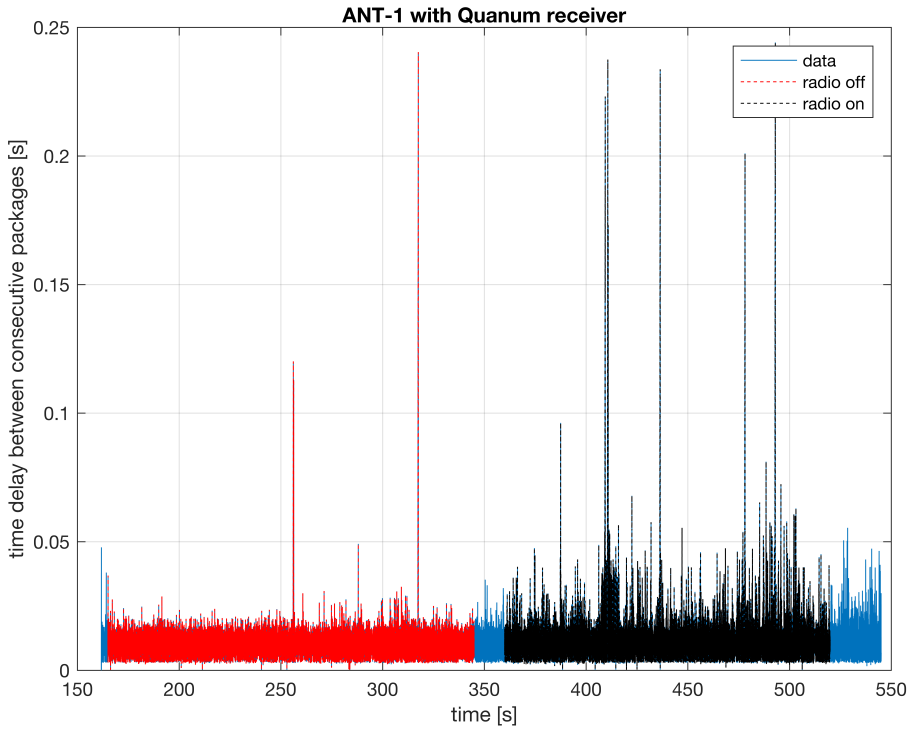


Figure 3.9: ANT-1 with Quanum i8 receiver (IA8)

Another graph is reported (Figure 3.10) which summarises the previous concepts in a single graph to better compare them.

Conclusion

After some analysis on the technical characteristics of both radios (FrSky Taranis and Quanum i8) it was discovered that the two radios use the same 2.4 GHz frequency but they use different communication technology. The FrSky Taranis, unlike the Quanum i8, implements a communication technology called Advanced Continuous Channel Shifting Technology (ACCST), which sends data to the entire 2.4 GHz band creating conflicts with 2.4 GHz Wi-Fi channel and

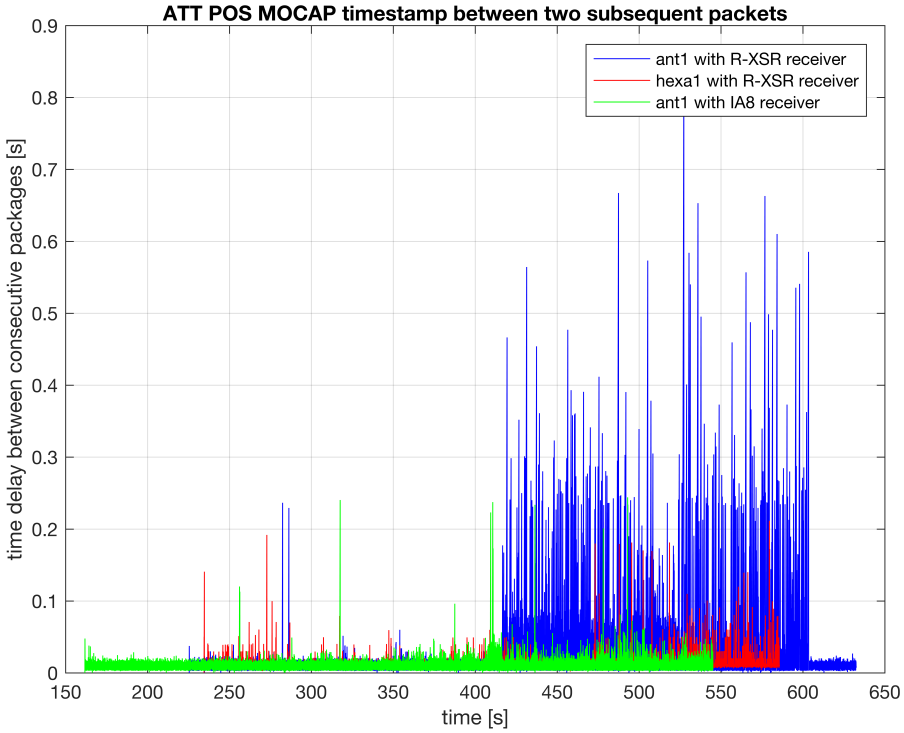


Figure 3.10: Global view

slowing down the reception of the packages.
Since one of the problems that cause latencies and timeout was found. For next experiments the best case will be considered, so Quanum i8 radio and ia8 receiver will be used for ANT-1.

3.4.2 Resources utilization analysis Raspberry Pi Zero

The second part of the system analysis is dedicated to the companion. In particular, it is necessary to analyze the resources utilized by ROS, when it is executed on a Raspberry Pi Zero W in operative conditions.

The benchmark was computed using an open source cross-platform monitoring tool written in Python called Glances [**?**]. Glances is based on a psutil (process and system utilities), which is a library for retrieving information on running processes and system utilization (CPU, RAM, disks, network and sensors). One of the advantages of Glances is the possibility to be executed in different modalities, such as to visualize real time system and processes information on terminal (*e.g.*, htop or top for Linux systems), to export the data in an external database (*e.g.*, Elasticsearch, InfluxDB and Cassandra) and to be able to save all the information in JSON or CSV file format. In our case the best solution, to avoid saturation of the Wi-Fi module and overloading of the CPU, is to save all the system and processes information in a CSV file, that will be later imported

in MATLAB and then analyzed.

Glances set-up

The Glances installation phase is quite simple if Pip Installs Packages (PIP) is used, which is a package manager for Python modules. As said before, Glances is based on Python, so you must have Python (in our case Python 2.7) and PIP installed. After checking the requirements for Glances, you have to install the latest stable version, using the following command line in the terminal:

```
sudo pip install glances
```

and the `sudo` command is necessary to execute the installation with superuser privileges.
Once Glances has been successfully installed, it is possible to test if it works simply writing in the terminal:

```
glances
```

and a list of processes and system information will appear.

Since the system needs to be monitored in operative conditions, the next step is to create a Glances configuration file in which we define the processes name to monitor. The main idea is not only to capture the global system and ROS process but also Glances process information since, as other monitoring tools, Glances can add an overhead in terms of CPU and RAM, so it is necessary to keep it in consideration. The configuration file can be created like a simple empty text file named "glances.conf", and some lines need to be added like the following:

```
[amp_ros]
# Use the default AMP (no dedicated AMP Python script)
# Monitor all the Python scripts
# Alert if more than 20 Python scripts are running
enable=true
regex=.*\/opt\/ros\/kinetic.*
refresh=5


[amp_mavros]
enable=true
regex=.*\/home\/pi\/mavros_catkin_ws\/.*
refresh=5
```

```
[amp_glances]
enable=true
regex=.*\/bin\/glances.*
refresh=5
```

where "mavros" is a ROS package which provides the communication between companion and FCU while the term "ros" is the main ROS process. Since Mavros is a ROS package, in the next pages it will be considered as a ROS process.
At the end, for starting Glances with the configuration file just created, the following command line needs to be used:

```
glances --export-csv <PATH_CSV_FILE> -C <PATH_CONF_FILE> --quiet -t 5
```

where:

- `--export-csv <PATH_CSV_FILE>`: this Glances option is used to export the system and process information in a specified CSV file;
- `-C <PATH_CONF_FILE>`: this option will execute Glances with a configuration defined in a specified config file;
- `--quiet`: this option will execute Glances without showing the system and process information in the terminal (reduce the load of Glances process);
- `-t 5`: defines the Glances sample time of 5 seconds.

Experiment results

After the data acquisition phase using Glances with the ANT-1 drone in real operative conditions, an accurate data analysis was made and in particular, it was identified which Raspberry Pi Zero W hardware components could cause a bottleneck in terms of SSH connection responsiveness and time delay in position messages from motion cameras: CPU, RAM and Wlan0 network interface. The data acquisition phase was made considering four different motion camera rate: camera off (0 Hz), 30 Hz, 100 Hz and 150 Hz. The camera rate is an important value because increasing the frequency in which the position information is sent to the drone, also increases the accuracy of the position estimated by the FCU but also increases the overload of the companion.

One of the Raspberry Pi Zero W performance limitations is the 1 GHz single-core ARM11 CPU, but as it can be easily perceived by intuition, more performing CPU means more consumption. This point must be taken into account when working with drones. As shown in Figure 3.11) three processes were analyzed. The CPU percentage used by Glances remains constant, around 25% in all four cases. This is intuitive since Glances works a fixed sample time of 5 seconds. The

ROS process starts to use the 12% of CPU at 30 HZ, 25% of CPU at 100 HZ and it finishes with 33% at 150 HZ. The growth of ROS load percentage is also quite intuitive, since increasing camera frequency also the number of packages received by ROS and sent to FCU increases.

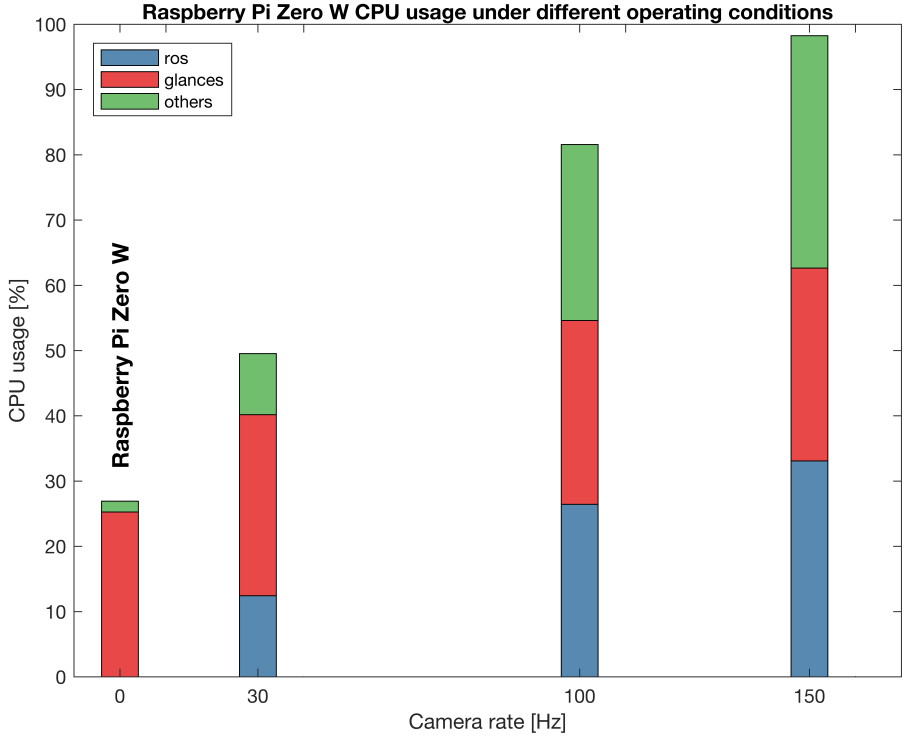


Figure 3.11: CPU used by Raspberry Pi Zero W

Figure 3.11 also shows a general view of remaining processes called as "others". Like in the case before the load increases by increasing the cameras frequency, but in this case mainly due to two system processes: kworker and irq.

Kworker processes (kernel worker processes) perform most of the actual processing for the kernel, especially in cases where there are interrupts. The reason why kworker processes increase the CPU load when the camera rate increases, is that the companion sends data to the FCU (with the same frequency at which the data arrive) by serial communication and the kernel worker processes are occupied managing the data transfer.

Irq processes (interrupt request processes) are processes used to handle interrupts like I/O events. As it happens for kworker processes, increasing the camera rate also increases the frequency with which data are received from the Wi-Fi module and they are sent to FCU, so interrupts increase too.

Another hardware component analysed in operative conditions is the RAM. The percentage of the total measured load of the RAM is 18% when ROS is not executed and 22% in the other three cases. As shown in Figure 3.12, there are

no significant changes in terms of total RAM used. The RAM size of 512MB is sufficient to run ROS without slowing down the processes execution.

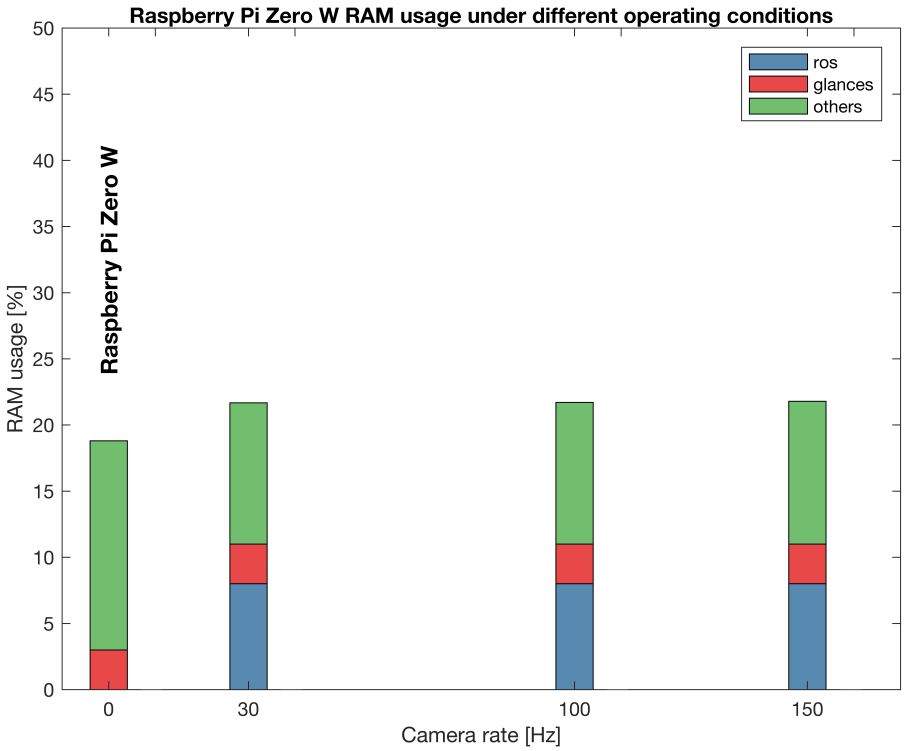


Figure 3.12: RAM used by Raspberry Pi Zero W

The last component monitored using Glances was the network interface Wlan0, used to communicate with the ground station. The results of this analysis, reported in Figure 3.13, were different than expected. The data reception rate (RX) value increases linearly with the camera frequency and this makes sense, since the companion was receiving data (depending on cameras rate), but also the data transmission rate (TX) value increases. After some in-depth analysis it was discovered that when the ROS process starts, it also executes a ROS node called rosout which is used to exchange information about status with others active ROS nodes.

Conclusion

The Raspberry Pi Zero W 1 GHz single-core CPU is not powerful enough to execute ROS processes, receive data from the GS and send them to the FCU at the same time. To solve this problem is necessary to find on the market a more powerful companion in term of CPU since the percentage of RAM used never saturates the maximum value.

Figure 3.13: Wlan0 used by Raspberry Pi Zero W

3.5 Hardware selection

As already discussed in the previous section (Section 3.4), the Raspberry Pi Zero W has some limitations, so the companion needs to be substitute with a new one. The hardware requirements for the companion have already been defined in the Section 3.1 but anyway for the research purpose the most important features are: hardware easily available on the market, weight less than 10 g, small size and no more than 5V and 2A for consumption. After an online research on the most important e-commerce (*e.g.*, Amazon, Bangood, Aliexpress, *etc.*) the best choices have been reported in the Table 3.5.

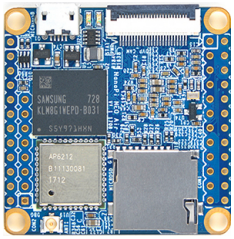
Since the Intel Edison companion was discontinued by Intel, the NanoPi NEO Air [**?**] is the most interesting board which respects all the requirements defined before.

3.6 NanoPi NEO Air companion evaluation

The hight Raspberry Pi Zero W CPU request was satisfied by the new NanoPi NEO Air companion (produced by FRIENDLYARM), which has 1.2 GHz quad-core processor and other features described in the Table 3.6. The NanoPi also has three advantages over the Raspberry: the first is the board shape, in fact

Name	NanoPi NEO Air	NanoPi NEO Plus 2	Banana Pi Zero	Orange Pi Zero
CPU	Quad-core Cortex A7	Quad-core Cortex A53	Quad-core Cortex A7	Quad-core Cortex A7
RAM	512 MB	1 GB	512 MB	512 MB
Wi-Fi	2.4 GHz 802.11 b/g/n	2.4 GHz 802.11 b/g/n	802.11 b/g/n	2.4 GHz 802.11 b/g/n
Dimensions	40 × 40 mm	40 × 52 mm	60 × 30 mm	46 × 48 mm
Weight	7.9 g	10.2 g	9 g	10.9 g
Power	5 V - 2 A	5 V - 2 A	5 V - 2 A	5 V - 2 A
Price	28 $	30 $	20 $	29 $

Table 3.5: Boards comparison



Name CPU	Quad-core Cortex-A7 1.2 GHz
RAM	512 MB
Wireless	2.4 GHz 802.11 b/g/n
Dimensions	40 × 40 mm
Weight	7.9 g
Power	5 V - 2 A
Price	28 $

Table 3.6: NanoPi NEO Air features

it's square layout permit to design a drone frame smaller and more compact than existing one. The second advantage is the possibility to install the OS inside an embedded Multi Media Card (eMMC), which is faster than an OS mounted on an SD card. The last NanoPi advantage is the dedicated Wi-Fi module antenna which permits to reduce the disturbances introduced by the 2.4 GHz remote controller. Once the UbuntuCore 16.04.4 LTS was mounted on the NanoPi eMMC and the ROS environment was compiled and installed on it, the same three tests as for Raspberry were made. Also in this case, we focused on CPU, RAM and Wlan0 network interface performances with different cameras frequency: cameras turned off, 30 Hz, 100 Hz and 150 Hz.
The results of NanoPi performances are reported in the following and are compared with Raspberry performances. In all three tests computed on the new NanoPi NEO Air companion, the results have improved. In particular the percentage of CPU utilization (Figure 3.14) is significantly improved, and now also increasing the motion cameras rate to the maximum value, the CPU utilization percentage is never saturated.

3.6.1 Communication rate between companion and PixFalcon

The last part of the system that has been monitored is the communication rate between the companion and the FCU, and for this part the new NanoPi companion was used. Differently from Raspberry Pi Zero W baud rate, which is 921600 bps, the NanoPi baud rate is 115200 bps and because of this significant difference in terms of baud rate value, we have to know if it can cause a bottleneck during the serial communication. A bottleneck on the serial communication be-

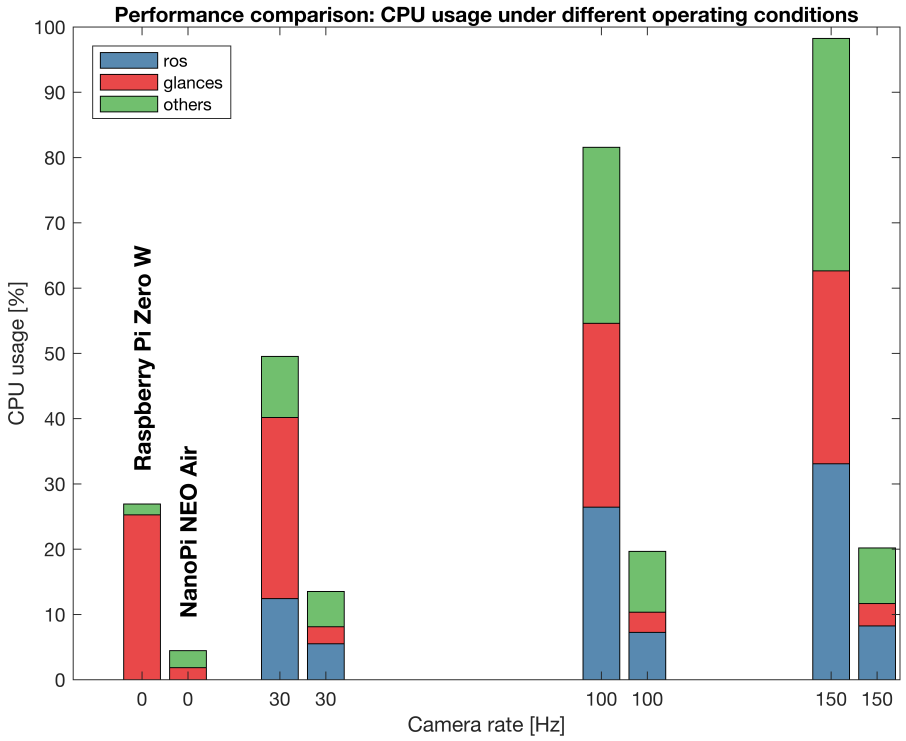


Figure 3.14: CPU load Rpi vs. NanoPi

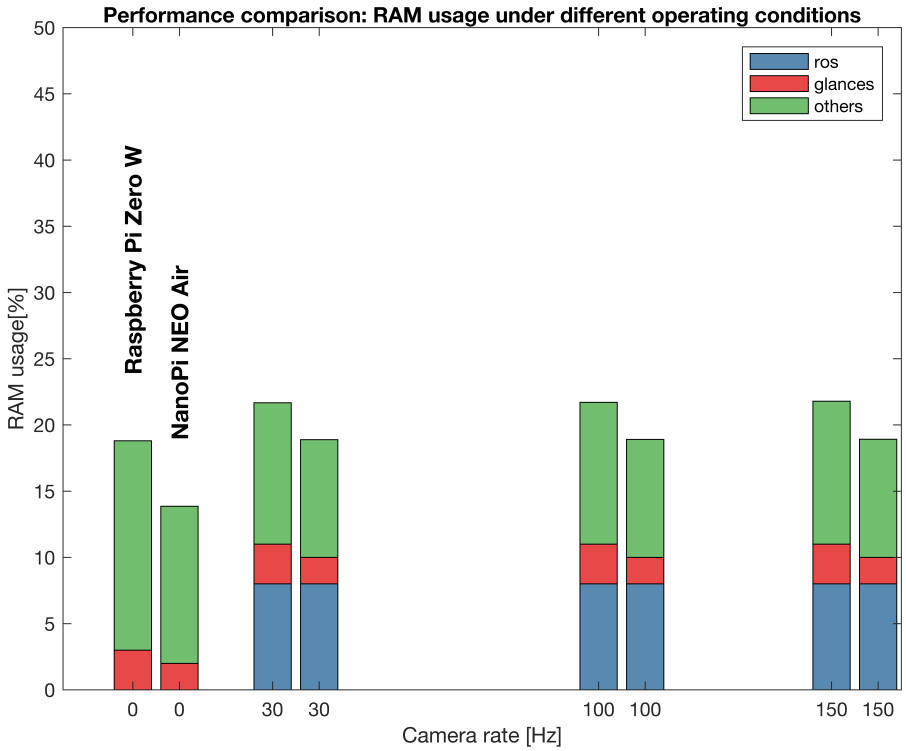


Figure 3.15: RAM load Rpi vs. NanoPi

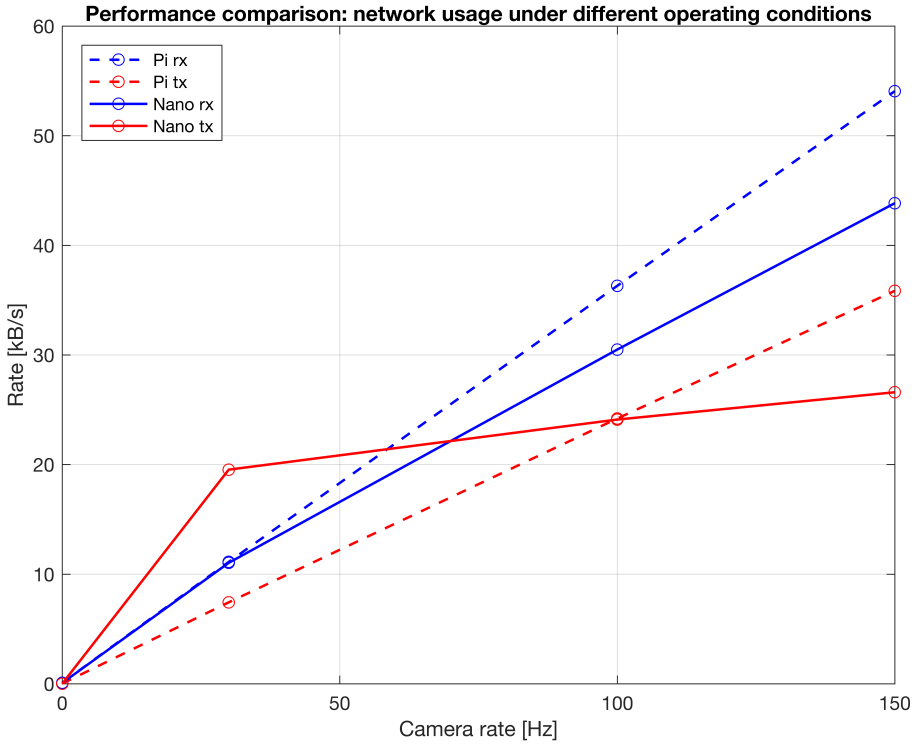


Figure 3.16: Wlan0 load Rpi vs. NanoPi

tween the companion and the PixFalcon means that the FCU cannot receive the position information as fast as the position information coming from the motion capture system and it can cause a not expected behaviour of the drone during the flight session. The serial communication encoding used by MAVlink introduces 2 bits of overhead, in fact the MAVlink frame used is composed by: 1 start bit, 8 data bits and 1 stop bit. This means that for transfer 1 Byte, 10 bits are required. The data rate can be easily computed as (3.1):

$$datarate = \frac{baudrate}{10} \tag{3.1}$$

and the latency can be computed in this way (3.2):

$$latency = \frac{1}{baudrate} \tag{3.2}$$

Experiment setup

The idea of this experiment is to retrieve the MAVlink status information using the NuttX real-time operating system that runs on PixFalcon which includes the NuttX Shell terminal (NSH). The NSH is very useful for diagnosing low level issues, it allows to execute some Unix style commands like “top” but permit also to:

- Display performance counters;
- Display px4io status information;
- Diagnose microSD errors;
- Diagnose sensor failures.

The NuttX Shell terminal is accessible in two different ways: connecting the FCU to a USB cable and controlling it by QGroundControl station or using the MAVproxy software already installed in the drone companion, through Wi-Fi connection. Since the data collection needs to be done in operative conditions the best solution is to use the Wi-Fi connection with the MAVproxy software.

For receiving data about FCU through MAVproxy, first of all we must ensure that the drone, in this case ANT-1, it is turned on (connected to a battery) and ready to fly with motion capture cameras (ROS is running on the drone companion and on the GS). After that, from a GS terminal it's necessary to execute MAVproxy using the following command line:

```
sudo mavproxy.py --master=udpout:<DRONE_IP_ADDRESS>:<SERVICE_PORT>
```

in which:

- `<DRONE_IP_ADDRESS>`: is the ip address of the drone currently used so in this case ANT-1 ip address;
- `<SERVICE_PORT>`: is the port number of the MAVproxy service and the default value is 14555.

When the script starts, the NSH module needs to be loaded and then executed using the following MAVproxy commands:

```
module load nsh

nsh start
```

once the NSH is executed, for retrieving information about data rate transmitted and data rate received, the following NSH command needs to be launched:

```
mavlink status
```

The experiment was made considering the variation of data rate on the MAVlink serial communication, increasing the motion cameras frequency. Seven different cameras rate was used and in particular for each of the following frequencies five samples are recorded: cameras off, 30 Hz, 60 Hz, 100 Hz, 140 Hz, 180 Hz, 200 Hz, 240 Hz.
In Figure 3.17, the results of the experiment are shown, and it is clear that the rate of the data received increase as the frequency of the motion cameras increase, while the rate of the data transmitted by FCU remain constant. The important result of this experiment is that else if the maximum threshold is 11,520 kB/s (computed with the Formula 3.1), the rate of the data received, saturate slightly the threshold value only when the cameras are at the maximum frequency. In operative condition the 240 Hz camera frequency is never used, so the result can be considered acceptable.

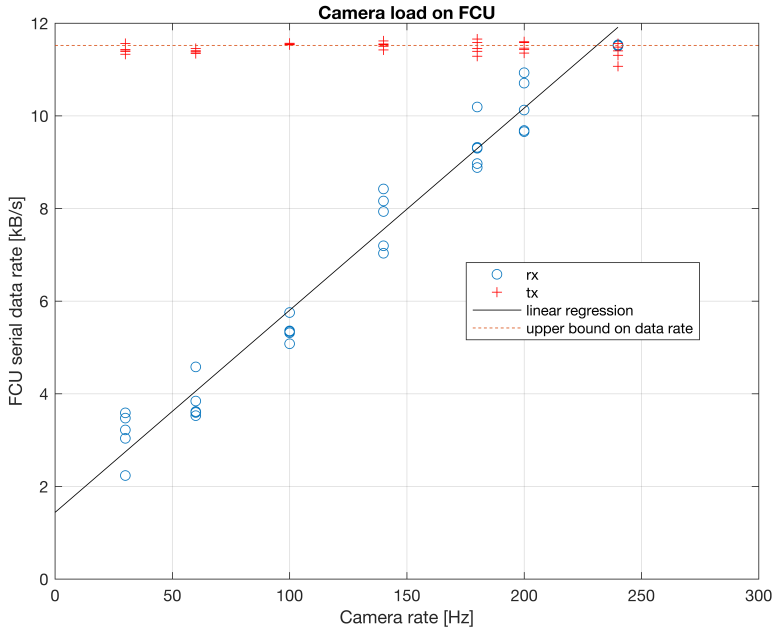


Figure 3.17: Cameras load on FCU

Conclusion

Since the cameras rate used for our purpose is always under 240 Hz, the data received not saturate the maximum rate threshold, the NanoPi does not cause a bottleneck for a serial communication even if its baud rate is less that Raspberry Pi Zero W baud rate. The rate of the data transmitted by FCU can be further reduced, since by default the topics not necessary for the laboratory purpose are

sent through it. For reducing the rate, the modality in which MAVlink starts needs to be modify.

3.7 Conclusion

At the end of hardware selection and evaluation phase, it is possible to conclude that the new NanoPi NEO Air is the best companion upgrade for ANT-1 drone, under different points of view. Its square shape, allow to design a smaller drone, which is a fundamental feature in research and development field. The high performances permit the execution of more complex and precise algorithms in less time, while the low price and the wide availability on the market allows fast replacement in case of damages.

Chapter 4

Software tools for control law development and implementation

This chapter will be dedicated to explain how the PX4 firmware was modified to import and use the new control laws, developed by researchers, in the simplest way possible.

The software tools used will also be described, in particular we will focus on the PX4 firmware, which is the official PixFalcon FCU firmware and the Simulink [**?**] software which is a development environment integrated in MATLAB. The last part of the chapter will be dedicated to describe the MATLAB GUI application that was created to compile and import the C++ library of the dynamic model created in Simulink into the PX4 firmware in an automatic way.

4.1 Introduction

As was discussed in the previous chapter, for this project the PixFalcon FCU was selected. The official PixFalcon firmware, PX4, is an open source software developed by world-class developers from industry and academia, and supported by an active world wide community, it powers all kinds of vehicles from racing to cargo drones through to ground vehicles and submersibles.

For control research purposes, however, it is necessary to be able to replace the PX4 attitude and position control modules with the custom one, implementing the control laws to be tested.

Clearly this task could be carried out by simply modifying the PX4 attitude and position control modules directly.

While very direct and certainly effective for minor modifications of the control laws, this approach turn out to be very impractical whenever significantly different control laws are to be implemented. Representative examples are the nonlinear and adaptive controllers discussed in the second part of this thesis.

Furthermore, in the framework of a model-based design approach for the con-

trol laws, the latter are initially specified using a modeling and simulation tool such as MATLAB Simulink. Therefore, the availability of a software tool enabling the automatic generation of PX4 attitude and position control modules, starting from their Simulink specification, would be very useful.

In this chapter first the Simulink simulation environment is described, then the PX4 firmware is introduced, with specific reference to the control modules and finally the developed automatic tool for control module generation is described.

4.2 Simulink

Simulink, an add-on product to MATLAB, is a graphical programming environment for modeling, simulating and analyzing multi domain dynamical systems and it is developed by MathWorks. Thanks to a set of customizable block libraries, it enables rapid construction of virtual prototypes to explore design concepts at any level of detail, with minimal effort. For the modeling part, Simulink provides a GUI for building models as block diagrams simply using the drag-and-drop operations. As we will se in the next chapter, it supports linear and nonlinear system, modeled in continuous-time, sampled time or hybrid of the two.

An interesting feature of Simulink is the possibility to generate C, C++ and other languages starting from a dynamic model. The generated code can be integrated like a source code, static libraries or dynamic libraries into application running outside of MATLAB and SImulink environment.

For the purpose of this thesis Simulink was used to model and simulate the PID (described in the next chapter), the nonlinear geometric and the adaptive attitude control laws. Once the simulation results were good enough a control law was exported as C++ library as described in the following.

Finally, since the C++ code exported in not properly certified, at the end of this chapter a section about code validation will be discussed.

4.3 PX4 architectural overview

4.3.1 Introduction to PX4

The PX4 firmware is an open-source autopilot system oriented toward low cost autonomous aircraft, fully compatible with the PixFalcon board and constantly under development by an active worldwide community. The PX4 firmware project started in 2009 and is being further developed and used at the Computer Vision and Geometry Laboratory of ETH Zurich. It can be download from GitHub [**?**] and the release used and presented in this thesis is v.1.7.3.

The customizability and ease of use of the PX4 firmware make it very popular in the research field. Unlike other autopilot open-source firmware, has the advan-

tage to support the information coming from the optical motion capture system (see Section 3.2.3) and other features are:

- controls many different vehicle frames/types, including: aircraft (multicopters, fixed wing aircraft and VTOLs), ground vehicles and underwater vehicles.
- Flexible and powerful flight modes and safety features
- Controllable from a separate on-vehicle companion computer via a serial cable.

4.3.2 Firmware structure

The entire PX4 firmware is based on the publish and subscribe pattern. The publish and subscribe is a messaging pattern where the senders of the messages, which are called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. The use of the publish-subscribe scheme means that:

- it is asynchronous and will update instantly when new data is available so this means that the system is reactive;
- all the operations and the communication are fully parallelized;
- the system components can consume data from anywhere in a thread-safe fashion.

In particular the entities which create, send and receive messages in PX4 are called modules. The PX4 firmware can be represented like a system composed by different modules, so each of them is a part of the system dedicated to compute a specific task. Modules communicate with each other through a publish-subscribe message bus named uORB, which is an asynchronous publish and subscribe messaging API used for inter-thread/inter-process communication.

The PX4 firmware consists of two main layers: the middleware and the flight stack. The middleware consists primarily of device drivers for embedded sensors, communication with the external world (*e.g.*, companion computer, GCS, etc.) and the uORB message bus. The flight stack, on the other hand is a collection of guidance, navigation and control algorithms for autonomous drones, see Figure 4.1.

4.4 Attitude controller implementation

The development of a new drone that will be used for educational and research purposes does not require only the use of hardware component with the features

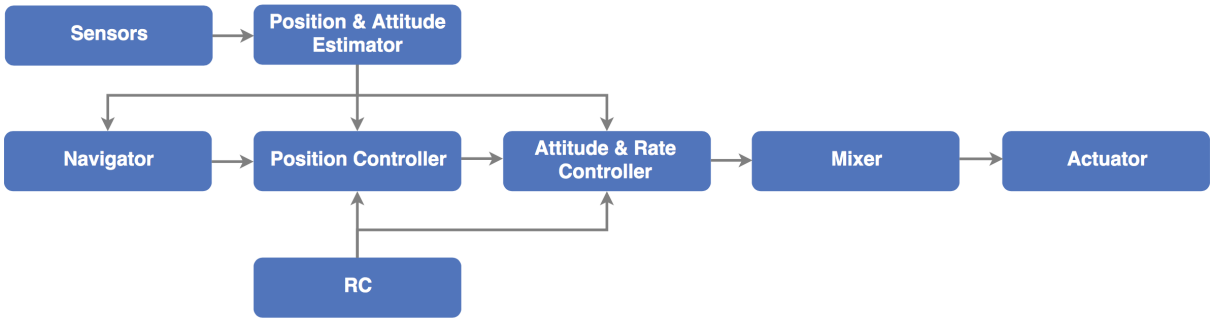


Figure 4.1: flight stack

defined in Section 3.1. It also requires an open source firmware which has the possibility to integrate the new control laws designed by researchers. Since PX4 is an open source firmware and also for its characteristics previously defined, it was chosen to support this project. In particular in this section we will focus on the "Attitude & Rate Controller" module shown in Figure 4.1 which by default contains a cascade PID architecture and we will explain how a new module was created to integrate and to use the control law defined in Simulink instead of the hardcoded one in PX4. For the purposes of this thesis the "Position Controller" module is not considered.

The idea is to exploit one of the features of Simulink, that is the possibility to export the C++ code (which is the same programming language of the PX4 firmware) of the dynamic model defined and import the generated library in the PX4 firmware. Before starting to explain how the new control law is implemented in PX4, it is necessary to define a Simulink template controller which provides the same I/O interface that will be defined in the PX4 firmware, so as to perfectly include the generated library in the firmware without I/O inconsistencies.

4.4.1 Definition of Simulink model interface

First of all we need to identify which are the I/O variables that will be used by the control laws and so integrated in the Simulink model interface. The model that has been defined has also taken into account the information available and the values format coming from the PX4 firmware both for inputs and outputs. The identified model is represented in Figure 4.2. As can be seen, the model is composed by seven inputs and one output:

- setpoint_attitude: which represents the attitude setpoint expressed in a quaternion form ($[q_x \quad q_y \quad q_z \quad q_w]$);

- setpoint_rates: which represents the angular rate setpoint expressed in rad/s for roll, pith and yaw;

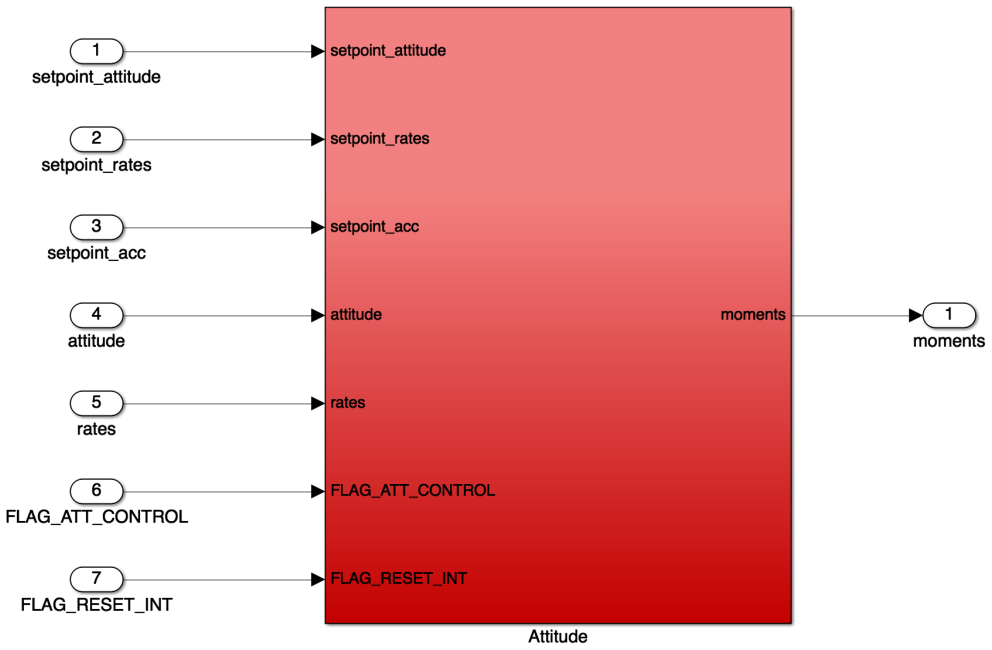


Figure 4.2: Simulink model template for the attitude controller

- setpoint_acc: which represents the angular acceleration setpoint expressed in rad/s^2.

Furthermore the attitude controller take as input two variables that come from the estimator:

- attitude: which represents the estimated attitude quaternion;
- rates: which represents the estimated angular rate expressed in rad/s for roll, pith and yaw.

Finally, two boolean values are added to the attitude controller template:

- FLAG_RESET_INT: this boolean value was introduced to avoid the continuous error integration by the attitude controller when the multirotor is not flying. For this reason it was decided that when the thrust value is below 20% the boolean value is equal to 1, so the integration is stopped. The boolean value is equal to 0 when the thrust is equal or above to 20% and in this case the attitude controller start to integrates again.
- FLAG_ATT_CONTROL: this boolean value was introduced to switch the flight controller used by the drone.

As output we have only one value:

- moments: which is a vector that contains the normalized moments (L, M and N) generated by the attitude controller for pitch, roll and yaw.

4.4.2 Interface implementation in PX4 firmware

Once defined the I/O interface of the Simulink attitude controller, the next step is to implement it also in the PX4 firmware. This operation is more complicated than the interface definition in Simulink because a new module needs to be created. As mentioned previously, PX4 is composed by different modules which correspond to specific task in the firmware. In this case the creation of a new module will be essential to handle the new controller that will be imported from Simulink and the Atom [**?**] IDE was used for code development. The tasks of the new module will be to subscribe to the needed topics, such as attitude, rate (coming from the estimator) and setpoints (coming from RC or GS), send this information to the exported Simulink library (which contains the control law) through the defined interface and retrieve the control actions (moments on pitch, roll and yaw).

The new module called "flight_controller"has been developed, taking as reference the template released by PX4 in the GitHub official repository. It is basically composed by a folder which contains: a ".hpp"file, a ".cpp"file a CMakeLists file and the library exported by Simulink. The ".hpp"is a header file, so it contains the variable declaration that will be defined in the ".cpp"file. The ".cpp"file is the core of the module. It is dedicated to create the association between the values coming from the PX4 system (like setpoints and estimator values) and the inputs needed by the control law (defined in the exported library), but also handle the association between the control actions retrieved by the control law and the actuators of the PX4 system. During the association phase between the PX4 module and the attitude controller library inputs and output, attention must be payed to the quaternion conversion. In fact the PX4 firmware uses the Hamiltonian convention, so the quaternion in composed in this way: $[q_w \quad q_x \quad q_y \quad q_z]$. On the other hand the attitude controller defined in Simulink required a quaternion form composed in this other way: $[q_x \quad q_y \quad q_z \quad q_w]$, in which the scalar component is no longer the first component, but becomes the last one. The CMakeLists is a file used by the PX4 system during the firmware compilation, used to "know"where the ".cpp"files are located. The CMakeLists also requires to specify the memory dimension that will be allocated when the module starts to run.

4.4.3 Flight controller Simulink implementation

Once defined the interfaces on both the Simulink model and the PX4 firmware, it is possible to implement a flight controller inside the "red box"of the model shown in Figure 4.2, knowing the available inputs and the output required by PX4. As said previously, Simulink permits to generate a C++ library of the model defined in the "red box"which implements three methods that need to be invoked in the PX4 module defined before (flight_controller module). The three methods are:

- initialize( ): permits to instantiate and allocate in the memory the variables

that will be used during the flight controller computational procedure.

- step( ): is the method used to call the library imported in PX4 and start its execution. The moments (L, M and N) are available only after the use of this method.

- terminate( ): this routine is called when the controller ends its execution. The terminate method was never used in the defined module, since the idea is that the attitude controller never stops its execution until the battery is disconnected from the drone. So this method in this specific case, results useless.

4.5 Simulink to PX4 application

The Simulink to PX4 (SLXtoPX4) application is a Graphical User Interface (GUI) developed using the App Designer environment integrated in MATLAB. The idea of this application is to make the process of building the controller model defined in Simulink and the PX4 firmware integration with the C++ library exported, as automatic as possible. Figure 4.3 shows the graphical appearance of the application. Its operation is quite simple and intuitive. In fact it is necessary to select the dynamic model defined in the ".slx "file and then the path in which the PX4 firmware is contained. When the "Run "button is clicked, the Simulink controller will be compiled, generated as C++ library and imported into PX4 firmware, more precisely in the "flight_controller"module defined before.

4.6 Validation

Since the C++ code generate in Simulink is not certified, the code generation needs to be validated to verify if the attitude control law defined in Simulink behaves as expected. For this test, the idea was to implement a PID control law architecture (better described in the following chapter) in Simulink, export the model as C++ library and include it in the PX4 firmware.

Once the modified firmware was flashed in the ANT-1 drone, a flight session was made in to the flight arena (for safety reasons) in such a way to verify the drone behavior during the manual flight session and also to analyze the data logged at the end of the test.

In the following two charts of the logged data are reported. In particular Figure 4.4 shows the attitude setpoints for roll, pitch and yaw axes, given by the remote controller during the flight session and the ANT-1 attitude (always on the roll, pitch and yaw axes) which follows well the setpoints.

Figure 4.5 instead, shows the control actions produced by the C++ library exported by Simulink dynamic model. Also in this case is clearly visible that

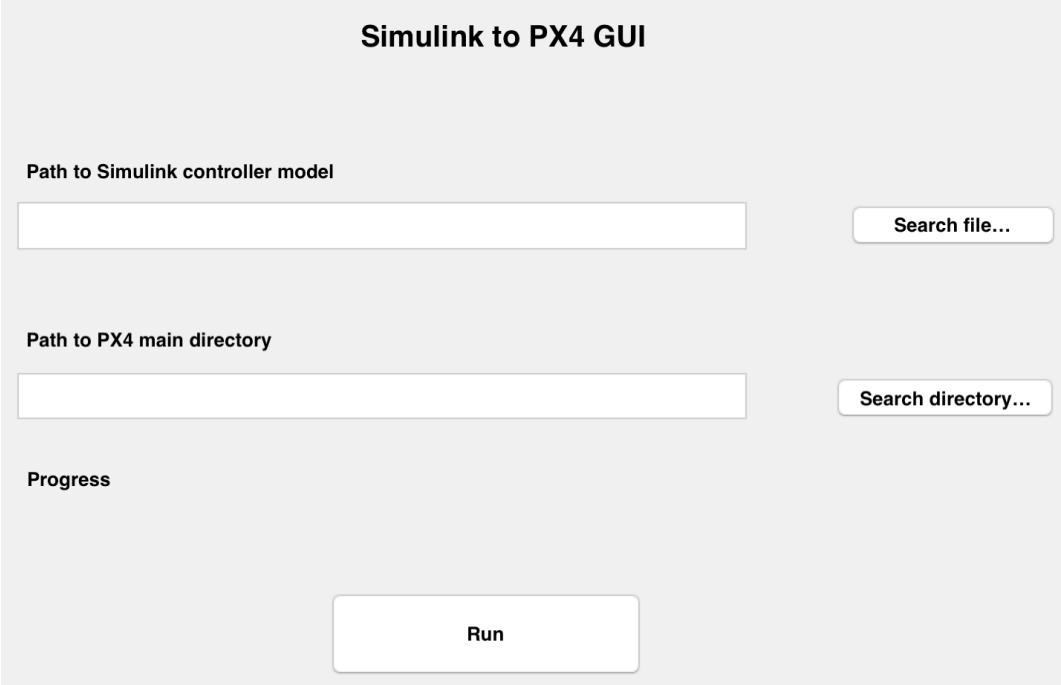


Figure 4.3: SLXtoPX4 GUI application

the values produced are not *null* or zeros and they are in the expected order of magnitude (remember that, the control action produced are normalized values).

In conclusion, since ANT-1 with the modify firmware, behaves as expected in simulation, the library exported can be considered reliable. In the second part of this thesis the Simulink to PX4 application will be used also for nonlinear and adaptive controllers.

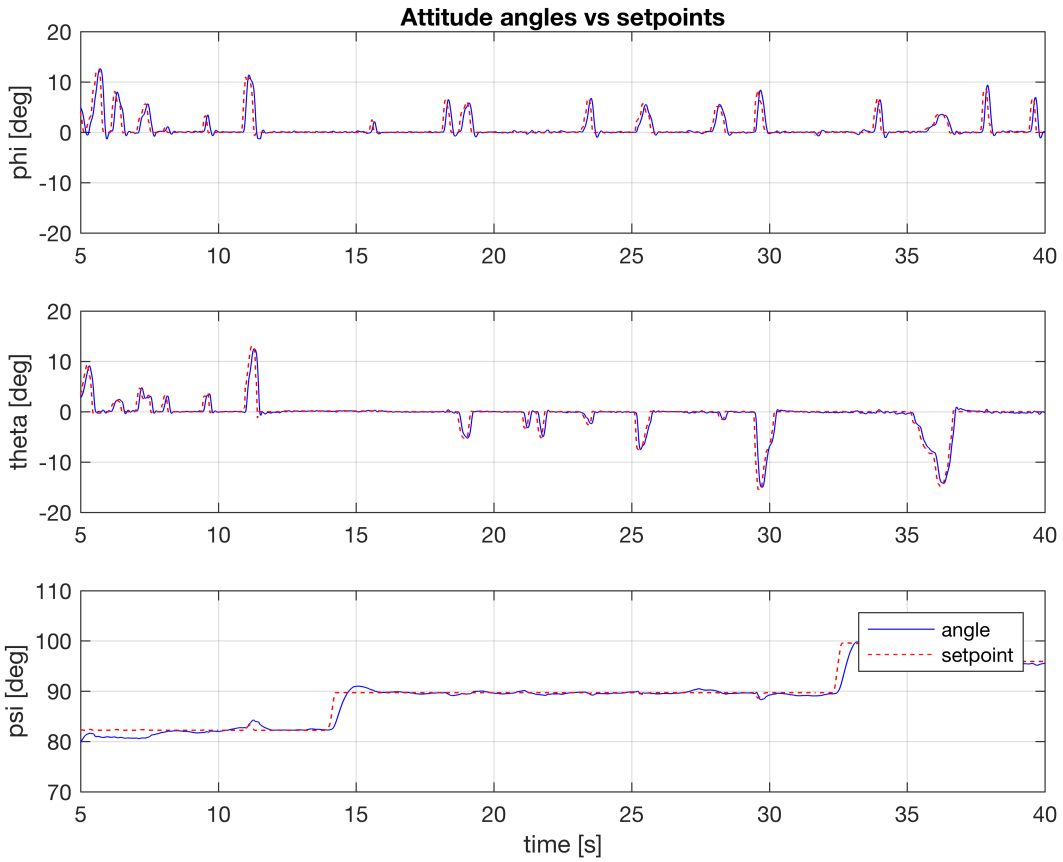


Figure 4.4: attitude angles vs setpoins

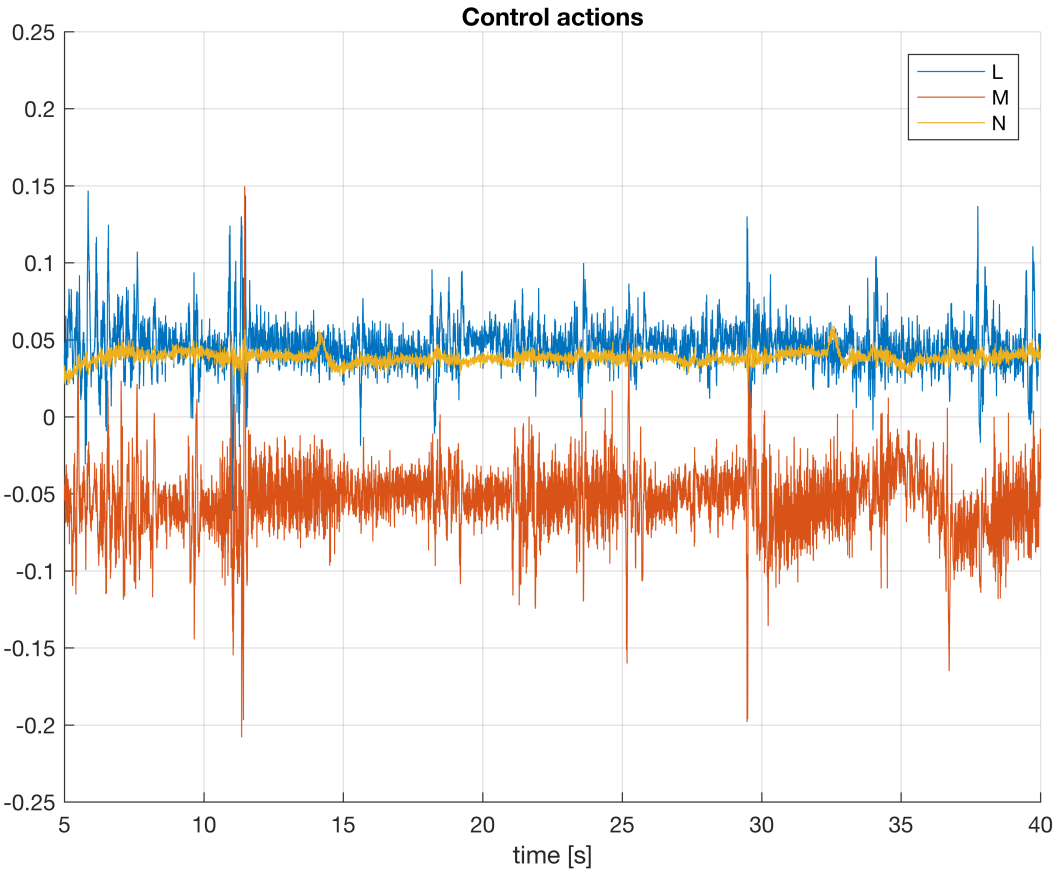


Figure 4.5: PID control actions

Chapter 5

Introduction to attitude control of multirotor UAVs

Thanks to the modified PX4 firmware, now the import of a custom attitude control law into the firmware is a simple operation. In this chapter the formalism and the method used to define and attitude control law of multirotor UAVs will be described. At the end of the chapter a linear attitude control law will be explained, and in particular we will focus on the attitude control law implemented in the original PX4 firmware.

5.1 Formalism

5.1.1 Reference frames and axes

The motion of a body in space is described thanks to reference systems that need to be properly chosen. Many conventions are known in the literature. The chosen reference systems follow the same way the NED standard. The Earth fixed frame, assumed to be an inertial frame, is defined as $\mathcal{F}_E = \{O_E, N, E, D\}$, where O_E is a point on the Earth surface. The N-axis and the E-axis are chosen to point respectively North and East and the D-axis completes the right-hand rule pointing downward. The second frame to be defined is the body frame $\mathcal{F}_B = \{O_B, X_B, Y_B, Z_V\}$, where O_B corresponds to the body center of mass. The X-body axis lies in the plane of symmetry and generally points forward. The Y-body axis points to the right wing, normal to the plane of symmetry, and the Z-body axis points down.

5.1.2 Rotation matrices and Euler angles

A rotation matrix is a matrix $\in \mathrm{SO}(3)$

$$\mathrm{SO}(3) := \left\{ R \in \mathbb{R}^{3\times 3} : RR^T = I_3,\, \det(R) = 1 \right\}, \tag{5.1}$$

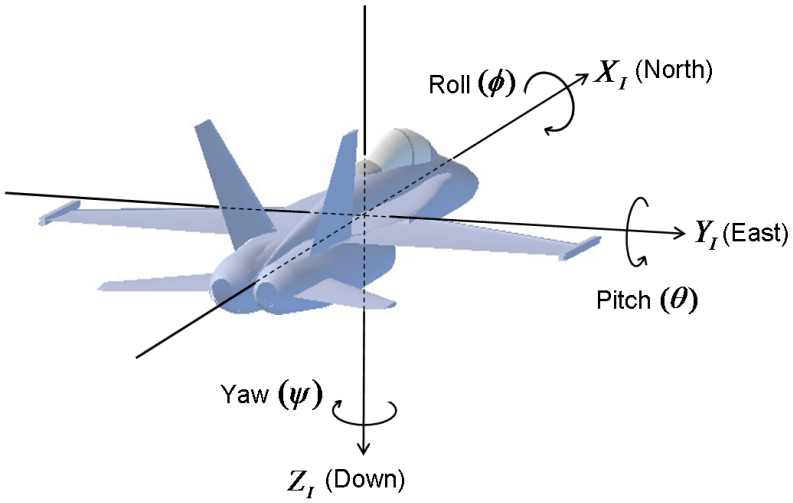


Figure 5.1: Body axes

A rotation matrix could expresses three different meanings:

- the orientation of a frame with respect to another frame;
- the transformation that relates the coordinates of a point in two different frames;
- the rotation of a vector in a coordinate frame.

In order to rotate a vector around different axes many times, it is useful to define the rotations around a coordinate axis and in particular:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{5.2}$$

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \tag{5.3}$$

$$R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

The rotation direction is considered positive counterclockwise, following the right-hand rule. Since rotation matrices are orthonormal, the inverse rotation is

obtained by transposing it. In fact:

$$R_x^{-1} = R_x^T \tag{5.5}$$
$$R_y^{-1} = R_y^T \tag{5.6}$$
$$R_z^{-1} = R_z^T. \tag{5.7}$$

Many consecutive rotations could be performed multiplying the rotation matrices, noting that rotations performed in different order produce different results ($R_x(\alpha)R_y(\beta) \neq R_y(\beta)R_x(\alpha)$). When a vector or a frame is rotated an arbitrary number of times, the final attitude vector could be represented by a minimal representation, that consists in performing just three consecutive rotations about R_z, R_y, R_x.

A minimal representation is a parameterization of the attitude with respect to three parameters, called Euler angles $\Phi = [\phi \quad \theta \quad \psi]^T$. They represents the three angles of the rotations performed about R_x, R_y, R_z, also called *roll, pitch, yaw.*

The rotation from the earth frame to the body frame can be expressed in this form:

$$T_{BE}(\phi, \theta, \psi) = R_X(\phi)R_Y(\theta)R_Z(\psi), \tag{5.8}$$

where the subscripts B and E stand for "Body" and "Earth", respectively.
Matrix T_{BE} resolves an Earth-based vector to body axes. Expanding the indicated matrix multiplication, the T_{BE} matrix has these elements:

$$T_{BE}(\phi, \theta, \psi) = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_{psi} & s_\phi c_\theta \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_{psi} & c_\phi c_\theta \end{bmatrix}, \tag{5.9}$$

where for the matrix (5.9) reported a short notation has been reported, which is $c_\theta = \cos(\theta)$, $s_\theta = \sin(\theta)$ and $t_\theta = \tan(\theta)$.
On the other hand, it is also possible to obtain the T_{EB} (the rotation matrix from "Earth" to "Body") simply transposing T_{BE}, by obtaining:

$$T_{EB} = T_{BE}^T. \tag{5.10}$$

5.1.3 Algebra for matrices

The purpose of this section is to recall some notions about matrix algebra, since they will be used for the advanced control laws, defined in the Chapter 6.

Skew-symmetric matrix

A matrix $A \in \mathbb{R}^{3\times3}$ is skew symmetric (or anti-symmetric) when the following condition is satisfied:

$$A^T = -A. \tag{5.11}$$

The $^\wedge$ operator represents the map transforming a vector $a \in \mathbb{R}$ into the associated skew-symmetric matrix $\hat{a} \in \mathrm{SO}(3)$:

$$\hat{a} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}. \tag{5.12}$$

Cross product

The transformation of a vector in its corresponding skew-symmetric matrix is useful also to compute the cross product. In fact, given two vectors $a, b \in \mathbb{R}^3$:

$$a \times b = \hat{a}b = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}. \tag{5.13}$$

Matrix derivative

The derivative of a rotation matrix is given by:

$$\dot{R} = R\hat{\omega} \tag{5.14}$$

where $R \in \mathrm{SO}(3)$ is a rotation matrix that expresses the transformation from a reference frame to another and ω is the vector of angular rates resolved in the first frame.

5.2 Attitude control model

This section will be dedicated to explain the mathematical and physical general concepts behind the attitude control law. In particular, the explanation will include the model kinematics description and then the use of the Newton-Euler method by considering forces and moments generated by the propellers, aerodynamic damping and external forces.

5.2.1 Kinematics

Let p be the position vector and v the velocity vector of the quadrotor center of mass expressed in the inertial frame:

$$p = \begin{bmatrix} n \\ e \\ d \end{bmatrix} \tag{5.15}$$

$$v = \dot{p} = \begin{bmatrix} \dot{n} \\ \dot{e} \\ \dot{d} \end{bmatrix}. \tag{5.16}$$

Thanks to the parametrization using the Euler angles, the attitude of the quadrotor could be expressed in different ways. One of the most common way to describe the quadrotor attitude is through the Euler angles vector, defined as:

$$\Phi = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}. \tag{5.17}$$

Differentiating the Euler angles (5.17), it is possible to obtain the Euler rates as:

$$\dot{\Phi} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{5.18}$$

and the body angular velocity as:

$$\omega_b = \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{5.19}$$

To get the body axes rates from the Euler rates, consider the Euler rates individually, resolve them individually to intermediate axes, and then finally to body axes (see [1]). Define the Euler rate elemental vectors:

$$\omega_{\dot{\phi}} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix}, \quad \omega_{\dot{\theta}} = \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix}, \quad \omega_{\dot{\psi}} = \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}. \tag{5.20}$$

Rotate $\omega_{\dot{\phi}}$ through the angle θ about the Y axis, and add the results to the $\omega_{\dot{\theta}}$ vector. Rotate that sum about X axis through angle ϕ, and add the result to the $\omega_{\dot{\phi}}$ vector. The results is the vector of body-axis angular rate,

$$\omega_b = \omega_{\dot{\phi}} + R_X(\phi)\big[\omega_{\dot{\theta}} + R_Y(\theta)\omega_{\dot{\psi}}\big]. \tag{5.21}$$

After some rearrangement,

$$\omega_b = E(\phi,\theta)\dot{\Phi} = \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & s_\phi s_\theta \\ 0 & -s_\phi & c_\phi c_\theta \end{bmatrix} \dot{\Phi}. \tag{5.22}$$

To get the Euler rate in terms of body axis rates, one must invert the transformation matrix E. Unlike the T_{BE} matrix, the inverse of E is not the transpose. Worse yet, the inverse is singular at pitch angle of $\pm$ 90°:

$$E^{-1}(\phi,\theta) = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & \frac{s_\phi}{c_\theta} & \frac{c_\phi}{c_\theta} \end{bmatrix}. \tag{5.23}$$

This unfortunate singularity is called gimbal lock.
Another type of parametrization used in the Chapter 6 for the advanced control laws is the quaternion parametrization.

Quaternions

To avoid gimbal lock, it is possible to use another representation called quaternion.
A quaternion is a four-dimensional representation of a sphere that can be used to represent the orientation of a rigid body or a coordinate frame in three-dimensional space and are generally represented in the form:

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{5.24}$$

where q_0, q_1, q_2, q_3 are real numbers such that $q^T q = 1$.

The quaternion elements generate the following coordinate transformation:

$$T_{BE} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 + q_0 q_3) & 2(q_1 q_3 - q_0 q_2) \\ 2(q_1 q_2 - q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 + q_0 q_1) \\ 2(q_1 q_3 + q_0 q_2) & 2(q_2 q_3 - q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}. \tag{5.25}$$

The Euler angles representation of q_{BE} is defined by:

$$\phi = \tan^{-1}\left(\frac{T_{BE}(2,3)}{T_{BE}(3,3)}\right), \tag{5.26}$$

$$\theta = \sin^{-1}\big(T_{BE}(1,3)\big), \tag{5.27}$$

$$\psi = \tan^{-1}\left(\frac{T_{BE}(1,2)}{T_{BE}(1,1)}\right). \tag{5.28}$$

The quaternion conjugate, denoted by *, can be used to swap the relative frames described by an orientation. For example, the attitude of frame B relative to frame A can be represented by the quaternion q_{AB}, and its conjugate q_{AB}^* describes the orientation of frame A relative to frame B (q_{BA}). The conjugate of q_{AB} can be described by the following equation:

$$q_{AB}^* = q_{BA} = \begin{bmatrix} q_0 \\ -q_1 \\ -q_2 \\ -q_3 \end{bmatrix}. \tag{5.29}$$

The quaternion product, denoted by $\otimes$, can be used to define compound orientations. For example, for two orientations described by q_{AB} and q_{BC}, the compounded orientation q_{AC} can be obtained in this way:

$$q_{AC} = q_{BC} \otimes q_{AB}. \tag{5.30}$$

For two quaternions, a and b, the quaternion product can be determined using the Hamilton rule:

$$a \otimes b = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \otimes \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4 \\ a_1b_2 + a_2b_1 + a_3b_4 - a_4b_3 \\ a_1b_3 - a_2b_4 + a_3b_1 + a_4b_2 \\ a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1 \end{bmatrix}. \tag{5.31}$$

The quaternion product is not commutative, so $a \otimes b \neq b \otimes a$.

5.2.2 Equation of motion

Linear motion

The quadcopter is assumed to be a rigid body and thus Newton-Euler equation it can be used to describe its dynamics. Starting from the second Newton's Law is possible to write the linear motion equations:

$$F = \frac{d(mV_b)}{dt} = \left(\frac{dm}{dt}\right)V_b + \left(\frac{\partial V_b}{\partial t} + \omega_b \times V_b\right) \tag{5.32}$$

and if the mass m of the quadcopter is constant, equation (5.32) can be written as:

$$m\dot{V_b} + \omega_b \times (mV_b) = F_{ext} \tag{5.33}$$

where V_b is the linear velocity resolved to the body axes, ω_b is the body angular velocity defined in the equation (5.19) and the F_{ext} represent the sum of all forces applied to the quadcopter such as, aerodynamics forces, gravity, forces generated by propellers and other kinds of external forces. The F_{ext} vector can be represented in this way:

$$F_{ext} = \begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix}. \tag{5.34}$$

Angular motion

The angular motion equation of the quadcopter is obtained applying the Euler's rotation formula:

$$I_n\dot{\omega}_b + \omega_b \times (I_n\omega_b) = M_{ext}. \tag{5.35}$$

The I_n value represent the inertia tensor, and it can be defined as:

$$I_n = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}. \tag{5.36}$$

If the quadcopter frame can be considered symmetric, then the inertia tensor can be considered a diagonal matrix:

$$I_n = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}. \tag{5.37}$$

Finally, M_{ext} is a vector of three elements and it represent the moments applied on the body-axis X, Y and Z like in the following:

$$M_{ext} = \begin{bmatrix} L \\ M \\ N \end{bmatrix}. \tag{5.38}$$

The 3×1 vectors F_{ext}, previously defined in the equation (5.33) and the M_{ext} in the equation (5.35), are the total external forces and moments respectively, that are applied to the body at its center of gravity.

5.2.3 Mixer

Once defined the rigid body dynamics one has to take into account the geometry of a quadrotor, in order to define the forces and moments that have to be considered and the application point.

The ANT-1 drone has a X- configuration, as well as the drone that will be created for educational and research, so the label of each propeller and its rotation direction are reported in Figure 5.2.

Each propeller produce a thrust and torque proportional to the square of the rotational speed:

$$T_i = K_T \Omega_i^2, \quad K_T = C_T \rho A R^2, \tag{5.39}$$

$$Q_i = K_Q \Omega_i^2, \quad K_Q = C_Q \rho A R^3, \tag{5.40}$$

where C_T and C_Q are the thrust and torque coefficients, ρ is the air density, A and R are the area of the propeller disk and its radius respectively and Ω is the rotational speed. Since the quadrotor is symmetrical, it is possible to consider the distance between the center of gravity and the propeller, equal for all arms of the quadrotor and represented with b.

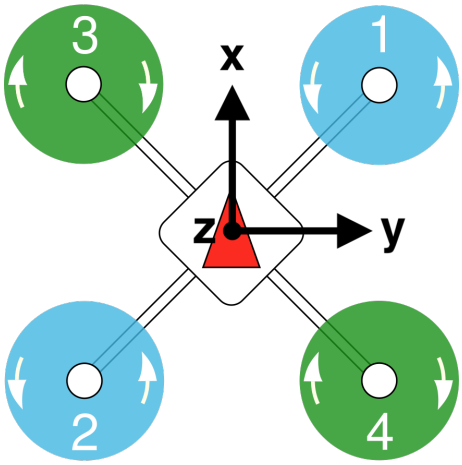


Figure 5.2: Quadcopter configuration

Now it is possible to write the equations of the forces and the moments produced by the four propellers:

$$F_{props} = -\begin{bmatrix} 0 \\ 0 \\ K_T(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \end{bmatrix}, \tag{5.41}$$

$$M_{props} = \begin{bmatrix} K_T\frac{b}{\sqrt{2}}(-\Omega_1^2 + \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \\ K_T\frac{b}{\sqrt{2}}(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \\ K_Q(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 - \Omega_4^2) \end{bmatrix}. \tag{5.42}$$

The mixer matrix, is a matrix that relates the thrust and the moments around each axis required by the quadrocopter to the speed of the actuators. Combining the forces and moments matrices is possible to obtain:

$$\begin{bmatrix} T \\ L \\ M \\ N \end{bmatrix} = \begin{bmatrix} K_T & K_T & K_T & K_T \\ -K_T\frac{b}{\sqrt{2}} & K_T\frac{b}{\sqrt{2}} & K_T\frac{b}{\sqrt{2}} & -K_T\frac{b}{\sqrt{2}} \\ K_T\frac{b}{\sqrt{2}} & -K_T\frac{b}{\sqrt{2}} & K_T\frac{b}{\sqrt{2}} & -K_T\frac{b}{\sqrt{2}} \\ K_Q & K_Q & -K_Q & -K_Q \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix}, \tag{5.43}$$

reversing the equation:

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4K_T} & -\frac{\sqrt{2}}{4K_Tb} & \frac{\sqrt{2}}{4K_Tb} & \frac{1}{4K_Q} \\ \frac{1}{4K_T} & \frac{\sqrt{2}}{4K_Tb} & -\frac{\sqrt{2}}{4K_Tb} & \frac{1}{4K_Q} \\ \frac{1}{4K_T} & \frac{\sqrt{2}}{4K_Tb} & \frac{\sqrt{2}}{4K_Tb} & -\frac{1}{4K_Q} \\ \frac{1}{4K_T} & -\frac{\sqrt{2}}{4K_Tb} & -\frac{\sqrt{2}}{4K_Tb} & -\frac{1}{4K_Q} \end{bmatrix} \begin{bmatrix} T \\ L \\ M \\ N \end{bmatrix}. \tag{5.44}$$

Notice that the mixer matrix is consistent with the label of each propeller and their rotation as defined by PX4, but since the PX4 mixer matrix receive

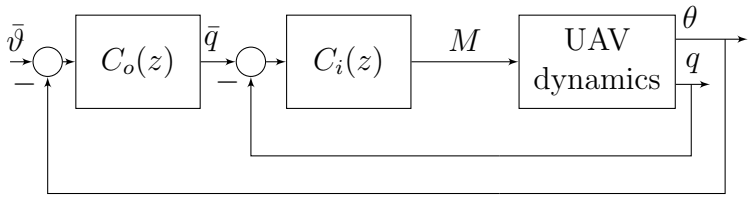


Figure 5.3: PID architecture implemented

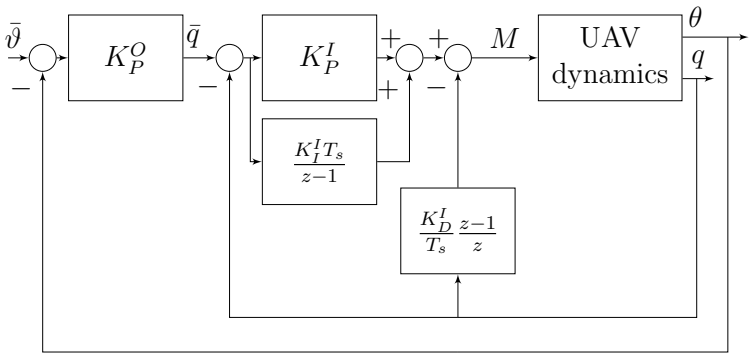


Figure 5.4: PID attitude controller block diagram.

normalized moments as input, the K_T, K_Q coefficient and the length of the arms b must not be considered. In fact the PX4 mixer matrix consider only the geometry of the drone like the number of motors, the configuration of the motors (in this case X configuration) the label of each propeller and their rotation.

5.3 Attitude control law

5.3.1 Introduction

An example of attitude control law architecture is the one implemented in PX4, which is based on two cascaded PID loops as displayed in figure 5.3 and it is implemented on all roll, pitch and yaw axes.

In particular in PX4, the outer regulator $C_o(z)$ is a P controller based on attitude feedback while the inner regulator $C_i(z)$ is a PID controller based on angular rate feedback.

The derivative action, implemented in this architecture, is computed on the process output and not on the error (Figure 5.4). In this way is not possible to generate an impulse on the control actions when there is a step in the reference signal of the inner loop.

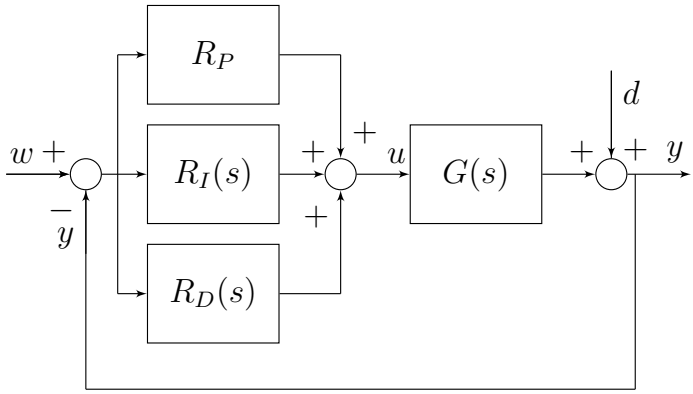


Figure 5.5: PID structure

5.3.2 PID regulator

One of the linear regulator most used in the industrial area is surely the Proportional, Integral and Derivative (PID) regulator. The reasons of this success are mainly three:

- simple structure,
- good performance for several processes,
- good results of tuning also without a specific model of the controlled system.

In robotics, the PID technique represent the basics of control. Even though a lot of different algorithms provide better performance than PID, this last structure is often chosen for the reasons expressed before.

5.3.3 PID regulator model

The traditional PID structure is composed of the addition of different contributes. In particular, as shown in the Figure 5.5 the control variable u, is generated like the sum of three contributes:

- the first term is proportional to the error e defined as the difference between the setpoint w and the controlled variable y which is the exit variable of the controlled system;
- the second term is proportional to the integral value of e, and this term in necessary to cancel the asymptotic error value due to constant disturbances;
- the last term is proportional to the derivative value of e and its purpose is to anticipate the behavior of the error.

In the time domain, the PID control law can be written in according to the equation (5.45):

$$u(t) = K_P e(t) + K_I \int_{t_0}^{t} e(\tau)\,\mathrm{d}\tau + K_D \frac{de(t)}{dt} \tag{5.45}$$

where K_P, K_I and K_D are positive or null constants (considering the process gain positive). The K_P term is the proportional coefficient, while the K_I and K_D are the integral and derivative coefficient respectively.

Applying the Laplace transformation to the equation (5.45), considering the value $t_0 = 0$, is possible to describe the PID control law as the following transfer function:

$$R_{PID} = K_P + \frac{K_I}{s} + K_D s = \frac{K_D s^2 + K_P s + K_I}{s} \tag{5.46}$$

in which it is possible to find the proportional ($R_p = K_p$), the derivative ($R_D = K_D s$) and the integral ($R_I = \frac{K_I}{s}$) terms like those represented in the Figure 5.5.

The ideal PID transfer function (equation (5.48)) is a system composed of two zeroes with negative real part and one pole in the origin. Since this function is improper, because of the derivative term ($R_D = K_D s$), it is not physically feasible. After a certain frequency, the derivative contribute must be attenuated to filter the off-band noise. For this reason, in the real derivator a pole is added as shown in the equation (5.47):

$$R_D(s) = \frac{K_D s}{1 + \frac{K_D}{K_P N} s} \tag{5.47}$$

where the N value is chosen to move the pole ($s = -\frac{K_P}{K_P N}$) outside of the interested system control frequency bandwidth. At the end is possible to write the non ideal PID transfer function adding the real derivator:

$$R_{PID} = K_P + \frac{K_I}{s} + \frac{K_D s}{1 + \frac{K_D}{K_P N} s} \tag{5.48}$$

5.3.4 Realization of PID regulators

Differently from the theoretical structure of PID regulators, some drawbacks are present during the real implementation. In particular the ideal PID present two main limitations:

- the derivate action is calculated from the error. If the task adds a step in the reference, the output of the derivator would present an impulse. This sharp movement can saturate the actuators and push away the system from the linear zone. For this reasons most of the PID architecture presents the derivate action of the process output only (Figure 5.6);

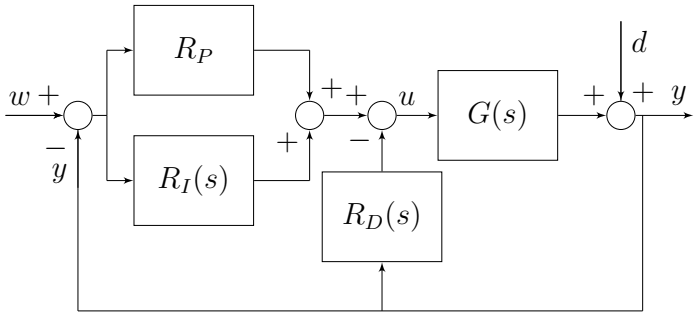


Figure 5.6: PID with output derivator

- the integral action combined with an actuator saturation can provide a non linear effect which can decrease the performance of the control system. When the integral value is large and the error changes sign it is necessary to wait a lot of time before the system restores its linear behavior (after the "discharging" of the integral action). This phenomenon is called integral wind-up. To avoid it, a saturator is added after the integral to limit its maximum and minimum values.

5.3.5 PID Simulink implementation

One of the control law implemented for the ANT-1 drone, is the linear PID attitude control law. The theory behind the PID control law was previously defined and in this section, the PID architecture implemented in Simulink will be explained.

Loop shaping

The PID gains value (K_P^O, K_P^I, K_I^I and K_D^I) have been found using a loop shaping process applied on ANT-1 pitch dynamic model already defined (see [6]) and simulated in Simulink.

Since the dynamic model was defined only for the pitch axes, to find the gains also for the roll and yaw axes, two different approaches are used.

For the roll axis, knowing the ratio between pitch and roll inertia which is +10%, the gains were found simply scaling the gains of the pitch axis by -10% and adjusted using trial and error method.

Instead, for the yaw axis is not possible to apply the ration between inertia, so for this axis only the trial and error method was used. Table 5.1 show the gains found with the approaches described before for the thee axes and Figure 5.7 shows how the PID was implemented in Simulink.

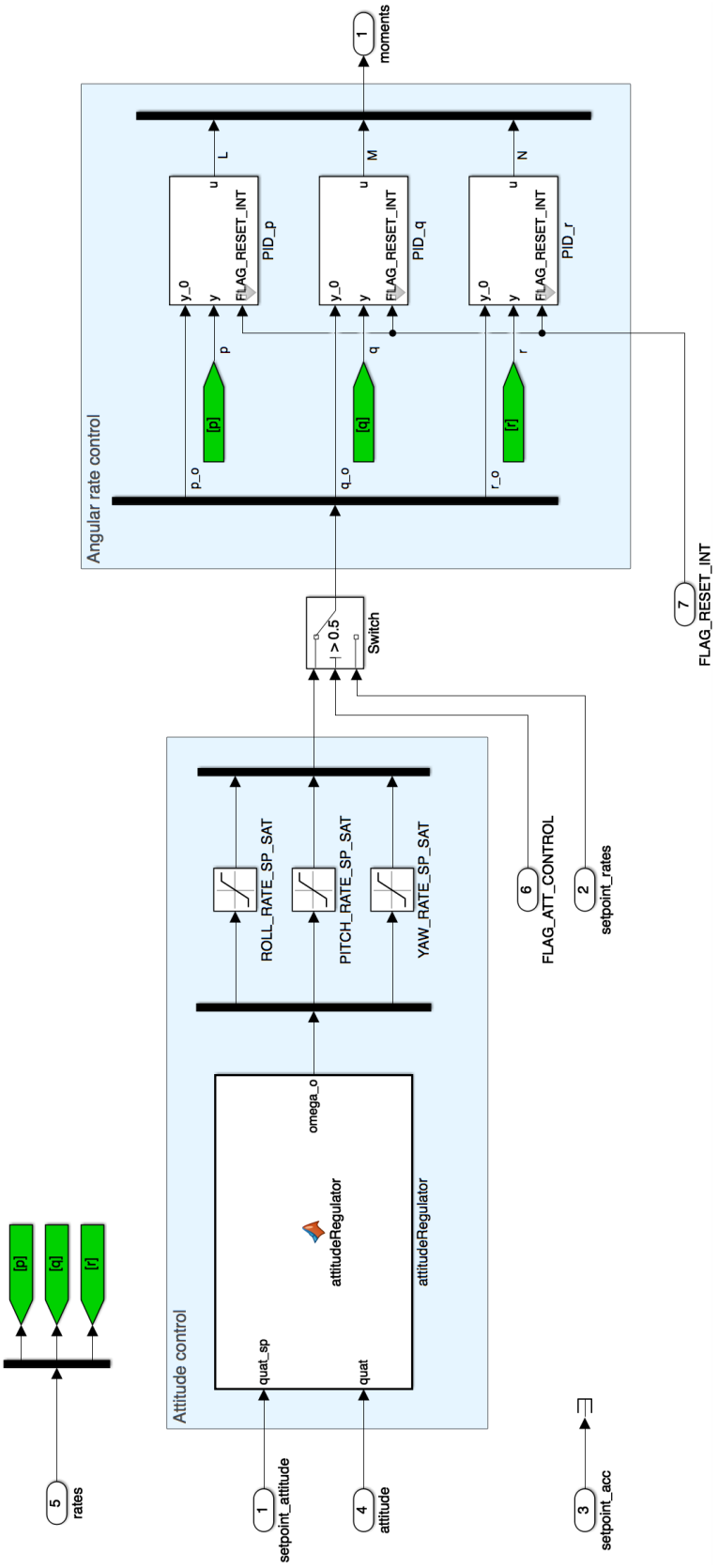


Figure 5.7: PID implemented in Simulink

Gains	Pitch	Roll	Yaw
K_P^O	13.0	13.0	4.0
K_P^I	0.14	0.10	0.06
K_I^I	0.21	0.21	0.05
K_D^I	0.003	0.003	0.001

Table 5.1: PID gains

Chapter 6

Advanced control law

In this chapter two advanced control laws will be presented, and in particular we will focus on the nonlinear geometric control law, which is based on the application of differential geometric techniques, and the adaptive control law.

Adaptive control laws provide a systematic approach for automatic adjustment of controllers in real time, in order to achieve or to maintain a desired level of control system performance when the parameters of the plant dynamic model are unknown or change in time. The proposed approach described in the following allows to combine a baseline controller (PID controller) with a Model Reference Adaptive Controller (MRAC) and to enable or disable the adaptive controller when needed, in order to take the advantages of both controllers.

6.1 Geometric control law

The first advanced control law presented in this chapter is based on geometric control theory which exports the application of differential geometric techniques to control systems. Despite the substantial interest in multirotor UAVs, little attention has been paid to designing nonlinear control systems for them. The nonlinear geometric controller presented here is developed on the group of rigid displacements SO(3) which is a special orthogonal group, yielding a control law that is independent of any parametrization of the configuration space. Indeed, Euler angles are not suitable since they suffer singularity conditions, whereas the use of quaternions requires special care, as they doubly cover the special orthogonal group SO(3) and have an intrinsic ambiguity in representing the attitude.

Geometric control is concerned with the development of control laws for dynamic systems evolving on nonlinear manifolds that cannot be globally identified with Euclidean spaces. By characterizing geometric properties of nonlinear manifolds intrinsically, geometric control techniques provide unique insights to control theory that cannot be obtained from dynamic models represented using local coordinates. In the next section, the Lyapunov-based approach will be used to derive

a geometric control law for attitude tracking.

6.1.1 Tracking errors

Before deriving the geometric control law, some definitions about attitude tracking errors need to be recalled. Since in this control architecture the attitude is directly represented as a rotation matrix, the attitude error in SO(3) can be obtained according to the right error representation, which is defined as:

$$R_e = R_d^T R \tag{6.1}$$

where $R_d \in \mathrm{SO}(3)$ is the desired orientation matrix:

$$R_d = [b_{d1} \quad b_{d2} \quad b_{d3}]. \tag{6.2}$$

The right angular velocity error is obtained by taking the time derivative of the attitude error (6.1):

$$\dot{R}_e = \dot{R}_d^T R + R_d^T \dot{R}. \tag{6.3}$$

Recalling also that

$$\dot{R} = R\hat{\omega} \tag{6.4}$$

$$\dot{R}_d = R_d \hat{\omega}_d \tag{6.5}$$

where $\hat{\omega} \in \mathbb{R}^3$ is the antisymmetric matrix associated with ω:

$$\hat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}, \tag{6.6}$$

it is possible to substitute equations (6.4) and (6.5) in the angular rate error equation (6.3), obtaining:

$$\dot{R}_e = (R_d \hat{\omega}_d)^T + R_d^T (R\hat{\omega}) = -\hat{\omega}_d R_d^T R + R_d^T R \hat{\omega}. \tag{6.7}$$

Finally it is possible to substitute equation (6.1) in equation (6.7) to get:

$$\dot{R}_e = -\hat{\omega}_d R_e + R_e \hat{\omega}. \tag{6.8}$$

Then, equation (6.8) can be written as

$$\dot{R}_e = R_e(\hat{\omega} - R_e^T \hat{\omega}_d R_e) = R_e \hat{e}_w, \tag{6.9}$$

in which the right angular velocity error is defined:

$$\hat{e}_w = \hat{\omega} - R_e^T \hat{\omega}_d R_e. \tag{6.10}$$

The final equation of angular rate error can be obtained by exploiting the following relation:

$$(R_e^T \hat{\omega}_d R_e)^\vee = R_e \omega_d, \tag{6.11}$$

so that

$$e_\omega = \omega - R_e^T \omega_d. \tag{6.12}$$

Another important definition, that will be used in the following, is the error navigation function. It can be used to extend the concept of distance to SO(3):

$$\Psi = \frac{1}{2}\mathrm{tr}(K_R(I_3 - R_e)) \tag{6.13}$$

where I_3 is the 3×3 identity matrix and K_R is a positive definite gain matrix:

$$K_R = \begin{bmatrix} K_{R_1} & 0 & 0 \\ 0 & K_{R_2} & 0 \\ 0 & 0 & K_{R_3} \end{bmatrix}. \tag{6.14}$$

The main properties of the error navigation function are:

1. Ψ is locally positive definite about $R_e = I_{3\times 3}$
2. the derivative of Ψ along the attitude error kinematics reads:

$$\dot{\Psi}(R_e) = e_R^T e_w \tag{6.15}$$

$$e_r = \mathrm{skew}(KR_e)^\vee = \frac{(KR_e - R_e^T K)^\vee}{2} \tag{6.16}$$

where e_R is the gradient of Ψ.

6.1.2 Control law

Let us recall the differential equations describing the rotational motion of a rigid body:

$$\begin{cases} \dot{R} = R\hat{\omega} \\ J\dot{\omega} + \hat{\omega} J\omega = \tau \end{cases} \tag{6.17}$$

Once the dynamics equations have been defined, the geometric control law is derived following a Lyapunov approach. To introduce the error dynamics, consider the following change of variables:

$$(R, \omega) \to (R_e, e_\omega = \omega - R_e^T \omega_d) \tag{6.18}$$

so the dynamic system (6.17) can be rewritten deriving R_e as shown in equation (6.9):

$$\dot{R}_e = R_e \hat{e}_\omega \tag{6.19}$$

deriving the angular rate error defined in equation (6.12):

$$J\dot{e}_\omega = J\dot{\omega} - J\dot{R_e}^T\omega_d - JR_e^T\dot{\omega}_d. \tag{6.20}$$

Exploiting the second equation of the (6.17), equation (6.20) can be rewritten as:

$$J\dot{e}_\omega = -\hat{\omega}J\omega + \tau + J\hat{e}\omega R_e^T\omega_d - JR_e^T\dot{\omega}_d. \tag{6.21}$$

Taking into account the change of variables and replacing equation (6.12), the final form is:

$$J\dot{e}_\omega = -(\hat{e}_\omega + R_e^T\hat{\omega}_d R_e)J(e_\omega + R_e^T\omega_d) + \tau + J\hat{e}_\omega R_e^T\omega_d - JR_e^T\dot{\omega}_d. \tag{6.22}$$

The next step is to define the Lyapunov function as the sum of the kinetic energy associated with the angular velocity error and the error navigation function (6.14):

$$V(R_e, e_\omega) = \frac{1}{2}e_\omega^T J e_\omega + \Psi(R_e). \tag{6.23}$$

By computing the time derivative of Ψ (6.15), one obtains:

$$\dot{V}(R_e, e_\omega) = e_\omega^T J\dot{e}_\omega + e_r^T e_\omega. \tag{6.24}$$

By substituting (6.20) in (6.24):

$$\dot{V}(R_e, e_\omega) = e_\omega^T(J\dot{\omega} - J\dot{R}_e^T\omega_d - JR_e^T\dot{\omega}_d + e_R) \tag{6.25}$$

the final equation is obtained:

$$\dot{V}(R_e, e_\omega) = e_\omega^T(-\hat{\omega}J\omega + \tau - J\dot{E}_e^T\omega_d - JR_e^T\dot{\omega}_d + e_R). \tag{6.26}$$

By defining:

$$\tau = -e_R - K_\omega e_\omega - K_I e_I + JR_e^T\dot{\omega}_d + (R_e^T\omega_d)^\wedge JR_e^T\omega_d \tag{6.27}$$

the $K_I e_I$ integral term was introduced, in order to compensate a constant disturbance torques. Unlike a linear PID control law, in which the the integral term is computed only on the attitude error, the integral term is computed both on attitude and angular velocity errors.

$$\dot{e}_I = e_R + K_{\omega I}e_\omega, \tag{6.28}$$

the closed-loop equilibrium $(R_e, e_\omega) = (I_3, 0)$ can be shown to be locally exponentially stable.

In conclusion, equations (6.27) and (6.28) represent the geometric control law, which includes all the consideration previously discussed.

6.2 Adaptive law

In this section the adaptive control law [7] system for a multirotor UAV is considered. The adaptive control is a valid solution to face the disturbances such as, *e.g.*, actuator degradation and faults, severe external disturbances, parameter uncertainties and time delays, because of learning whilst operating. The proposed approach allows to seamlessly combine a baseline linear controller (in this case a PID controller) with an adaptive one and to disable or enable the adaptive controller when needed, in order to take the advantages of both controllers. The adaptive control schema described in this thesis is Model Reference Adaptive Control (MRAC), which is the most widely known adaptive control technique, and will be formally described in the following.

6.2.1 Problem statement

The problem under study is to design an adaptive controller that can be seamlessly implemented in an already existing control architecture, capable of controlling the angular velocity dynamics of a multirotor UAV. For that purpose consider the Euler equations of rigid body angular motion, written in the principal inertial axes:

$$J_{xx}\dot{p} + (J_{zz} - J_{yy})qr = L \tag{6.29}$$
$$J_{yy}\dot{q} + (J_{xx} - J_{zz})pr = M \tag{6.30}$$
$$J_{zz}\dot{r} + (J_{yy} - J_{xx})pq = N, \tag{6.31}$$

where J_{xx}, J_{yy}, J_{zz} are the principal moments of inertia, $M_{ext} = [L \quad M \quad N]^T$ represent the external moment applied on the quadrotor vehicle and $\omega = [p \quad q \quad r]^T$ is the body angular velocity. Letting $H_0 = diag(J_{zz} - J_{yy}, J_{xx} - J_{zz}, J_{yy} - J_{xx})$ and denoting with J_0 the inertia matrix, the Euler equations can be written as:

$$\dot{\omega} = J_0^{-1}(M_{ext} - H_0 f(\omega)) \tag{6.32}$$

where $f(\omega)$ can be defined as:

$$f(\omega) = [qr \quad pr \quad pq]^T. \tag{6.33}$$

External moments acting on the quadrotor can be decomposed into three categories: aerodynamic damping moment, moment due to propellers, and moment due to external disturbances. For what concerns the aerodynamic damping moment, we assume it to be proportional to the rates ω, hence given by $A\omega$, where A is a constant uncertain matrix. The moment due to the propellers is just the control action (output of the actuators), which will be indicated by g. Finally, external disturbances are denoted as σ. Therefore we can write:

$$M_{ext} = A\omega + g + \sigma \tag{6.34}$$

and letting $K = J_0^{-1}, H = KH_0$ equation (6.32) becomes:

$$\dot{\omega} = KA\omega + Kg + K\sigma + Hf(\omega). \tag{6.35}$$

When the quadrotor is hovering the term $f(\omega)$ is negligible, but uncertainties and disturbances are still acting on the system, which means that the matrices K, A, H are uncertain. Let now the subscript 0 denote nominal values, and the subscript δ the uncertainty, so the matrices A and H can be rewritten in the following way:

$$A = A_0 + A_\delta \tag{6.36}$$

$$H = K(H_0 + H_\delta). \tag{6.37}$$

For the K matrix it is better to express the uncertainty in multiplicative form:

$$K = K_0\Lambda_K \tag{6.38}$$

where Λ_K can be expressed as:

$$\Lambda_K = I_3 + K_0^{-1}K_\delta \tag{6.39}$$

so equation (6.32) can be rewritten considering the nominal and the uncertain values:

$$\dot{\omega} = K_0\Lambda_K((A_0 + A_\delta)\omega + g + \sigma) + (H_0 + H_\delta)f(\omega) \tag{6.40}$$

The idea of adaptive control is that the system has to operate mainly in nominal conditions and to have the adaptation active only when necessary. To that purpose let the control input u be given as:

$$u = u_b + u_a \tag{6.41}$$

where u_b is the control action provided by the baseline controller, while u_a is the contribution to the control action given by the adaptive controller. Similarly, let $g = g_b + g_a$ where $g_b = G(s)u_b$ and $g_a = G(s)y_a$. For the purpose of this thesis the MRAC attitude control will be explained in the following.

6.2.2 MRAC design of attitude control

Plant model

In the plant model:

$$\dot{\omega} = K_0\Lambda_K((A_0 + A_\delta)\omega + g_b + g_a + \sigma) + (H_0 + H_\delta)f(\omega) \tag{6.42}$$

the nominal part is given by:

$$K_0(A_0\omega + g_b). \tag{6.43}$$

By adding and subtracting $K_0A_0\omega + K_0g_b$ to the right-hand side of (6.42) we get:

$$\begin{aligned}\dot{\omega} &= K_0\Lambda_K((A_0 + A_\delta)\omega + g_b + g_a + \sigma) + (H_0 + H_\delta)f(\omega) \pm K_0(A_0\omega + g_b) \\ &= K_0(A_0\omega + g_b) + K_0\Lambda_K(\alpha_1\omega + \alpha_2 g_b + g_a + \sigma + \alpha_3 f(\omega))\end{aligned} \tag{6.44}$$

where

$$\alpha_1 = (J - \Lambda_K^{-1})A_0 + A_\delta \tag{6.45}$$
$$\alpha_2 = J - \Lambda_K^{-1} \tag{6.46}$$
$$\alpha_3 = (K_0\Lambda_K)^{-1}(H_0 + H_\delta). \tag{6.47}$$

notice that Λ_K^{-1} always exists since:

$$\Lambda_K = I_3 + K_0^{-1}K_\delta > 0 \tag{6.48}$$

as K_0 is positive definite and K_δ is positive semi-definite.

Control law

The adaptive control law u_a is defined as:

$$u_a = -\hat{\alpha_1}\omega - \hat{\alpha_2}g_b - \hat{\sigma} - \hat{\alpha_3}f(\omega) \tag{6.49}$$

where $\hat{\alpha_1}$ is the estimate of α_1, $\hat{\alpha_2}$ is the estimate of α_2, $\hat{\sigma}$ is the estimate of σ and finally $\hat{\alpha_3}$ is the estimate of α_3. Hence, letting:

$$\Delta\alpha_i = \hat{\alpha_i} - \alpha_i, \quad i = 1, 2, 3 \tag{6.50}$$
$$\Delta\sigma = \hat{\sigma} - \sigma \tag{6.51}$$

we get:

$$\dot{\omega} = K_0(A_0\omega + g_b) - K_n\Lambda_K(\Delta\alpha_1\omega + \Delta\alpha_2 g_b + \Delta\sigma + \Delta\alpha_3 f(\omega)) \tag{6.52}$$

where the assumption that $g_a \approx u_a$ has been used for small adaptive gains.

Observer-like reference model and error dynamics

We now build an observer of the nominal part of the plant in the following way:

$$\dot{\hat{\omega}} = K_0(A_0\hat{\omega} + g_b) + Le \tag{6.53}$$

where:

$$e = \hat{\omega} - \omega \tag{6.54}$$

and L is a Hurwitz matrix, added to assign the error dynamics, given by:

$$\dot{e} = (K_0A_0 + L)e + K_0\Lambda_K(\Delta\alpha_1\omega + \Delta\alpha_2 g_b + \Delta\sigma + \Delta\alpha_3 f(\omega)). \tag{6.55}$$

Matrix L can be chosen in many ways. Notice that high values for L may result in high-gain feedback in the observer loop, which may cause undesired phenomena, such as peaking.

Adaptive laws

Based on the control law and on the error equation, the following adaptive laws can be deduced:

$$\dot{\hat{\alpha}_1} = \text{Proj}(\hat{\alpha_1}, -\Gamma_1 \omega e^T PB) \tag{6.56}$$

$$\dot{\hat{\alpha}_2} = \text{Proj}(\hat{\alpha_2}, -\Gamma_2 g_b PB) \tag{6.57}$$

$$\dot{\hat{\alpha}_3} = \text{Proj}(\hat{\alpha_3}, -\Gamma_3 f(\omega) e^T PB) \tag{6.58}$$

$$\dot{\hat{\sigma}} = \text{Proj}(\hat{\sigma}, -\Gamma_4 e^T PB) \tag{6.59}$$

where $\text{Proj}(\cdot, \cdot)$ is the projection operator and the initial conditions are set to $\hat{\alpha_1}(0) = \hat{\alpha_2}(0)\hat{\sigma}(0) = 0$ and $\hat{\alpha_3}(0) = K_0^{-1} H_0$. The symmetric matrix P is chosen so to satisfy the Lyapunov equation:

$$A_e^T P + P A_e = -Q, \quad A_e K_0 A_0 + L, \tag{6.60}$$

where $Q = Q^T > 0$ was chosen as $Q = J$.

6.3 Implementation issues

6.3.1 Geometric control law implementation

The theoretical formulas of the nonlinear geometric control law just reported, have been reported in Simulink and in particular was implemented in the Simulink template described in Section (4.4.3) which defines the I/O interfaces required by the modified version of PX4.

The control torque in equation (6.27) requires both the desired angular velocity and acceleration to be implemented. This means that the position controller should provide not only a desired attitude R_d but ω_d and $\dot{\omega}_d$ as well and this not only makes the controller structure more complex but has potentially negative effects when only a roughly estimated inertia matrix is available. Furthermore, in case the vehicle is manually piloted, the pilot sends commands in terms of desired angles to the on-board controller and the corresponding angular velocity and acceleration must be somehow computed on-line, unlike the scenario in which both the attitude and its derivatives are provided, for instance, by a ground control computer. One option is setting $\omega_d, \dot{\omega}_d = 0$, namely, set-point tracking. This, however, represents a limitation in achieving the desired performance in terms of attitude response to reference, resulting in a sluggish response.

Linearized geometric control law

During the tuning phase, working directly with the gains of equation (6.27) is hard. Since this is a nonlinear control law, there is no clear understanding of the

effect of changing different gains. Indeed, by referring to the linearized closed-loop obtained for$R_e \approx I_3 + \hat{\theta}_e$, $e_\omega \approx \omega - \omega_d =: \omega_e$ and $\|\omega_d\| \ll 1$, *i.e.*,

$$s\theta_e = \omega_e \tag{6.61}$$

$$sJ\omega_e = -K_\omega \omega_e - \tilde{K}_R \theta_e - K_I e_I + \tau_e \tag{6.62}$$

$$se_I = \tilde{K}_R \theta_e + K_{\omega_I} \omega_e \tag{6.63}$$

in which $\tilde{K}_R$ is defined as:

$$\tilde{K}_R = \begin{bmatrix} \frac{K_R^{22}+K_R^{33}}{2} & 0 & 0 \\ 0 & \frac{K_R^{11}+K_R^{33}}{2} & 0 \\ 0 & 0 & \frac{K_R^{11}+K_R^{22}}{2} \end{bmatrix}, \tag{6.64}$$

one sees that there is a non-trivial relationship between the gains in K_R and the proportional gains $\tilde{K}_R$ (connected to roll, pitch and yaw angles) in the linearized control law.

The characteristic polynomial of the linearized closed-loop system can be written as:

$$s^3 J + s^2 K_\omega + s(\tilde{K}_R + K_I K_{\omega_I}) + K_I \tilde{K}_R = 0 \tag{6.65}$$

in which τ_e can be neglected since it is an exogenous signal and does not contribute to the stability of the closed-loop system. At this point a viable approach to select the gains of the diagonal matrices is to make $s^3 J^{ii} + s^2 K_\omega^{ii} + s(\tilde{K}_R^{ii} + K_I^{ii} K_{\omega_I}^{ii}) + K_I^{ii} \tilde{K}_R^{ii}$ a Hurwitz polynomial for $i = 1, 2, 3$. A different path is to exploit the knowledge of identified models G_{ω_i}, and try to tune the gains in order to achieve satisfactory performance when referring to the complementary sensitivity function from θ_d to θ, given by:

$$T_s(s) = \frac{G_s(s)}{s + G_s(s)} \tag{6.66}$$

where $G_s(s) := \frac{G_{\omega_i}}{s^2}\left(K_\omega s^2 + s(\tilde{K}_R + K_I K_{\omega_I}) + K_I \tilde{K}_R\right)$. Besides the approximations already introduced in the linearized closed-loop (6.61)-(6.62), $\theta = T_s(s)\theta_d$ is valid when the desired attitude motion is not large, namely, when $R_d(t) \approx I_3 + \hat{\theta}_d(t)$, or when it is occurring mainly about one axis.

The result of the Simulink implementation of the nonlinear control law is reported in Figure 6.1.

6.3.2 Adaptive control law implementation

Baseline controller

As mentioned in the previous section, the presented adaptive attitude control law allows to combine a baseline controller with an adaptive one, and to disable

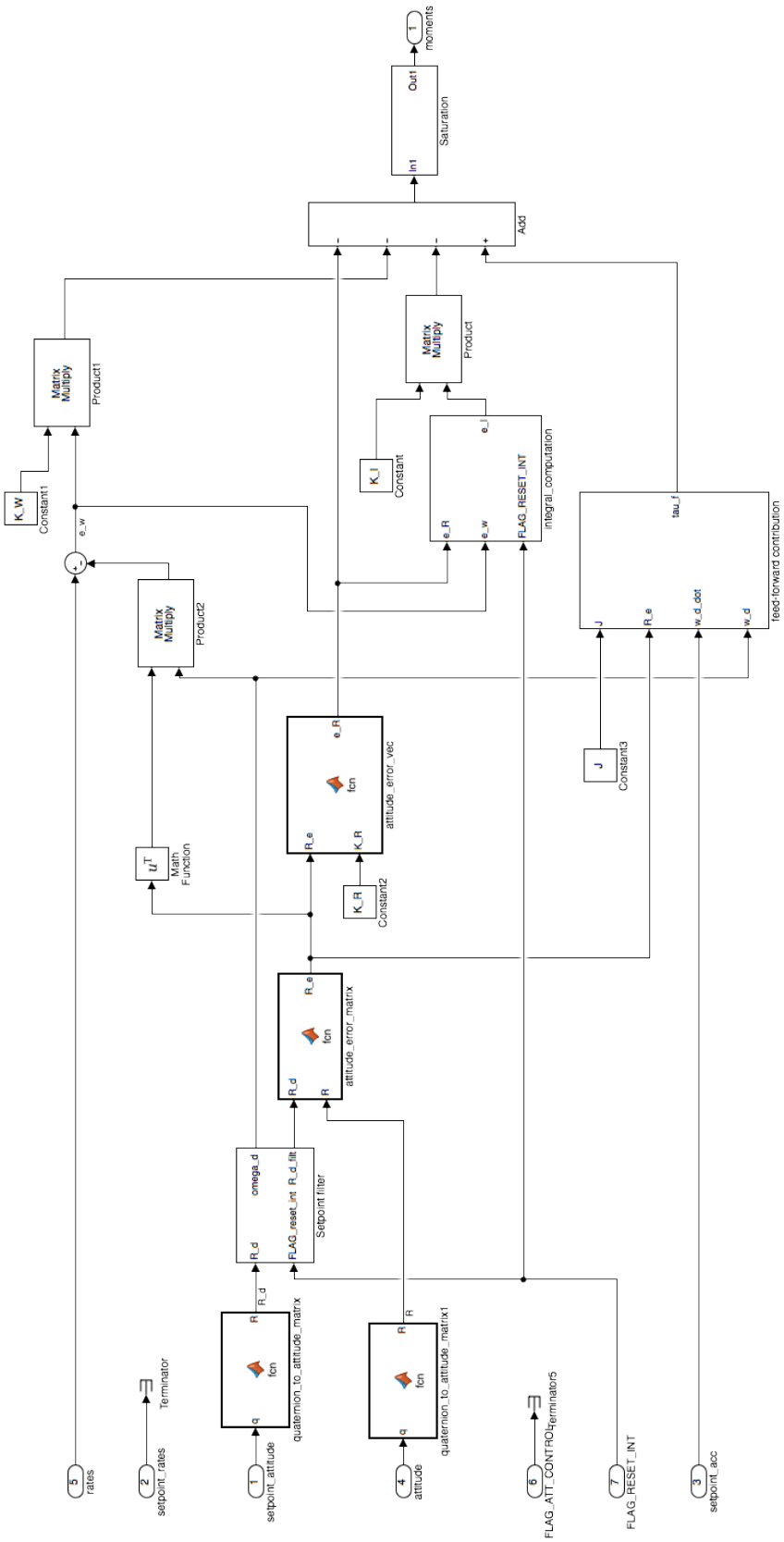


Figure 6.1: Nonlinear geometric control law implemented in Simulink

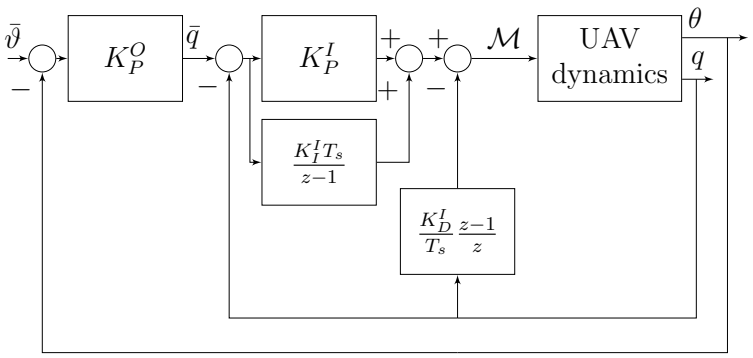


Figure 6.2: baseline attitude controller

or enable the adaptive controller when needed, in order to take the advantages of both controllers.

The baseline controller, has a cascade architecture as shown in Figure 6.2. For each axis, a PID inner loop tracks the desired angular rate, which is fed by a proportional outer loop on the attitude angle.

In particular the outer regulator (indicated with superscript ^{O}) is a P controller, while the inner regulator (indicated with ^{I}) is a PID controller. The baseline controller has been tuned using the loop shaping process explained in Section 5.3.5 and also the gains value for the three axes are reported.

MRAC gains

Since there is no systematic way to tune the MRAC gains, both the bounds and the gains values were tuned by trial and error. Figure 6.3 represents the adaptive control law implemented in Simulink.

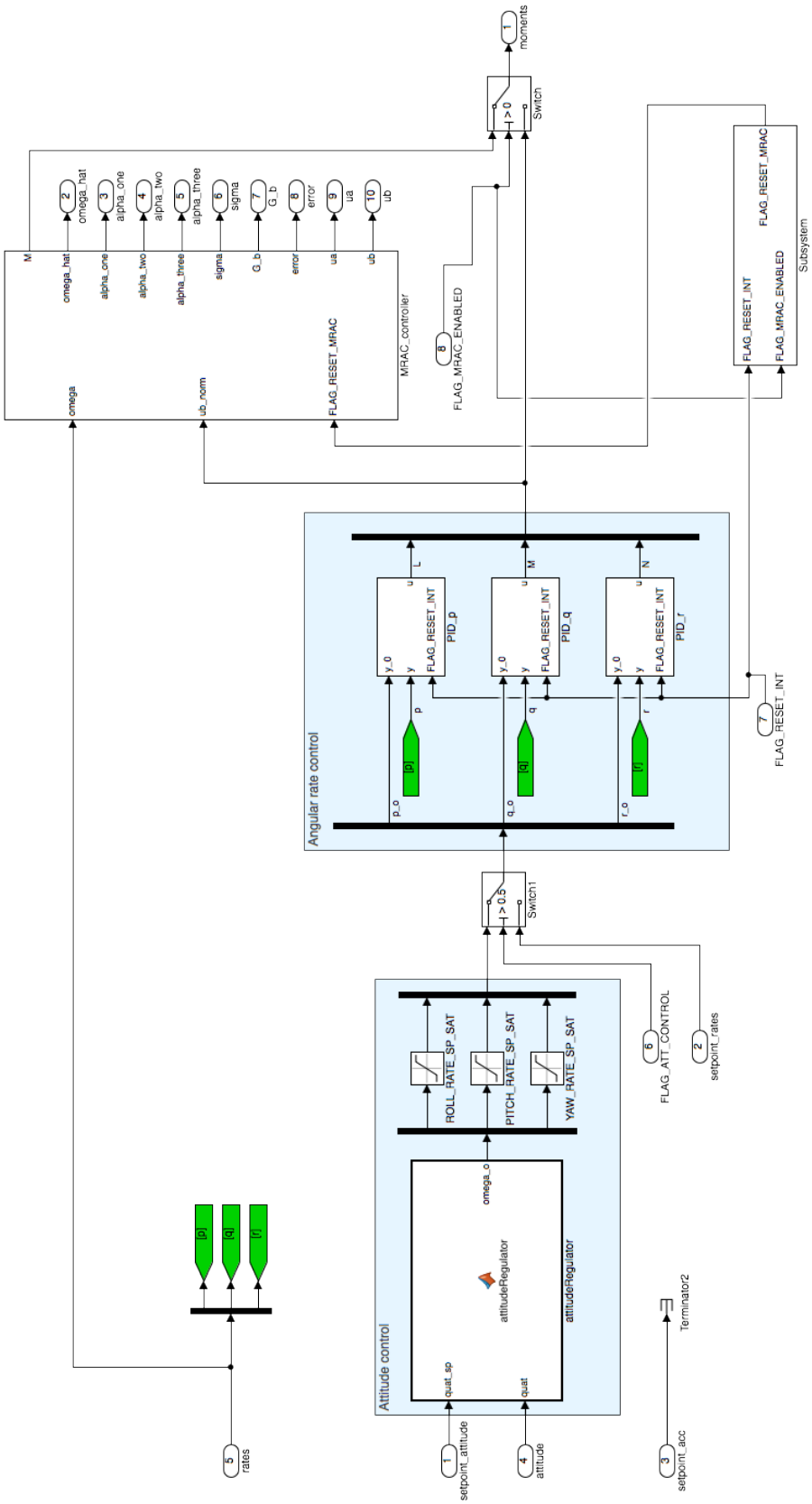


Figure 6.3: Adaptive control law implemented in Simulink

Chapter 7

Experimental results

This chapter contains the experimental results obtained comparing the advanced attitude control laws described in the previous chapter (geometric and adaptive control law) with the baseline attitude control law (cascade PID architecture). Since they are executed on the FCU, the percentage of resource used (CPU and RAM) by the PixFalcon to support these computations will be also evaluated.

The quadrotor considered for the experimental part is the ANT-1 drone with the hardware modification reported in Chapter 3.

7.1 Nonlinear geometric control law experimental results

This section is dedicated to describe the experimental results of the geometric control law that has been proposed to tackle the attitude tracking problem for rigid bodies. First a reference signal related issues is explained and then nonlinear geometric architecture is compared with the cascaded PID architecture. To a better comparison the two architectures have been tuned using the same H_∞ approach.

7.1.1 Reference signal related issues

In Section 6.3.1 it has been pointed out that the geometric architecture requires the desired attitude R_d and at least the corresponding angular velocity ω_d in the approximated case which neglects the feed-forward contribution in 6.27. A smooth trajectory generator is therefore needed to provide with those information the controllers and make a fair comparison between the PID and nonlinear geometric architectures. The trajectory generator has been implemented on-board (see Figure 6.1) in the form of a filter so that the existing software architecture should not be modified: the desired attitude trajectory is passed through a second-order

geometric filter from which a continuously differentiable signal and its derivative can be provided to the controllers. The following filter has been used:

$$\dot{R}_d^f = R_d^f \hat{\omega}_d^f \tag{7.1}$$

$$\dot{\omega}_d^f = -\omega_n^2 e_R(R_e^f) - 2\xi\omega_n\omega_d^f \tag{7.2}$$

where $R_e^f := R_d^T R_d^f \in \mathrm{SO}(3)$ and $e_R(\cdot)$ is defined as (6.16). Note that the filter approximates a linear second-order filter of the form $\ddot{\theta}_d^f = -\omega_n^2(\theta_d^f - \theta_d) - 2\xi\omega_n\dot{\theta}_d^f$ for a small attitude motion $R_d(t) \approx I_3 + \hat{\theta}_d(t)$.

7.1.2 Tuning

The linearized version of the two control architecture presented in Section 6.3.1 is used to carry out tuning of the gains. The tuned gains are then plugged in the nonlinear control law architecture for validation in the time domain. To synthesize, only the results related to the pitch axis will be presented.

Tuning was carried out by means of the structured H_∞ approach (see [8]). This approach allows to tune the parameters of a fixed-structure controller, so as to meet requirements encoded in the frequency domain; moreover, it guarantees robustness against model uncertainty. The approach proved to be of great practical usefulness and has been widely used in aerospace applications in recent years, in particular on rotorcraft. For the purpose of this experiment, both the PID and linearized geometric control laws have been tuned using H_∞ approach, and two different requirements were taken into account:

- performance: the requirement on performance is defined as a weighting function on the attitude angle sensitivity, *i.e.*, the transfer function from the disturbance on the pitch angle d_θ to θ;
- control moderation: the requirement on control moderation is defined as a weighting function on the control sensitivity, *i.e.*, the transfer function from d_θ to M.

7.1.3 Numerical results

Three different control law gainsets were obtained for the cascaded PID architecture:

- a *high-bandwidth* PID cascaded control law, denoted as C_H, which is subject to the H_{BW} performance requirement defined in Section 7.1.2;
- a *low-bandwidth* PID cascaded control law, denoted as C_L, subject to the L_{BW} performance requirement defined in Section 7.1.2;

Table 7.1: Cascaded PID control architecture gain values.

Gain	C_H	C_L	C_L^{D0}
K_P^I	0.142	0.187	0.0646
K_I^I	0.287	0.362	0.107
K_D^I	0.00263	0.00344	0
K_P^O	12.5	3.30	3.10
ω_{BW}[rad/s]	10	2.98	2.26

Table 7.2: Linearized geometric control architecture gain values.

Gain	G_L
K_P	0.264
K_D	0.0757
K_I	0.3
K_{wI}	0.0390
ω_{BW}[rad/s]	2.85

- a *low-bandwidth* PID cascaded control law with no derivative action in the inner loop, denoted as C_L^{D0}, which is subject to the same L_{BW} performance requirement but with the additional constraint $k_D^I = 0$.

Numerical optimization was carried out by means of the MATLAB `systune` routine. The gain values for the three cascaded PID control laws are shown in Table 7.1, along with the achieved sensitivity bandwidth.

Finally, gains for the linearized geometric architecture were tuned according to the L_{BW} performance requirement and are stated in Table 7.2. This control law is denoted as G_L.

7.1.4 Experiment design

In order to compare the two proposed control architecture, three different experiments were designed to test the closed-loop response to the attitude angle reference:

- a series of doublet signals at small attitude angle, to test the system behaviour close to the equilibrium condition;
- a series of doublet signals at large attitude angle, to test the system behaviour significantly far from the equilibrium;
- a sweep signal to excite the closed-loop system within the control bandwidth, and to check the tracking performance.

The doublet half-period is $T = 0.5[s]$, with an amplitude of $A = 10[deg]$ for the small-amplitude experiment and $A = 30[deg]$ for the large-amplitude experiment; the sweep signal has an amplitude $A = 10[deg]$ and excites the system in the frequency range $[0.225, 2.25][Hz]$.

The tests were conducted with the quadrotor placed on a test-bed, constraining all the degrees of freedom except for the rotational motion about the pitch axis.

Figure 7.1 shows a comparison between the attitude angle responses of the system closed in loop with the C_H and C_L controllers to the large-amplitude doublet input. The C_H controller achieves a quicker response than C_L, at the price of slight oscillations in the response, which can be interpreted in the frequency domain as a trade-off between bandwidth and damping ratio.

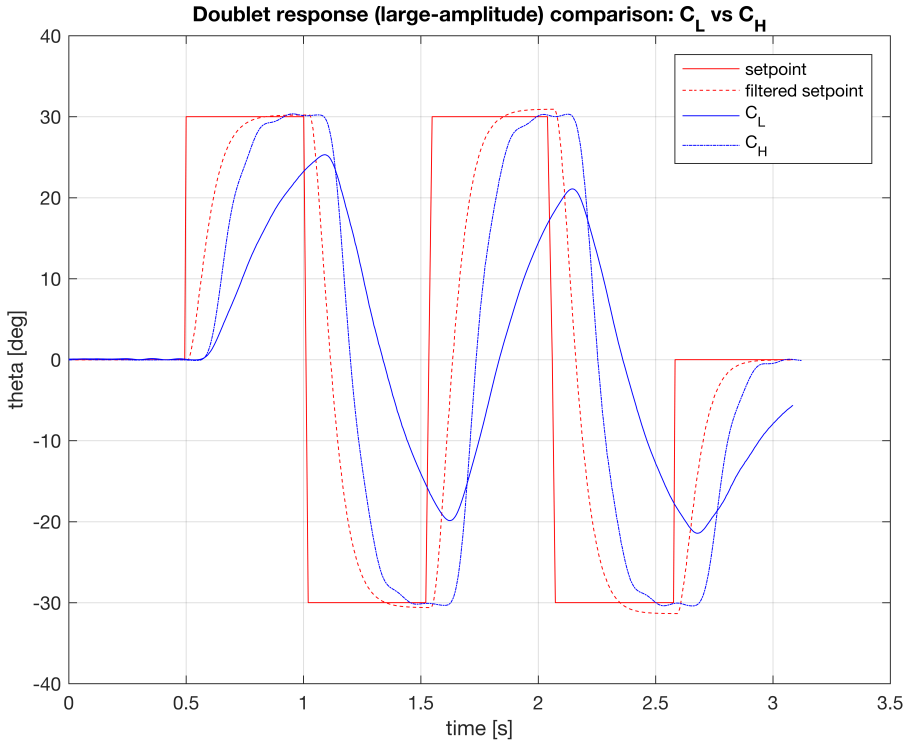


Figure 7.1: Comparison of doublet responses: controllers C_H and C_L.

Figure 7.2 shows a comparison of doublet responses between C_L^{D0} and G_L; these two controllers are comparable from the point of view of bandwidth and do not feature any derivative action on attitude rate; however, the response of G_L is more aggressive and presents many oscillations.

The test results, conducted at large amplitude angle, showed an attitude response comparable to the small amplitude experiments (not shown for brevity reasons); this confirms that the nonlinear control law is able to handle large displacements from the equilibrium condition without impairing performance.

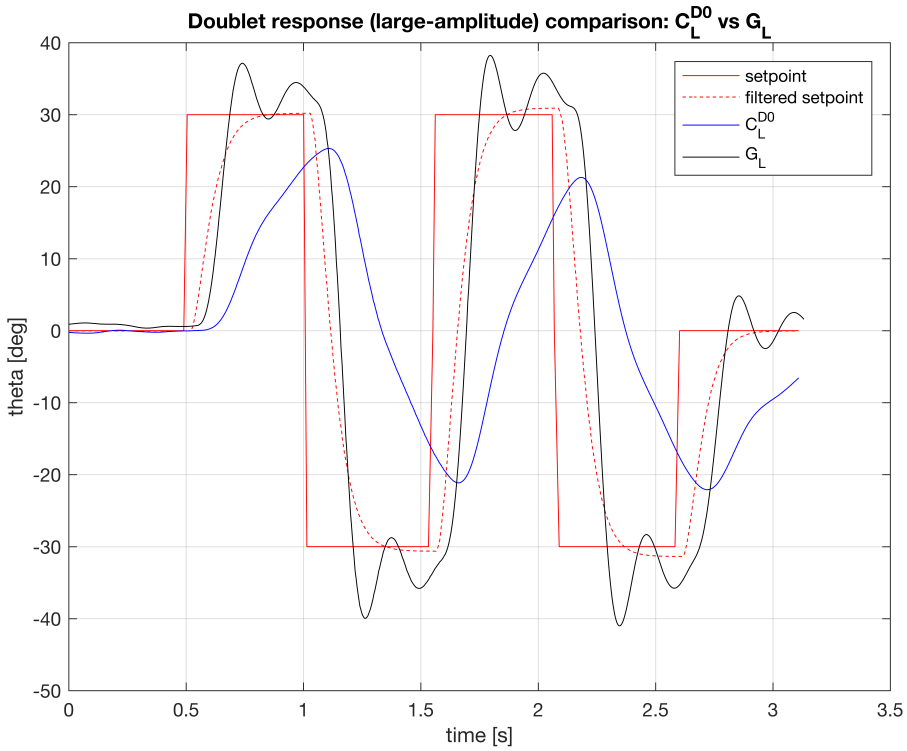


Figure 7.2: Comparison of doublet responses: controllers C_L^{D0} vs G_L.

Figure 7.3 shows the response of the G_L controller to the sweep reference signal. It can be noticed that, in the initial time instants, the tracking is excellent; as the sweep frequency increases with time, however, an unexpected effect of resonance shows up, resulting in an attitude response even larger than the reference signal.

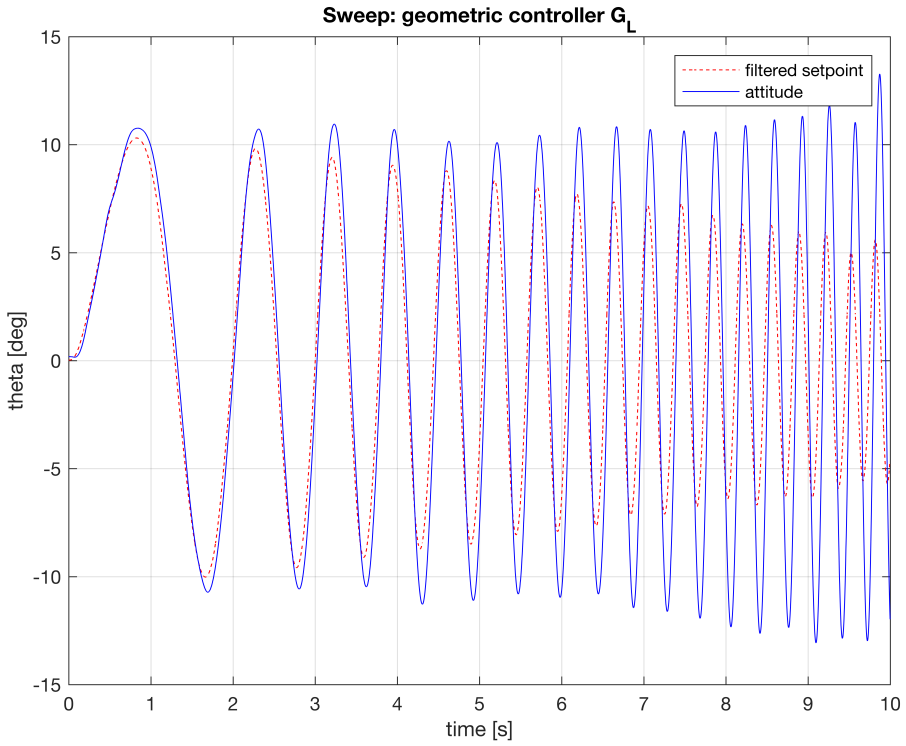


Figure 7.3: Sweep response: controller G_L

Figure 7.4 shows a comparison between the sweep responses of the G_L and C_L^{D0} controllers, with focus on the initial time instants: the nonlinear geometric architecture achieves better tracking than the cascade PID architecture, both in terms of amplitude of the response and limited phase lag with respect to the reference signal. On the other hand, the cascaded architecture features a significant lag in the response, even at low frequencies, and a quick decrease in peak magnitude as the frequency increases.

7.1.5 Conclusions

In this section a non geometric control law has been implemented on top of an existing controller was proposed. Since the geometric architecture requires the desired attitude R_d and at least the corresponding angular velocity ω_d, an on-board filter has been implemented as trajectory generator. In conclusion flight tests were carried out on the unconstrained quadrotor; the vehicle was tested in

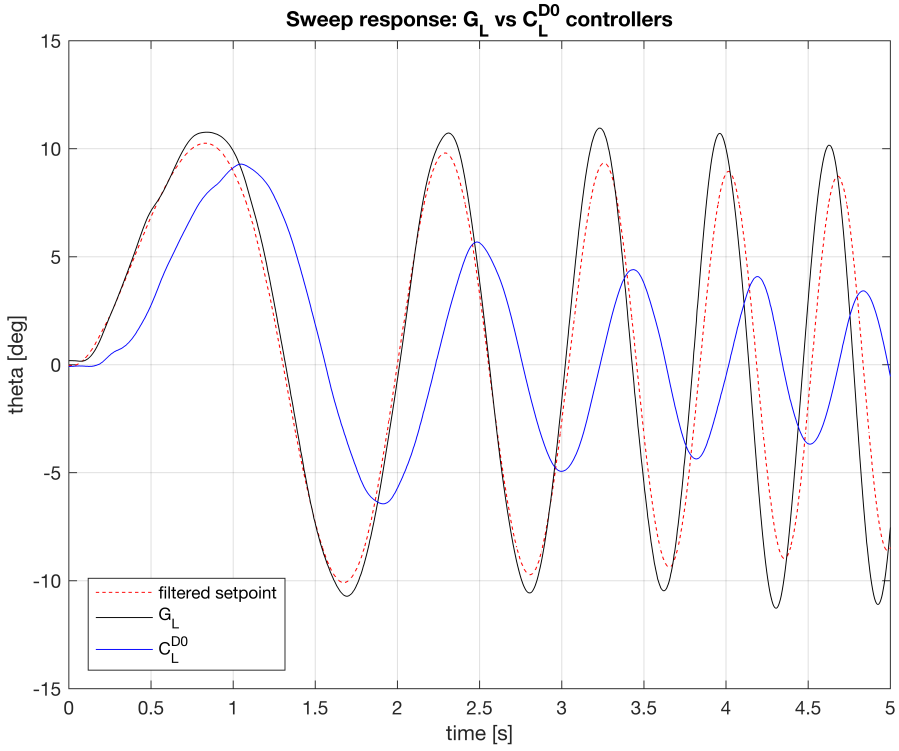


Figure 7.4: Sweep response: controller G_L

Coefficient	Value
α_1	10^{-2}
α_2	10^{-1}
α_3	1
λ	0.3
σ	0.2

Table 7.3: Bound on the adaption law coefficients

Coefficient	MRAC gains
α_1	10^{-3}
α_2	10^{-1}
α_3	1
λ	–
σ	2×10^{-2}

Table 7.4: MRAC gains

attitude control, with the pilot commanding attitude angles. The tests confirmed the results obtained on test-bed, with all the controllers guaranteeing stability and acceptable maneuverability from the pilot's point of view.

7.2 Adaptive control law experimental results

This section is dedicated to describe the experimental result obtained by testing the adaptive control law in a flight session and comparing this architecture with the cascade PID control law in the same experiment.

As described is the previous chapter, the cascade PID controller architecture was used as a baseline to show the performance improvement of the adaptive law in term of capability to reject disturbances. The adaptive control law was implemented and validated on all the three axes of roll, pitch and yaw. To synthesize, only the results related to the pitch axis will be presented.

Since there is no systematic way to tune the MRAC gains, both the bounds and the gain values were tuned by trial and error. For these experimental results, the bounds and the gains on the adaptation coefficients are reported respectively in Table 7.3 and Table 7.4.

7.2.1 Experiment design

The experiment was designed to verify the ability of the adaptive control law to quickly recover level attitude when a disturbance occurs on the actuators. Such experiment consisted in the injection of a step disturbance on the pitch axis while

Control law	$t_R[s]$	$e_{PK}[deg]$	$e_{RMS}[deg]$
baseline	1.496	15.3	6.19
MRAC	0.340	11.6	3.21
	(-77.3%)	(-24.1%)	(-48.2%)

Table 7.5: Performance metrics: pitch response to input disturbance (in brackets the relative reduction with respect to baseline).

the drone was hovering. The disturbance (with an amplitude of half the maximum available control moment $\overline{m}$) is a step on the pitch axis summed to the control moment u. Supposing the step disturbance Δu defined as:

$$\Delta u = [\quad 0 \quad 0.5\overline{m} \quad 0 \quad]^T \tag{7.3}$$

so the actual commanded actuator control action is u_{cmd}:

$$u_{cmd} = u + \Delta u. \tag{7.4}$$

The experiment is representative of the effect of a (partial or total) loss of throttle on one or more motors, as this would produce a loss of thrust, which in turns produces a disturbance moment about the body axes of the quadrotor.

Figure 7.5 shows the pitch angle and pitch angular rate responses to the disturbance on the pitch axis injected at time instant $t = 0$. Analyzing the obtained results obtained, it is possible to notice that the adaptive control law has a better performance than the baseline law (cascade PID) in term of attitude angle disturbance reaction time t_R , which is defined as the time needed to recover the initial attitude angle θ_0 within a range of $\pm 1.5deg$. The disturbance reaction times for the two control laws are reported in Table 7.5, along with the peak e_{PK} and the RMS e_{RMS} of the error e_θ, where e_θ is defined as the difference between the pitch angle θ and its value θ_0 before the disturbance injection:

$$e_\theta = \theta - \theta_0. \tag{7.5}$$

In the two cases it can be noticed that the pitch angle undergoes an initial sudden variation and then the control comes into play and the pitch angle returns to the initial value.

In the cascade PID architecture, the angle slowly returns to zero with the attitude remaining always positive, while the the adaptive law the attitude quickly return to zero and during the transient also reaches negative values. This is an interesting "braking" effect of the quadrotor which results in a small displacement from its initial position.

The adaptive law achieves a smaller peak and also the RMS of the error is reduced respect to the PID control law.

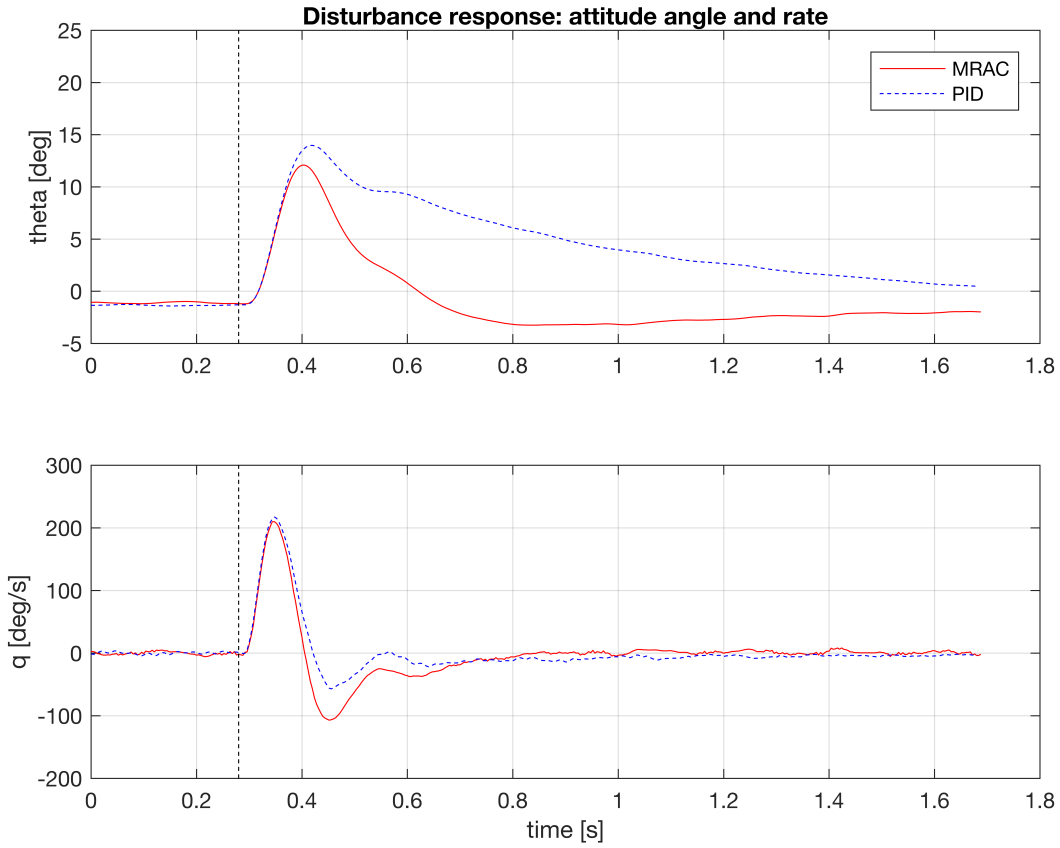


Figure 7.5: Disturbance response: attitude angle and rate

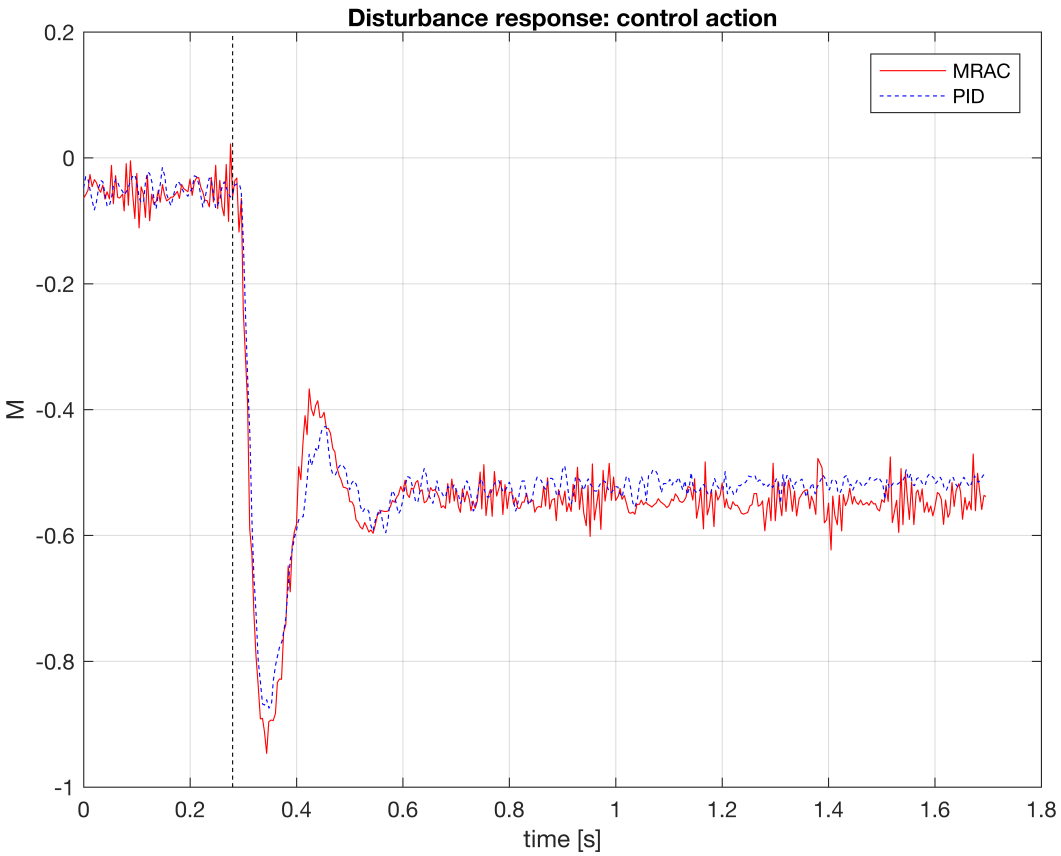


Figure 7.6: Disturbance response: control action

Control law	u_{RMS}
baseline	0.491
MRAC	0.509

Table 7.6: Performance metrics: pitch response to input disturbance (in brackets the relative reduction with respect to baseline).

Figure 7.6 shows the respective control actions for the two control laws. The control actions reported are dimensionless values, so it means that the value $m = 1$ indicates the maximum control moment which can be produced on the three axes.

The effect of adding the step disturbance downstream the controller can be noticed in that the steady-state value of the control action response counterbalances the disturbance (with equal magnitude and opposite value). Despite the results seem to be very similar, the two control actions actually produce very different results in term of pitch angle response. One difference between the PID and the adaptive laws can be seen in the steady-state behavior of the control action response, which has a slightly larger value in the case of the baseline law, resulting in the slow decrease of the attitude angle depicted in Figure 7.5. The RMS of the control action u_{RMS} for the two control laws is reported in Table 7.6. The RMS is similar in both cases, showing that the employed control energy is comparable.

The contribution of the MRAC is shown in Figure 7.7, where the control action has been split into its components u_a and u_b as expressed by equation (6.41). More precisely u_a is the control action produced by MRAC and u_b is the control action produce by the baseline control law (cascade PID).

7.2.2 Conclusions

In this section an adaptive control law that can be implemented on top of an existing controller was proposed. The adaptive law has shown improvements in terms of disturbance rejection performance, with only a small increase of the required control power. The proposed design of the adaptive controller showed superior performance with respect to the nominal controller when operated in the presence of significant disturbances and the experimental results confirmed the overall performance improvement with respect to the baseline controller.

In the following will be also considered the hardware resources consumed by the PID, geometric and adaptive control laws during their execution, in particular the CPU and RAM percentage used by the PixFalcon FCU will be showed.

7.3 FCU Performance evaluation

The last part of the system taken into account is the FCU of the ANT-1 drone, in particular the PixFalcon FCU. In the previous chapters, mainly three different

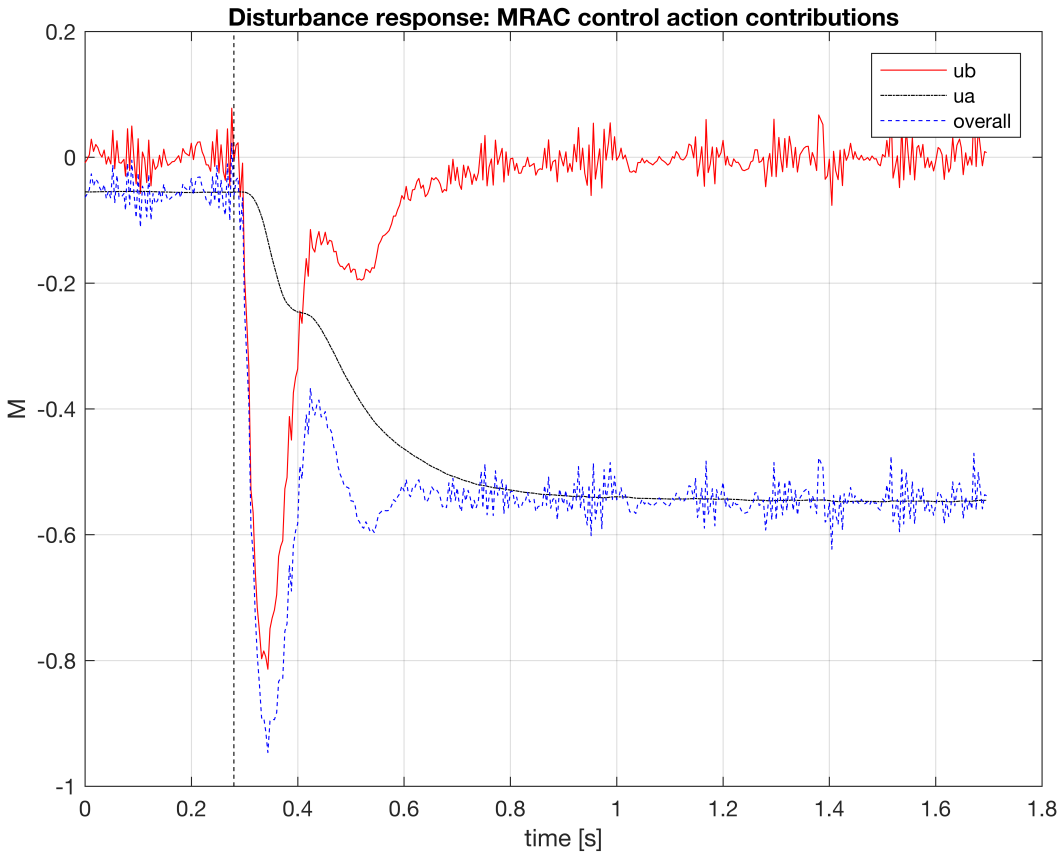


Figure 7.7: Disturbance response: MRAC control action contributions

attitude control laws have been explained. Since they are executed on the FCU and not on the companion (NanoPi), we would like to know the percentage of resources used, to avoid bottleneck if it occurs.

Thanks to the PX4 log system it was possible to retrieve information about the total CPU and RAM used by FCU in operative conditions. To get more specific information about single PX4 modules (in this case the "flight_controller" module) in terms of CPU and RAM load, a Python script was used to start the NuttX Shell terminal and retrieve the FCU information. In detail, the procedure followed was:

- connect the ANT-1 drone PixFalcon to the PC using a micro USB cable (remove the SD card if it is inserted);
- using the PC, got to the PX4 firmware folder and execute the following Python script `./Tools/mavlink_shell.py`;
- once the NuttX Shell terminal starts, stop the original attitude control law using the `mc_att_control` stop command, and start the modified attitude control low using `flight_controller` start command;
- at the end, to obtain the information about all the module currently running on the FCU, execute the command `top`, and a list of running modules with load details will appear.

Table 7.7 collects all the information retrieved from the FCU for each attitude control law considered in this thesis. The "Module name" field, indicates the name of the module in which the control law is contained. For example, the "PX4 control law" (which is a cascade PID attitude control law) is the default control law defined in the original PX4 firmware and more precisely in a module named "mc_att_control ". The remaining control laws are imported from Simulink into a module named "flight_controller" , which is contained in the modified firmware as described in the Chapter 4.

The RAM load for the single module, is a value expressed in Byte as given by the NuttX Shell terminal but since the PixFalcon total RAM is 256KB, the module RAM load is very low.

In conclusion the CPU load increases with the complexity of the attitude control law while the total RAM percentage is quite constant. Anyway the PixFalcon board does not cause a bottleneck in the ANT-1 drone system.

Attitude control law	Module name	CPU [%] (single module)	RAM [Byte] (single module)	CPU [%] (total)	RAM [%] (total)
PX4 control law (PID)	mc_att_control	2.85	1152	47.65	73.95
PID cascade structure	flight_controller	2.02	744	43.56	72.00
Geometric linearized	flight_controller	6.69	744	48.27	72.27
Geometric nonlinear	flight_controller	10.28	1328	55.28	73.26
Adaptive	flight_controller	9.41	1040	53.73	72.47

Table 7.7: PixFalcon FCU performance: CPU and RAM load for each attitude control law explained

Chapter 8

Concluding remarks

The purpose of this thesis was to design and educational drone that has the possibility to implement, test and support user-defined advanced control laws.

The conducted activities start in the third chapter with a system analysis in order to better understand the wireless communication latencies between the old hardware version of ANT-1 (Raspberry Pi Zero W companion) and the ground station.

In the fourth chapter, how the PX4 firmware was modified to integrate an attitude control law implemented in Simulink were explained. To simplify the process of importing control law in the firmware, a GUI Matlab application was created to automate the process of exporting C++ code from a Simulink model, into the correct firmware directory. At the end, the C++ code exported from Simulink and integrated in PX4 was validated by testing it in a flight session and checking if the control actions are in the expected order of magnitude.

In the fifth chapter, the formalism and the method used to define an attitude control law of multirotor UAVs were described and in the sixth chapter the nonlinear geometric and the adaptive architectures were reported.

The last chapter deals with the experimental results obtained in the flight sessions with cascade PID, nonlinear geometric and adaptive control laws and with the Flight Control Unit performance evaluation based on these three different architectures.

8.0.1 Further developments

The possible further developments of this project are:

- extend the PX4 code import functionality also for the position control law by defining a new PX4 module and defining a new Simulink I/O interface;
- Optimize network latencies by replacing the ROS mocap node with the Virtual Reality Peripheral Network (VRPN).

Bibliography

[1] M.Giurato. Design, integration and control of a multirotor UAV platform. Master's thesis, Politecnico di Milano, 2015.

[2] Ryze. Tello edu. `https://www.ryzerobotics.com/tello-edu?site=brandsite&from=landing_page`.

[3] Parrot. Parrot education. `https://edu.parrot.com`.

[4] MathWorks. Support package for parrot. `https://it.mathworks.com/hardware-support/parrot-minidrones.html`.

[5] MathWorks. Pixhawk pilot support package. `https://it.mathworks.com/hardware-support/pixhawk.html`.

[6] D.Chevallard. Design, identification and control of a micro aerial vehicle. Master's thesis, Politecnico di Milano, 2017.

[7] G. Bressan, A. Russo, D. Invernizzi, M. Giurato, S. Panza, and M. Lovera. Adaptive augmentation of the attitude control system for a multirotor uav. *Define*, 2018.

[8] P. Apkarian and D. Noll. Nonsmooth H_∞ synthesis. *IEEE Transactions on Automatic Control*, 51(1):71–86, 2006.