POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Automation Laboratory Report

Quadcopter

Anna Maggioni - 10667508

Matteo Mastromauro - 10680859

Camilla Pina - 10674687

Alex Zagati - 10658724

A.Y. 2023-2024

Contents

Contents **1**

1 Introduction **4**

2 Lab Setup **5**
2.1 Technical Features 5

3 Model development **6**
3.1 Angular Euler-Lagrange equations 8
3.2 Euler-Lagrange equations for linear motion 8
3.3 State–Space representation 9
3.3.1 2-dof State-Space representation 9
3.3.2 1-dof State-Space representation 10

4 Sensors **11**
4.1 Introduction 11
4.1.1 Sensor Interface 11
4.1.2 Sensors Calibration 11
4.1.3 Accelerometer Calibration 11
4.1.4 Gyroscope Calibration 12
4.1.5 Internal State Estimator 12
4.1.6 Noise Analysis 13
4.1.7 Bias & Variance Estimation 13

5 Parameters Estimation **14**
5.1 Mechanical Model 14
5.2 Inertia Tensor Estimation 14
5.2.1 Introduction 14
5.3 Thrust-PWM Relationship 17
5.3.1 Introduction 17
5.3.2 Dynamometer 17
5.3.3 Balance 18

6 Model Implementation **20**
6.1 Model linearization 20

7 Control in Simulation **21**
7.1 Linear Control – 3-dof Pole Placement 21
7.2 Linear Control - 3-dof H2 21

7.2.1 2-dof H2 . . . 23
7.2.2 1-dof H2 . . . 23
7.2.3 H2 Control - LMIs . . . 24
7.3 Nonlinear Control - 3-dof Inverse Dynamics . . . 25
7.3.1 Inverse Dynamics - Pole Placement . . . 25
7.3.2 Inverse Dynamics - PID . . . 26
7.4 Nonlinear Control - 1-dof VSC . . . 27

8 Observer **28**
8.1 Introduction . . . 28
8.2 Sensor Model and Reference Frame . . . 28
8.3 Extraction of g . . . 29
8.3.1 Low-Pass Filter . . . 29
8.3.2 Sliding Median Filter . . . 30
8.3.3 Sliding Average Filter . . . 31
8.4 Computation of the angles . . . 31
8.5 EKF's Algorithm . . . 32
8.5.1 System Model - Roll and Pitch . . . 32
8.5.2 System Model - Yaw . . . 33
8.6 Implementation . . . 34
8.7 Validation and Simulation . . . 35

9 Implementation on the Real System **37**
9.1 Tests Setup . . . 37
9.2 External disturbances . . . 37
9.3 1-dof PID Control . . . 37
9.3.1 1-dof PID Control - Roll . . . 38
9.3.2 1-dof PID Control - Pitch . . . 41
9.3.3 Allocation Logic . . . 42
9.3.4 L(s) analysis . . . 43
9.3.5 Closed loop Identification . . . 43
9.4 1-dof Pole Placement Control - Pitch . . . 44
9.4.1 Closed loop Identification . . . 47
9.5 1-dof H2 Control - Pitch . . . 47
9.5.1 Closed loop identification . . . 51

10 Final analysis and considerations **53**
10.1 Comparison between the implemented controllers . . . 53
10.2 Other controllers . . . 54
10.3 Challenges for 2-dof and 3-dof controllers on the real system . . . 54

List of Videos **55**

Bibliography **56**

1 Introduction

Quadcopter, also known as quadrotor, is a helicopter with four rotors. The rotors are directed upwards and they are placed in a square formation at equal distance from the center of mass of the quadcopter [4].

The purpose of this report is to model and control the attitude of X500V2 drone, constrained in its central part to a platform by a 3D joint.

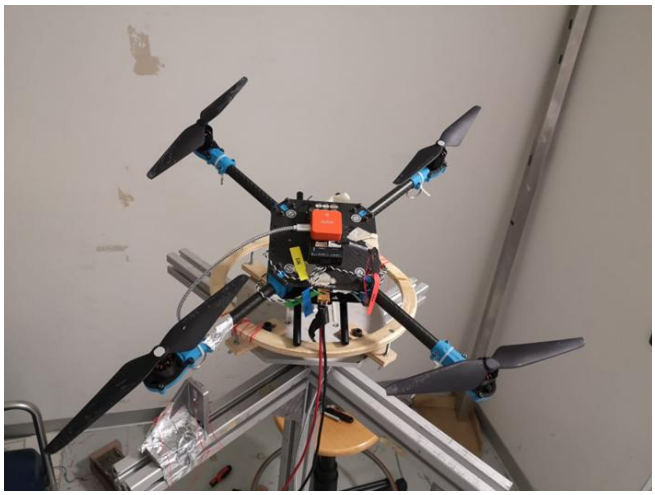

Figure 1.1: Quadcopter

This is obtained through two main steps:

- Model development + Identification of system parameters through a black box approach
- Control schemes implementation and verification on both the model and the real system

The report makes reference mainly to [4], with the adjustment of some equations and expressions to adapt them to this particular system.

2 Lab Setup

2.1 Technical Features

The quadcopter used during this experience is the X500 V2 Kits by Holybro [1]. Here below, the main features are summarized.

Motor Specification	Value
Idle Current (10V)	0.8A
Peak Current	17A
Max Power	272W
Rated Voltage	16V
Weight	64 ± 2g

Table 2.1: Motor specifications

Mechanical Specification	Value
Wheelbase	500mm
Material	Carbon Fiber
Material thickness	2mm
Weight	610g
Maximum Payload	1500g

Table 2.2: Mechanical specifications

Electronic Specification	Value
Processor	STM32H753VI (32bit ARM Cortex M7 - 400 MHz)
Failsafe co-processor	STM32F103 (32bit ARM Cortex-M3)
Sensors	(all connected via SPI)
Accelerometer	(3) ICM20948, ICM20649, ICM20602
Gyroscope	(3) ICM20948, ICM20649, ICM20602
Compass	(1) ICM20948
Barometric Pressure Sensor	(2) MS5611
Sensor Refresh Rate	200 Hz

Table 2.3: Electronic specifications

Power Specification	Value
External Power Supply	
Input voltage	4.1V - 5.7V
Rated input current	2.5A
Rated input/output power	14W

Table 2.4: Power specifications

3 Model development

The quadcopter structure is presented in Figure 3.1, including the corresponding angular velocities, torques and forces created by the four rotors (numbered from 1 to 4).

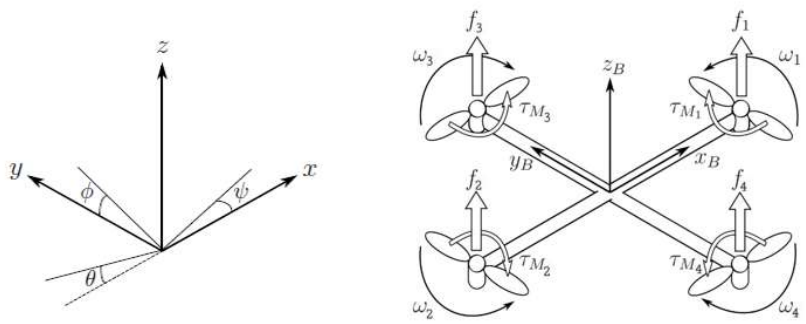

Figure 3.1: Quadcopter's structure and frames

The image shows also the two reference frames used in this work: the inertial and the body frame. The origin of both the terns is located in the quadcopter center of mass.

The system is characterized by 6 states: 3 Euler angles and their corresponding velocities, all considered in the inertial frame:

$$\boldsymbol{x} = \begin{bmatrix} \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}' \in \mathbb{R}^6 \tag{3.1}$$

where:

- Roll angle ϕ determines the rotation around the x-axis
- Pitch angle θ determines the rotation around the y-axis
- Yaw angle ψ determines the rotation around the z-axis
- $\dot{\phi}$ is the roll rate
- $\dot{\theta}$ is the pitch rate
- $\dot{\psi}$ is the yaw rate

Among these states, only the Euler angles measurements are available and so they represent the output of the system: $y \in \mathbb{R}^3$.

For what concerns the control inputs, they are three and correspond to torques τ_ϕ, τ_θ and τ_ψ. To obtain their expressions, other relationships need to be defined. For instance, the force f_i generated by the angular velocity ω_i along the direction of the i-th rotor axis is expressed as:

$$f_i = \tilde{k}_i \omega_i^2 \tag{3.2}$$

Then, the combined action of the rotor forces creates thrust T along the body z-axis. In

particular, T is defined as:

$$T = \sum_{i=4}^{4} f_i = \sum_{i=4}^{4} \tilde{k}_i \omega_i^2 \tag{3.3}$$

By applying the torque balance, it is now possible to claim that they depend on the angular velocities of the four rotors with the following relationship:

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} l(-\tilde{k}_2\omega_2^2 + \tilde{k}_4\omega_4^2) \\ l(-\tilde{k}_1\omega_1^2 + \tilde{k}_3\omega_3^2) \\ b(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2) \end{bmatrix} \in \mathbb{R}^3 \tag{3.4}$$

where l is the distance between a rotor and the COM of the body, $\tilde{k}_i$, i=1..4, are the lift constants, τ_{mi} , i=1..4, is the torque around the i-th rotor axis and is defined as:

$$\tau_{M_i} = b\omega_i^2 \tag{3.5}$$

with drag constant b.

All the variables defined so far are in the inertial frame. If quantities in the body frame are needed, the following matrix, which is the rotation matrix from the body frame to the inertial frame, can be used:

$$\boldsymbol{R} = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi C_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix} \tag{3.6}$$

where $S_x = sin(x)$ and $C_x = cos(x)$.

Lastly, since quadcopter structure is symmetric, with the arms aligned with the x and y axes, the inertia matrix $\boldsymbol{I}$ is diagonal with $I_{xx} = I_{yy}$:

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{3.7}$$

It is now possible to write quadcopter's dynamic equations. Since the linear and angular components do not depend on each other, they can be studied separately.

3.1 Angular Euler-Lagrange equations

The angular Euler-Lagrange equations are defined in [4] as:

$$\boldsymbol{\tau} = \boldsymbol{J}(\boldsymbol{\eta})\ddot{\boldsymbol{\eta}} + \boldsymbol{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})\dot{\boldsymbol{\eta}} \tag{3.8}$$

where:

- $\boldsymbol{\eta}$ is the vector: $\boldsymbol{\eta} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}'$
- $\boldsymbol{J}$ is the Jacobian matrix. It's defined as:

$$\boldsymbol{J} = \begin{bmatrix} I_{xx} & 0 & -I_{xx}S_\theta \\ 0 & I_{yy}C_\phi^2 + I_{zz}S_\phi^2 & (I_{yy} - I_{zz})C_\phi S_\phi C_\theta \\ -I_{xx}S_\theta & (I_{yy} - I_{zz})C_\phi S_\phi C_\theta & I_{xx}S_\theta^2 + I_{yy}S_\phi^2 C_\theta^2 + I_{zz}C_\phi^2 C_\theta^2 \end{bmatrix} \tag{3.9}$$

- $\boldsymbol{C}$ is the Coriolis term, which contains the gyroscopic and centripetal terms:

$$\boldsymbol{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \tag{3.10}$$

where:

$$\begin{aligned}
C_{11} &= 0 \\
C_{12} &= (I_{yy} - I_{zz})(\dot{\theta}C_\phi S_\phi + \dot{\psi}S_\phi^2 C_\theta) + (I_{zz} - I_{yy})\dot{\psi}C_\phi^2 C_\theta - I_{xx}\dot{\psi}C_\theta \\
C_{13} &= (I_{zz} - I_{yy})\dot{\psi}C_\phi S_\phi C_\theta^2 \\
C_{21} &= (I_{zz} - I_{yy})(\dot{\theta}C_\phi S_\phi + \dot{\psi}S_\phi C_\theta) + (I_{yy} - I_{zz})\dot{\psi}C_\phi^2 C_\theta + I_{xx}\dot{\psi}C_\theta \\
C_{22} &= (I_{zz} - I_{yy})\dot{\phi}C_\phi S_\phi \\
C_{23} &= -I_{xx}\dot{\psi}S_\theta C_\theta + I_{yy}\dot{\psi}S_\phi^2 S_\theta C_\theta + I_{zz}\dot{\psi}C_\phi^2 S_\theta C_\theta \\
C_{31} &= (I_{yy} - I_{zz})\dot{\psi}C_\theta^2 S_\phi C_\phi - I_{xx}\dot{\theta}C_\theta \\
C_{32} &= (I_{zz} - I_{yy})(\dot{\theta}C_\phi S_\phi S_\theta + \dot{\phi}S_\phi^2 C_\theta) + (I_{yy} - I_{zz})\dot{\phi}C_\phi^2 C_\theta \\
&\quad + I_{xx}\dot{\psi}C_\phi^2 S_\theta C_\theta - I_{yy}\dot{\psi}S_\phi^2 S_\theta C_\theta - I_{zz}\dot{\psi}C_\phi^2 S_\theta C_\theta \\
C_{33} &= (I_{yy} - I_{zz})\dot{\phi}C_\phi S_\phi C_\theta^2 - I_{yy}\dot{\theta}S_\phi^2 C_\theta S_\theta - I_{zz}\dot{\theta}C_\phi^2 C_\theta S_\theta + I_{xx}\dot{\theta}C_\theta S_\theta
\end{aligned} \tag{3.11}$$

3.2 Euler-Lagrange equations for linear motion

The Euler-Lagrange equations for linear motion defined in [4] were not used since they are related to linear velocities, which in this study are equal to zero, being the quadcopter hinged to a static support.

3.3 State–Space representation

By exploiting 3.8, it is now possible to obtain the state space representation of the system. The first step is to rewrite that equation in function of $\boldsymbol{\eta}$. Then, by setting

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\eta} \\ \dot{\boldsymbol{\eta}} \end{bmatrix} \tag{3.12}$$

the following system is obtained:

$$\begin{cases} \dot{\boldsymbol{x}}_{\boldsymbol{1}} = \boldsymbol{x_2} \\ \dot{\boldsymbol{x}}_{\boldsymbol{2}} = \boldsymbol{J}^{\boldsymbol{-1}}(\boldsymbol{x_1})[\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{x_1}, \boldsymbol{x_2})\boldsymbol{x_2}] \end{cases} \tag{3.13}$$

Notice that this is a nonlinear representation. The linearized one will be automatically computed later via simulations on Simulink since it requires too much effort to be obtained manually.

3.3.1 2-dof State-Space representation

In case of the 2-dof representation, the quadrotor was theoretically considered fixed with respect to the yaw movement. It is important to underline the word "theoretically" since practically the yaw could not be fixed, but only constrained in its movement. The states thus become:

$$\boldsymbol{x} = \begin{bmatrix} \phi & \theta & \dot{\phi} & \dot{\theta} \end{bmatrix}' \tag{3.14}$$

while the vector of involved torques is:

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \end{bmatrix} \tag{3.15}$$

and the inertia matrix is ($I_{xx} = I_{yy}$):

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & 0 \\ 0 & I_{yy} \end{bmatrix} \tag{3.16}$$

In the angular Euler-Lagrange equation 3.8 the Jacobian matrix J and the Coriolis term C become:

$$\boldsymbol{J} = \begin{bmatrix} I_{xx} & 0 \\ 0 & I_{yy}C_\phi^2 \end{bmatrix} \quad \boldsymbol{C} = \begin{bmatrix} 0 & I_{yy}(\dot{\theta}C_\phi S_\phi) \\ I_{yy}(\dot{\theta}C_\phi S_\phi) & 0 \end{bmatrix} \tag{3.17}$$

The equations 3.13 of the state-space representation are still valid.

3.3.2 1-dof State-Space representation

In case of just 1-dof, the quadrotor is fixed with respect to both pitch and yaw movements. The state vector now contains just 2 states:

$$\boldsymbol{x} = \begin{bmatrix} \phi & \dot{\phi} \end{bmatrix}' \tag{3.18}$$

And there is just one single torque involved:

$$\boldsymbol{\tau} = \tau_\phi \tag{3.19}$$

Similarly, the inertia matrix becomes:

$$\boldsymbol{I} = I_{xx} \tag{3.20}$$

Finally, in equation 3.8 the Jacobian matrix $\boldsymbol{J}$ and the Coriolis term $\boldsymbol{C}$ become:

$$\boldsymbol{J} = I_{xx} \quad \boldsymbol{C} = 0 \tag{3.21}$$

Once again, the state-space representation equations are still 3.13.

4 Sensors

4.1 Introduction

The CUBEPILOT Cube Orange Flight Controller mounted on the quadcopter is equipped with three integrated sensors:

- Accelerometer: (3) ICM20948, ICM20649, ICM20602
- Gyroscope: (3) ICM20948, ICM20649, ICM20602
- Compass: (1) ICM20948

4.1.1 Sensor Interface

The sensors were available through the "UAV Toolbox Support Package for PX4 Autopilots" Simulink Package.

4.1.2 Sensors Calibration

Due to the mounting position of the integrated IMU, its x-axis direction coincided with the bisector of quadcopter's legs, in opposition with the model in 3.1, which has the x and y axes directed along the drone legs. In addition, the available sensors were also characterized by different measuring reference frames. For these reasons, a huge effort was put into converting sensors' measurements to the reference frame of the dynamical model here studied.

4.1.3 Accelerometer Calibration

Accelerometer calibration has been particularly challenging since the Simulink Package stated a wrong reference frame for the measurements in its description. While it was declared that the measuring would be done according to a NED reference frame (North-East-Down), it was noticed that, by keeping the drone horizontally, the sensor was measuring a vertical acceleration of $g = -9.81[m/s^2]$. This evident mistake was the trigger that led to questioning all sensors references. The right reference was found by manually inclining the quadcopter around drone bisectors. The real reference resulted to be a left-handed triad, with positive Z pointing upwards, as shown in Figure 4.1. To bring the measurements to a right-handed triad, a minus sign was put in front of the x-axis measurements. This step was necessary to perform the matrix rotation that would convert sensors' measurements to the body reference frame, as described in subsection 8.2.

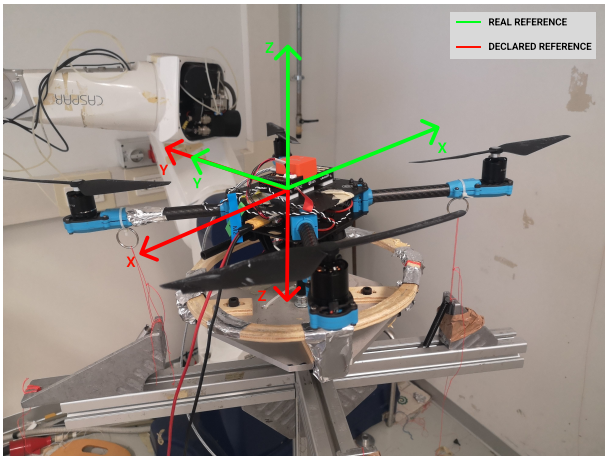

Figure 4.1: Comparison of Accelerometer Reference Frames

4.1.4 Gyroscope Calibration

For what concerns the gyroscope, no further indications were given about the correct reference to consider. Indeed, being the gyroscope mounted on the same IMU as the accelerometer, it could be erroneously thought that they were sharing the same reference. As for the accelerometer, by manually rotating the drone around its two bisectors, it has been proved that the gyro reference is indeed the one in Figure 4.2.

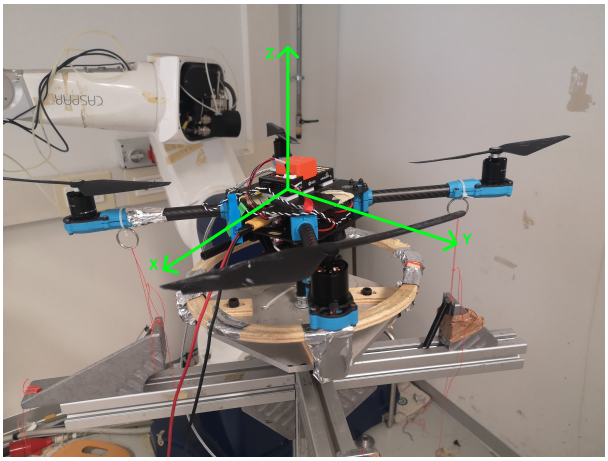

Figure 4.2: Gyroscope Reference Frame

4.1.5 Internal State Estimator

The last available sensor block was a built-in state estimator, which gave as output roll, pitch and yaw angles, with their corresponding rates. With the aim of using it as benchmark for the state estimator (Extended Kalman Filter) that will be subsequently derived, a reference frame identification was needed also in this case. As for the gyro, by rotating the quadcopter around its geometrical axes, it was found the correct measuring tern to be the same as the accelerometer (Figure 4.1).

4.1.6 Noise Analysis

Noise analysis on acceleration measurements was conducted with the goal of designing controllers robust with respect to measuring uncertainties. This analysis was not needed for the gyroscope and the magneto-meter since their noise level is two orders of magnitude lower. To perform this task, an acquisition of the drone standing still was analyzed with a Fast Fourier Transform (Figure 4.3). The analysis of the spectrum amplitude at high frequency evidenced that the sensor noise is of the order of 10^{-3}. The same FFT analysis was then conducted on more acquisitions, in which the drone was performing different movements at a different speed. This was done with the aim of exciting even the nonlinear part of sensors dynamics. The obtained results were still coherent with the one displayed in Figure 4.3.

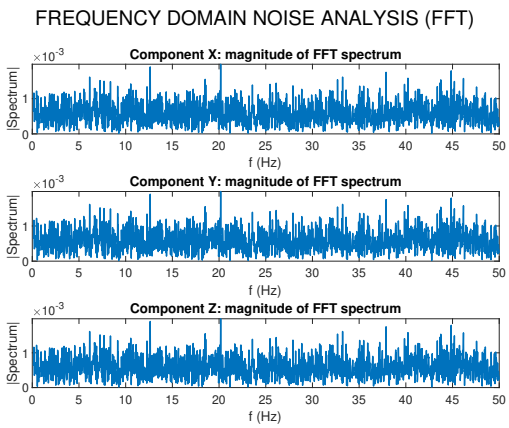

Figure 4.3: IMU noise analysis: FFT of the acceleration

4.1.7 Bias & Variance Estimation

The final step of sensor configuration was to estimate bias and variance of the IMU angle velocity measurements around the quadcopter zero position (the origin of the mechanical model). This step is crucial to correctly tune the Extended Kalman Filter weight matrix. The experimental activity consisted in acquiring data from the accelerometer and the gyroscope while the quadcopter was constrained by four ropes connected to the four ends. A spirit level was used to determine the correct zero position of the drone. Results are displayed in Table 4.1.

	ω^b_{mean}	a^b_{var}	ω^b_{var}	B^b_{var}	RPY_{var}
X-axis	-0.0079201	0.0003835	2.6581e-06	8.321e-05	4.1222e-06
Y-axis	-0.0045188	0.00040196	2.6466e-06	7.4496e-05	3.9054e-06
Z-axis	0.0092139	0.00029165	2.8585e-06	8.1874e-05	0.00038687

Table 4.1: IMU - Bias and Variance estimation

5 Parameters Estimation

5.1 Mechanical Model

The mechanical model of the quadcopter (3.8) includes just the Inertia Tensor (3.9) as physical parameter. In order to design and implement all the control logics, a system identification has been performed. The aim was to estimate mainly two quantities:

- Inertia tensor
- PWM-motor thrust relationship

5.2 Inertia Tensor Estimation

5.2.1 Introduction

The state of the art of inertia estimation requires to constrain the drone to the ceiling and oscillate it in a pendulum-like way. Since the setup did not allow this kind of approach, it was decided to pursuit different techniques:

- Mass-free spring oscillation
- Geometrical computation
- CAD analysis

Simplified System Free Oscillation

This first approach consisted in constraining the x-axis of the drone with two springs of known stiffness coefficient and exciting the roll dynamics. The constrained quadcopter was modelled as a beam hinged in the center with springs at the two extremities (Figure 5.1).

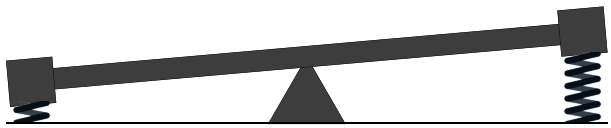

Figure 5.1: Mass-spring model used for the system identification

By exploiting this simplification, the 1-dof model of the drone can be formalized as in 3.18 - 3.21. The goal was to measure, via accelerometer acquisitions, the frequency of the free oscillation (Figure 5.2) and compute the inertia of the model by inverting the following formula:

$$I_{xx} = \frac{K}{w_n^2} \tag{5.1}$$

where K is the stiffness of the two springs.
From this approach, the roll inertia (I_{xx}) resulted 0.015 [kgm^2]. Since the quadcopter is symmetric with respect to the XZ and YZ planes, pitch Inertia (I_{yy}) can be assumed equal to roll Inertia.
To prove this assumption, the same experiment was performed by constraining the y-axis and exciting the pitch dynamics. The same approach could not be used to estimate the yaw Inertia coefficients due to the available setup. Being the drone constrained in its geometric center, in fact, it was impossible to excite its yaw dynamics.

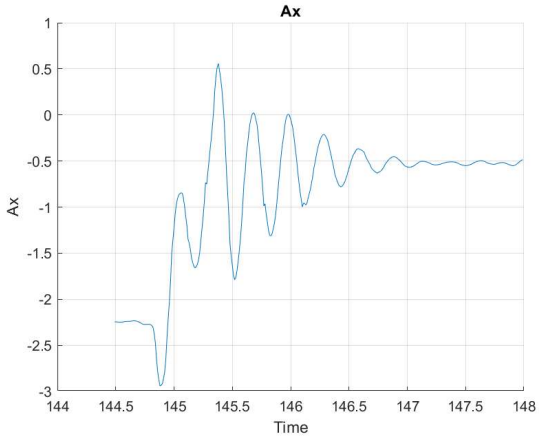

Figure 5.2: Acquisition: 1dof System Free Oscillation

Geometrical Computation

This second approach is completely different with respect to the former one, since it aims at computing the inertia by exploiting the mass and geometrical characteristics of the model. This step is to be interpreted as a validation of the first estimation technique. The main hypothesis of this experiment was to reduce the model to a cave cylinder with two masses at the ends, which represent the motors and the propellers. This implies that the inertia of the two axes are supposed as uncorrelated, which is reasonable due to the cross-like shape of the quadcopter projection on the XY plane. The inertia was then computed by considering an axis of rotation passing through the geometrical center and perpendicular to the cylinder. The inertia of such a system can be computed as:

$$I_{xx} = \frac{1}{12} \cdot 2m_{axis}L^2 + 2m_{tot}L^2 \tag{5.2}$$

The geometrical and mass data are taken from the X500 V2 Kits [1]. The final result was $I_{xx} = 0.0148$ [kgm^2]. The obtained value is coherent with the one obtained at the previous step.

CAD Inertia Computation

This additional validation step has been added during the final part of the laboratory experience due to some difficulties encountered while implementing the controllers. In particular, the one degree of freedom control was delivering poor performances and since roll inertia is the only parameter in the mono-axial model, a further analysis was undertaken to determine its real value. This approach was possible thanks to the 3D CAD project of the quadcopter available on the website [1]. The only step to be done was to choose the right material for the main components. Results are shown in Table 5.1 and are quite close to the geometrical computation performed at the beginning of the project. Despite this additional validation phase, this step still did not solve the problem of poor performances of the mono-axial controller.

Inertia Tensor	Value
I_{xx}	0.02194 $[kg \cdot m^2]$
I_{yy}	0.02192 $[kg \cdot m^2]$
I_{zz}	0.03407 $[kg \cdot m^2]$

Table 5.1: Inertia Tensor

Final Considerations about Inertia

Despite having validated the inertia tensor estimation results, the experimental experience of the mono-axial control has highlighted that the whole tensor was underestimated with respect to the real value. This was evidenced by the fact that the controller was not allocating enough torque to perform the rollover of the drone when starting from an inclined position. The easiest but, at the same time, most probable explanation is that none of the three estimation techniques were considering friction. In particular, the main source of friction in the model was the joint connecting the drone to the support, which has been also substituted during the six-month period of the laboratory activity. By constraining one axes of the quadcopter, it is likely that the spring/ropes were pushing one side of the joint, enhancing the friction action. This explanation is coherent with all the estimation experiences performed. It is in fact clear that in geometrical computation, knowledge about friction cannot be included. Even in the free-oscillation method, the model does not consider any damping factor. This difficulty in inertia estimation has been overcome by performing a closed loop identification.

5.3 Thrust-PWM Relationship

5.3.1 Introduction

The four motors of the quadcopter were controlled with the PWM techniques, as imposed by the built-in Simulink package. The mapping of PWM values to their correspondent motor's thrust force was then needed to interface the model with the real system. To accomplish this task, two experimental activities were conducted:

- Force measurements using the dynamometer
- Force measurements using the balance

5.3.2 Dynamometer

The idea of this experiment was to estimate the Thrust-PWM relationship by measuring the force applied by a single motor at different PWM values. To accomplish the task, three of the four motors were connected to the quadcopter support, while the remaining one was left free to move and connected to the end of the dynamometer. The experiment consisted in manually moving the drone to its equilibrium position and reading the force value displayed by the dynamometer. After observing the value of force needed to keep the drone in the origin, the quadrotor was switched on and the PWM of the unconstrained motor was raised until the zero position was reached. To maximize the precision of the measure, a spirit level was constrained to the quadcopter, facilitating the recognition of the equilibrium position. Since this experiment had to be repeated many times, in order to find a truthful curve, a special tool was designed and printed to make the procedure faster (Figure 5.3).

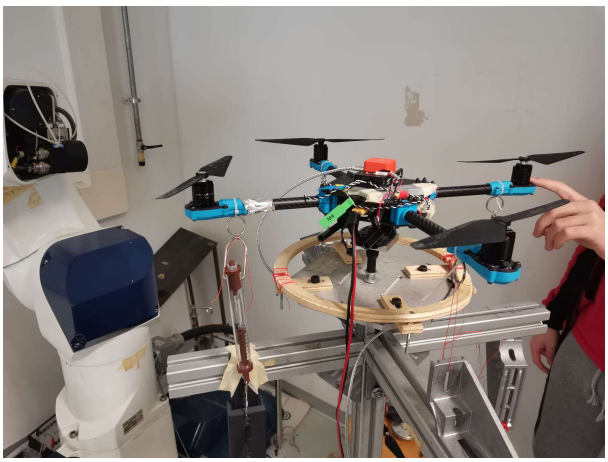

Figure 5.3: Dynamometer setup

This 3D printed support was in fact fixed to the drone basement and by exploiting the holes in it, it was possible to increase or decrease the force without having to change the

setup, saving up a lot of time. The main shortcoming of this setup was the number of holes in the support structure, which was limited, thus restricting the number of possible measurements. Another error source is the parallax error that may have been introduced when reading the measurements on the analog scale. The scaling coefficient between Thrust and PWM is 0.0083 (Figure 5.4).

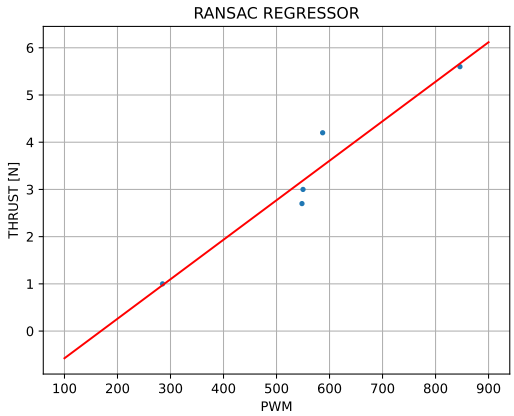

Figure 5.4: Dynamometer Regression

5.3.3 Balance

Setup

Towards the end of the laboratory experience, it was noticed that controller performances were poor in certain PWM regions, approximately from 500 to 700. By manually controlling the motors, it was observed that in this region the Thrust-PWM relationship appeared to be nonlinear, since visually the motors would increase their rotation speed faster. Despite being an empirical observation, this led to think that the quantity of points measured in the previous curve fitting experiment was not enough to enlighten a possible nonlinear relationship. For this reason, a second experiment was designed, this time using a balance as instrumentation. The experiment consisted in putting a one-kilogram mass on the balance, and constraining it with a 3D printed support to one of the motors; the opposite motor on the same axis was then switched on, in order to push the mass against the balance. The weight difference was then registered and multiplied by the gravity acceleration constant to compute the pushing force. The experiment was conducted for PWM values spanning from 0 to 600, with a resolution of 10. The limitation on high tested values was due to the fact that after a certain PWM, around 500, the air produced by the rotating motor was flowing onto the balance, adding uncertainty to the measure. Despite trying to deflect the air via a cardboard positioned under the drone, the measure for high PWM still resulted uncertain. The experiment was conducted for all the four motors to consider in the model the possible differences between them.

Curve Fitting

The obtained measures were fitted with the aim of finding an analytic expression for the thrust-PWM curve. Two fitting were conducted:

- Linear fitting (Figure 5.5): $f = K \cdot PWM$
- Polynomial fitting (Figure 5.6): $f = K_0 \cdot PWM + K_1 \cdot PWM^2$

Linear fitting was used mainly to make a comparison with the coefficient found with the dynamometer setup. As expected, conducting an experiment with a larger number of measurements, the resulting thrust-PWM coefficient was different. This can be attributed to the fact that, with this new experiment, it was possible to better study the intermediate PWM zone, which in the previous experiment was represented by just two points. The cloud of points was then fitted with a more suitable 2nd order polynomial. Results are shown in the tables 5.2 - 5.3 below. It can be noticed that the motors have slightly different coefficients; this has to be attributed to slightly different measuring conditions dictated by the setup itself. The balance, for instance, might have been fixed in a another position or inclination while repeating the experience for the four motors.

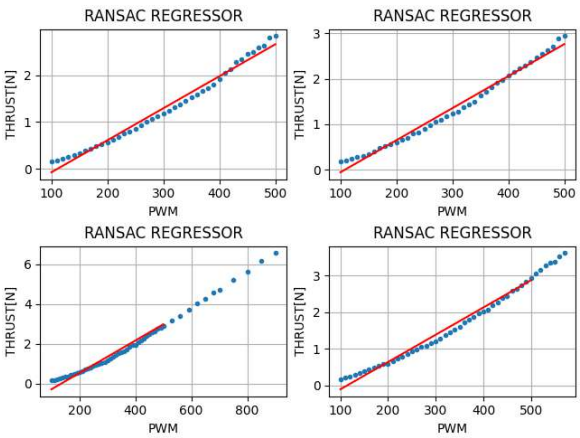

Figure 5.5: Linear Regression

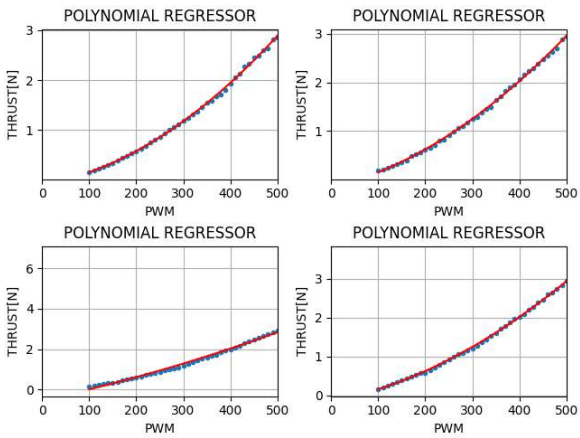

Figure 5.6: Polynomial regression

Motor	K
$Motor_1$	0.006843244
$Motor_2$	0.007038590
$Motor_3$	0.008149636
$Motor_4$	0.007466973

Table 5.2: Linear Regression

Motor	K_0	K_1
$Motor_1$	0.001665161	$8.63014 \cdot 10^{-6}$
$Motor_2$	0.002426698	$7.68649 \cdot 10^{-6}$
$Motor_3$	0.004978554	$3.47348 \cdot 10^{-6}$
$Motor_4$	0.002444508	$7.49622 \cdot 10^{-6}$

Table 5.3: Polynomial Regression

6 Model Implementation

Using the equations and the expressions found so far, it was possible to implement the open-loop system's model in Simulink, where block G contains the scheme on the right of Figure 6.1:

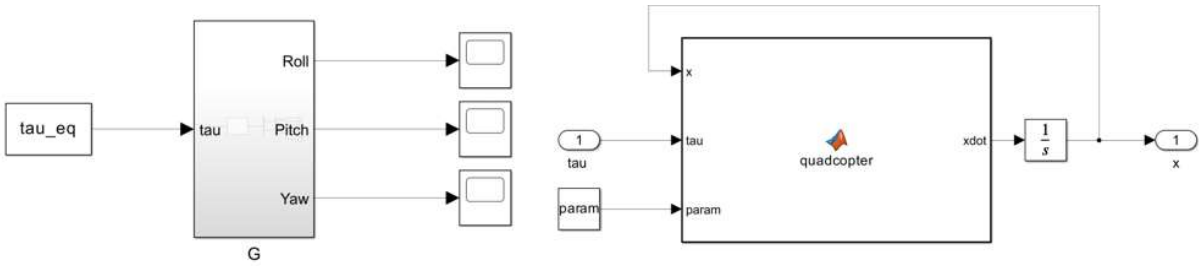

Figure 6.1: 3-dof open loop model and Simulink quadcopter model

while the block quadcopter contains the equation of motion 3.13.
Then this scheme was easily re-adapted to the 1-dof and 2-dof cases.

6.1 Model linearization

In order to use the implemented model in some of the following schemes, its linearized version was needed. Since the model is strongly nonlinear, the MATLAB function "linearize" was used to automatically compute the linearization by exploiting the open-loop system's model around the zero equilibrium.
In this way, matrices A, B, C, D of the system could be written for the 1-dof, 2-dof and 3-dof cases. The following system is obtained:

$$\begin{cases} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{cases} \tag{6.1}$$

7 Control in Simulation

In this chapter, all the implemented control schemes are reported for the 3-dof case. The adaptation to the 2-dof and 1-dof cases is trivial in most of the cases (only changing vector' dimensions and recomputing the control gains accordingly is needed) and thus it is reported only when not trivial.

Moreover, the control schemes here contained are referred to the first implementation phase. In a second moment, further details were considered, like the presence of the ZOH and of external disturbances. The anti-windup is also added when needed.

7.1 Linear Control – 3-dof Pole Placement

The first implemented control strategy in simulation is the pole placement with intergrators.
The linearized model used is 6.1, found in previous paragraphs. After checking that the pair (A, B) was reachable, the enlarged system was designed so that the enlarged state became:

$$x_e = \begin{bmatrix} x & v \end{bmatrix}' \tag{7.1}$$

where v is the integral of the error vector e.
Then, after checking that the enlarged pair (A_e, B_e) was reachable too, the gain matrix K was designed using the MATLAB function "place". The new poles were chosen so that they were all negative, the nearest possible to the original ones.
K was then split into K_x and K_v and the complete control scheme was implemented in Simulink, as shown in Figure 7.1, where block G is the model explained in Chapter 6.

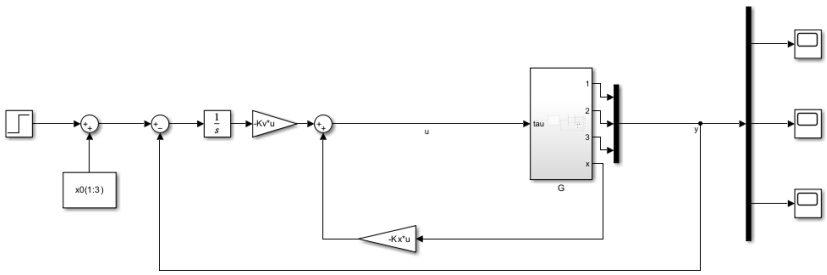

Figure 7.1: 3-dof Pole Placement - Control Scheme

7.2 Linear Control - 3-dof H2

In this subsection, a more advanced control startegy is implemented and analyzed: H2 control.

Given the state space representation in Equation 6.1, a static state-feedback control law was used:

$$u = Kx + \gamma \tag{7.2}$$

where $K \in \mathbb{R}^{m \times n}$ is the control gain. K is used for the stabilization while γ is used to design an external loop for the performances by introducing the regulator R [3].

The scheme in Figure 7.2 represents the general idea of this control.

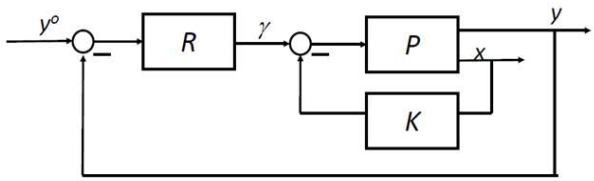

Figure 7.2: H2 - Control Scheme Structure

To find K, a MATLAB function was developed to solve an optimization problem exploiting LMIs [2]. The function takes as inputs the matrices of the state-space representation and gives as outputs the gain K and the feasibility of the problem. The LMIs used in the function (subsection 7.2.3) were the ones related to H2 control and to the minimization of the control effort.
The H2 control was realized as an LQ control problem.

The transfer function between γ and y was obtained by combining 6.1 with 7.2 and it consists in a matrix of 3×3 transfer functions. In the simulink files it is placed inside the "stable_sys" block while in the computations is named G_{stable}.

As shown in Figure 7.2, γ is the control action coming from the controller located in the outer loop. In this case, γ was realized through three PID controllers, respectively for roll, pitch and yaw angles control.

To allow for the three PIDs to be independently tuned, a decoupler was realized [5]
If G_{stable} is defined as:

$$G_{stable}(s) = \begin{bmatrix} G_{11}(s) & G_{12}(s) & G_{13}(s) \\ G_{21}(s) & G_{22}(s) & G_{23}(s) \\ G_{31}(s) & G_{32}(s) & G_{33}(s) \end{bmatrix} \tag{7.3}$$

then the desired decoupled system is:

$$G_{desired}(s) = \begin{bmatrix} G_{11}(s) & 0 & 0 \\ 0 & G_{22}(s) & 0 \\ 0 & 0 & G_{33}(s) \end{bmatrix} \tag{7.4}$$

The transfer matrix of the decoupler, $G_{decoupler}(s)$, can be found as:

$$G_{decoupler}(s) = inv(G_{stable}(s)) \cdot G_{desired}(s) \tag{7.5}$$

A PID regulator was then implemented for each angle.
The resulting Simulink control scheme is shown in Figure 7.3.

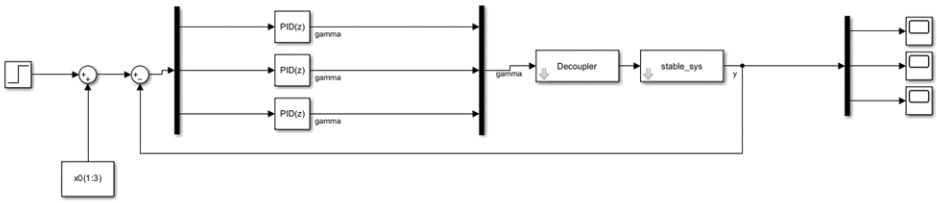

Figure 7.3: 3-dof H2 - Control Scheme

7.2.1 2-dof H2

In the 2-dof case, the control structure is analogous to the 3-dof one. Only the differences will be treated here.

$G_{stable}(s)$ is now a 2x2 matrix transfer function, that can be defined as:

$$G_{stable,2dof}(s) = \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} \tag{7.6}$$

The decoupler can be computed as:

$$G_{decoupler,2dof}(s) = \begin{bmatrix} 1 & -\frac{G_{12}(s)}{G_{11}(s)} \\ -\frac{G_{21}(s)}{G_{22}(s)} & 1 \end{bmatrix} \tag{7.7}$$

This can be verified by the fact that the product of 7.6 by 7.7 is the decoupled system, on which two PIDs can be independently designed, as in Figure 7.4.

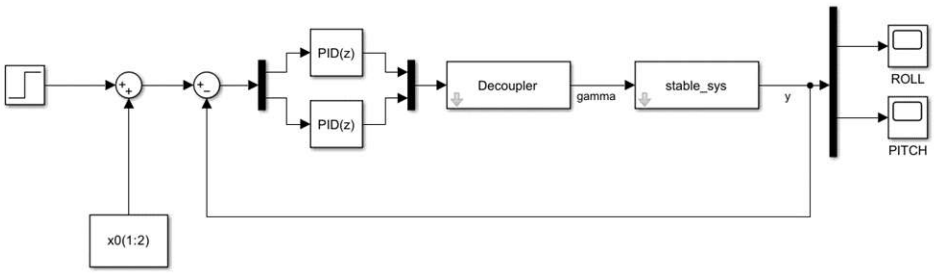

Figure 7.4: 2-dof H2 - Control Scheme

7.2.2 1-dof H2

This is the simplest implementation, where $G_{stable,1dof}(s)$ is a single transfer function and thus no decoupler is needed and so the regulator is directly designed on the system (Figure 7.5).

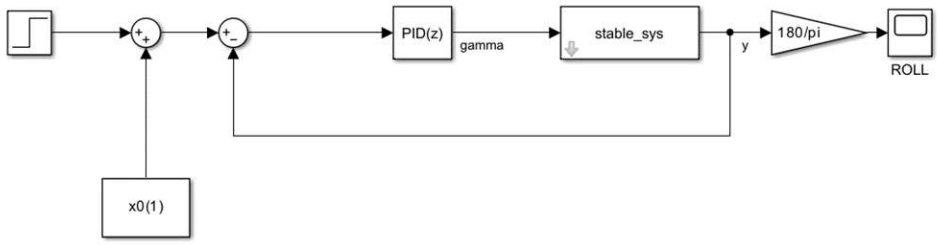

Figure 7.5: 1-dof H2 - Control Scheme

7.2.3 H2 Control - LMIs

In this subsection, the LMIs used in the control problem are briefly described [2].

From 6.1 and 7.2, it is possible to obtain the closed-loop matrix $(A+BK)$. From the definition of Hurwitz stability of $(A+BK)$ with matrix $P = P' > 0$, the LMI for the feasibility of the control problem is obtained:

$$YA' + AY + L'B' + BL < 0 \tag{7.8}$$

$$Y > 0 \tag{7.9}$$

where $Y = P^{-1}$ and $L = KP^{-1}$.

By writing the state-space as:

$$\begin{cases} \dot{x} &= Ax + Bu + B_w w \\ z &= Cx + D_u u \end{cases} \tag{7.10}$$

where w represents a noise or a disturbance, and by defining:

$$C = \begin{bmatrix} \sqrt{Q} \\ 0 \end{bmatrix} \quad D_u = \begin{bmatrix} 0 \\ \sqrt{R} \end{bmatrix} \tag{7.11}$$

LQ-like optimal performances can be imposed.
The LMIs used for H2 norm minimization are:

$$YA' + AY + BL + L'B' + B_w B_w' \leq 0 \tag{7.12}$$

$$\begin{bmatrix} S & CY + D_u L \\ L'D_u' + YC' & Y \end{bmatrix} \geq 0 \tag{7.13}$$

The ones used for the minimization of the control effort are:

$$\begin{bmatrix} \kappa_L I & L' \\ L & I \end{bmatrix} \geq 0 \tag{7.14}$$

$$\begin{bmatrix} \kappa_Y I & I \\ I & Y \end{bmatrix} \geq 0 \tag{7.15}$$

$$\kappa_L \geq 0 \tag{7.16}$$

$$\kappa_Y \geq 0 \tag{7.17}$$

with

$$f = trace(S) + 0.02\kappa_L + 0.01\kappa_Y \tag{7.18}$$

as objective function to minimize.

7.3 Nonlinear Control - 3-dof Inverse Dynamics

Another type of control strategy is the inverse dynamic control, which is a special case of feedback linearization.

Assuming to have a perfect knowledge of the dynamic model, the following control law was taken into consideration:

$$\boldsymbol{\tau} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{v} + \boldsymbol{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})\dot{\boldsymbol{\eta}} \tag{7.19}$$

whose application lead to:

$$\ddot{\boldsymbol{\eta}} = \boldsymbol{v} \tag{7.20}$$

The system dynamics has been made linear, completely decoupled and composed of n double integrators, where n is the order of the system. Thus, the following system matrices were obtained:

$$\begin{aligned} A &= \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} \quad B = \begin{bmatrix} 0_{3\times1} \\ 1_{3\times1} \end{bmatrix} \\ C &= \begin{bmatrix} I_{3\times3} & 0_{3\times3} \end{bmatrix} \quad D = \begin{bmatrix} 0_{3\times1} \end{bmatrix} \end{aligned} \tag{7.21}$$

Two different types of controllers were built on this linear system: a pole placement and a PID.

7.3.1 Inverse Dynamics - Pole Placement

After checking that the pair (A, B) was reachable, the gain matrix K was designed using the MATLAB function “place”. The new poles were chosen so that they were all negatives and the nearest possible to the original ones.

After that, the Simulink scheme in Figure 7.6 was implemented, where the "System" block contains the open-loop model of the system, while the block "Controller" is

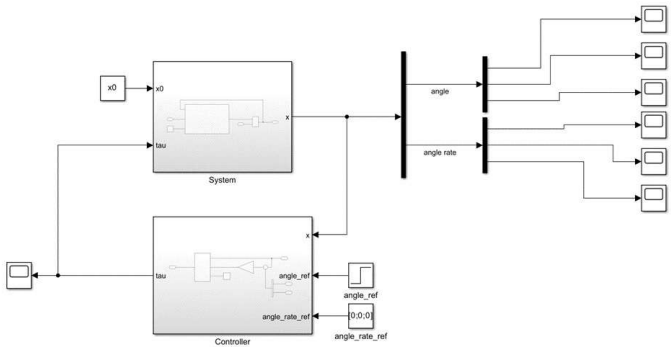

Figure 7.6: Inverse Dynamics Control Scheme

composed as in Figure 7.7 where the MATLAB function contains Equation 7.19.

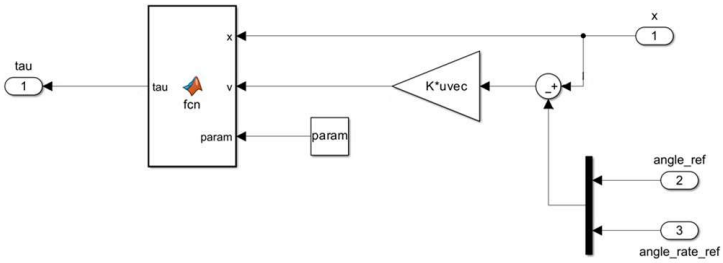

Figure 7.7: 3-dof Inverse Dynamics with Pole Placement - Controller

7.3.2 Inverse Dynamics - PID

For what concerns the PID controller, the scheme is the same as Figure 7.6, with the only difference that now there is a PID block (Figure 7.8), tuned to obtain the desired performances.
In addition, only the angle reference is needed here.

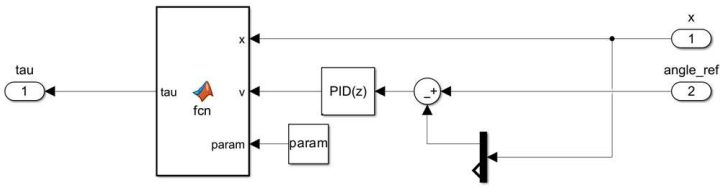

Figure 7.8: 3-dof Inverse Dynamics with PID - Controller

7.4 Nonlinear Control - 1-dof VSC

A second type of nonlinear control strategy that was taken into consideration is the Variable Structure Controller. This method, which was chosen because of its strong robustness, was implemented for the 1-dof case only [6].

The design procedure consisted in two steps:

1. Determining the switching function such that the system constrained to the sliding surface $s(x) = 0$ converged to a (pseudo-)equilibrium with $y = y^\circ$.
2. Determining a control law $u = k(x; y^\circ)$ such that all the state trajectories starting outside the sliding surface crossed the surface in finite time.

To do so, the following characteristics and problems were considered:

- High frequency input switching issue;
- Robustness with respect to load disturbance;
- Robustness with respect to parameter uncertainty.

To realize the first step of the procedure, the system constrained to the sliding surface was enlarged to solve the high frequency switching issue, then the eigenvalues of the enlarged system were chosen.
This allowed to find all the necessary parameters and to move onto the second step.
The complete control scheme is reported in Figure 7.9.

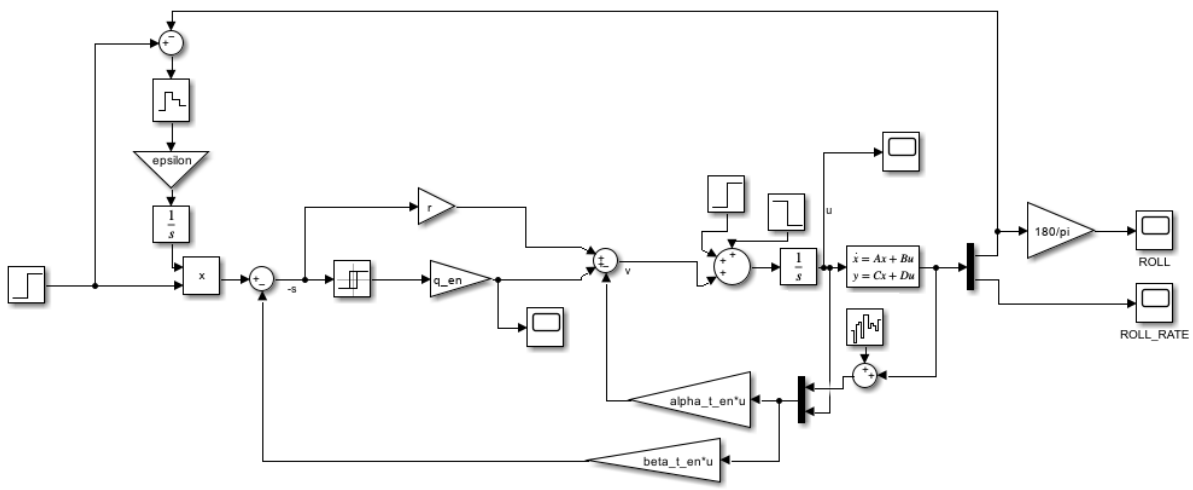

Figure 7.9: 1-dof VSC Control Scheme

8 Observer

8.1 Introduction

As mentioned in the previous chapters, the available sensors measure linear acceleration, angular velocity, and magnetic field in the so-called 'IMU-reference frame.' However, a controller needs to feedback the state of the system, so an observer must be included in the scheme to retrieve this information from the IMU. The design of such a block cannot rely on the dynamic model of the system (the quadcopter) due to model uncertainty; consequently, a kinematic-based observer is considered.

8.2 Sensor Model and Reference Frame

Before explaining the basics of the observer, it is fundamental to introduce all the reference frames involved in the computation. The "body-reference frame"(F_b) has been introduced with Figure 3.1, the "IMU-reference frame"(F_{imu}) is the one in which the measures are taken and is depicted in figure 3.1, while the "Inertia-reference frame"(F_i) is the one attached to the earth. The angle of interest are the ones that represent the orientation of F_b with respect to the F_i. So the F_{imu} frame is rotated in order to align its axes with the F_b frame. To do so, the vectors acceleration, velocity and magnetic field, measured by the sensor, are rotated by 45° around the z-axis. The corresponding matrix 8.1 is reported below.

$$R_{b\backslash imu} = \begin{bmatrix} cos(45°) & -sin(45°) & 0 \\ sin(45°) & cos(45°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8.1}$$

From now on it is enough to focus on the remaining two reference frames F_b and F_i. Recalling Equation 3.6, matrix R describes the relations between the coordinates of a generic vector $\boldsymbol{v}$ expressed in F_b and F_i frames respectively:

$$\boldsymbol{v}^b = \boldsymbol{R}'\boldsymbol{v}^i \tag{8.2}$$

where the superscripts indicate the coordinate system in which the vector is expressed. In the F_i frame, the principal vectors are the Earth's gravitational fields, modeled as

$$\boldsymbol{g^i} = [0\,, 0\,, -g]' \tag{8.3}$$

where $g = 9.81[m/s^2]$ denotes the strength of the gravitational field, and the Earth's magnetic field, expressed as

$$\boldsymbol{B^i} = [Bcos(\delta)\,, 0\,, Bsin(\delta)]' \tag{8.4}$$

where B denotes the strength of the field while the angle δ indicates the angle of inclination of the field, which depends on the location on the Earth.
On the other hand, the vectors measured by the IMU in the F_b frame are indicated as a^b, ω^b and B^b for the linear acceleration, angular velocity and magnetic field respectively. The only one notion that must be done regards the expression of the acceleration a^b: the sensor measures the total acceleration including the gravitational field. Hence, the signal from the IMU is modeled as follows:

$$\boldsymbol{a}^b = \boldsymbol{R}'(\boldsymbol{a}^i + \boldsymbol{g}^i) \tag{8.5}$$

This is a pivot observation in the observer design.

8.3 Extraction of g

The computation of the angles, presented in the next section 8.4, needs a measurement of ω^b, g^b and B^b, but only a^b is available from the sensor. To retrieve this information, the data from the linear acceleration must be correctly manipulated by means of three actions:

1. Low-Pass Filter
2. Sliding Median Filter
3. Sliding Average Filter

8.3.1 Low-Pass Filter

Firstly, a low-pass filter is applied to the signal to remove the high-frequency noise present (this action is done also for the angular velocity and magnetic field measurements). The filter is a 1-order discrete transfer function with unitary gain and a pole located at the same frequency f_{filter} [Hz].

$$filter(s) = \frac{1}{1 + \frac{1}{f_{filter}} s} \tag{8.6}$$

The selection of the pole of this filter is a crucial step for the correct working of the other stages in the Observer, hence a deep noise analysis is carried out.

Motor Noise Analysis

During the quadcopter working, the rotation of the four motors generates vibrations, with a corresponding frequency that depends on the motors' RPM, which, in turn, is related to the PWM of each motor. To analyze this effect, a series of experiment are performed. The quadcopter is mounted with four springs to keep it still as much as possible and the motors are all set to a specific PWM value in each test. With this setup, seven data

acquisitions are performed with different values of PWM in each one, as reported in Table 8.1. The spectrum of linear acceleration (as well as angular velocity and magnetic field)

Test	PWM
1° Test	100
2° Test	300
3° Test	500
4° Test	600

Table 8.1: noise motor tests

is then computed, revealing that the motion of the motors introduces high-frequency disturbances, mainly located at around 8 [Hz], 25 [Hz], and 40 [Hz]. Figure 8.1 and 8.2 highlight this conclusion. This is enough to conclude that a cut-off frequency of 10 [Hz] for

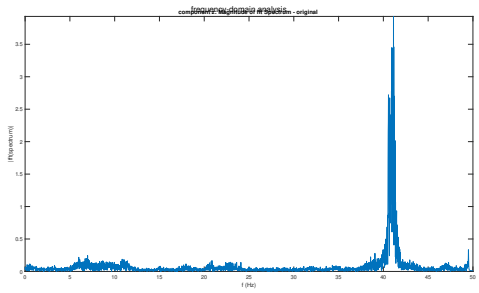

Figure 8.1: Motor Noise Analysis: 100 PWM

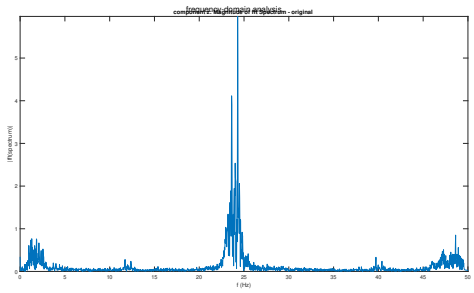

Figure 8.2: Motor Noise Analysis: 500 PWM

the filter is able to eliminate those disturbances. Additionally, to prove this hypothesis, the same signals are filtered in Simulink and then compared. Figure 8.3 and 8.4 show the result. The motor noise is drastically removed from the original signal.

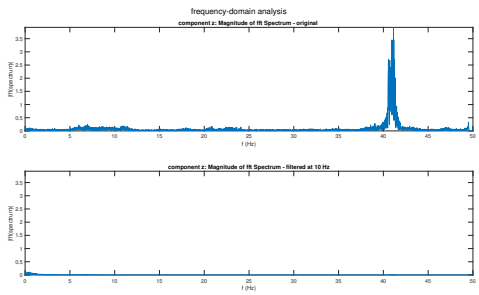

Figure 8.3: Motor Noise Analysis: 100 PWM - Filtered

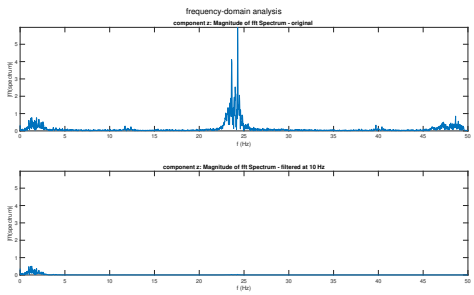

Figure 8.4: Motor Noise Analysis: 500 PWM - Filtered

8.3.2 Sliding Median Filter

The accelerometer measures the total acceleration including the gravitational field. To extract only the $\boldsymbol{g_i}$ component, a Sliding Median Filter is used.

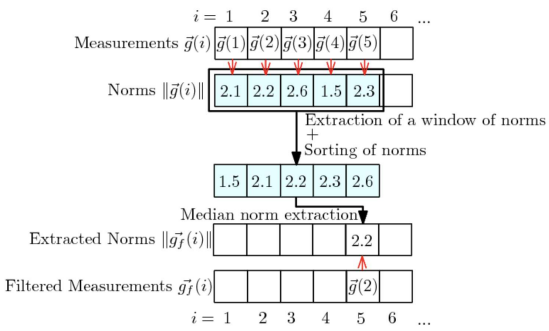

Figure 8.5: Sliding Median filter

The sliding median filter uses a sliding window of n norms. At the beginning, the window contains the first n norms of the first samples. Then, the norms of the window are sorted. Finally, the median value of the window is extracted, and the sample, whose norm is the median value, is output from the filter, as shown in 8.5.

This technique removes the most of the incorrect samples from the measurements. The output of this process is indicated as $\boldsymbol{a}^b_{smf}(k)$.

8.3.3 Sliding Average Filter

To increase the precision and remove residual vibrations in the measurement, a Sliding Average Filter is used. The sliding average filter uses a sliding window of n samples. At the beginning, the window contains the first samples. Then, the average value of the window is extracted and output from the filter. The expression of the filter is given by:

$$\boldsymbol{g}^b(t) = \frac{1}{n} \cdot \sum_{k=t-n+1}^{t} \boldsymbol{a}^b_{smf}(k) \tag{8.7}$$

8.4 Computation of the angles

The measurements of $\boldsymbol{g}^b$ (obtained from $\boldsymbol{a}^{\boldsymbol{b}}$) and $\boldsymbol{B}^{\boldsymbol{b}}$, combined with the knowledge of the corresponding vectors in the F_i frame, allow to compute the roll, pitch and yaw angles. Indeed, using Equation 8.2, 8.3, it is possible to compute the roll and pitch angles:

$$\begin{aligned} \tilde{\phi} &= \text{atan2}\left(\frac{g^b_y}{g^b_z}\right) \\ \tilde{\theta} &= \text{atan2}\left(\frac{-g^b_x}{g^b_y S_{\tilde{\phi}} + g^b_z C_{\tilde{\phi}}}\right) \end{aligned} \tag{8.8}$$

Once the Observer uses this information to estimate the roll and pitch angles (section 8.5), indicated as $\hat{\phi}$ and $\hat{\theta}$, Equations 8.2 and 8.4 are used to finally compute the yaw angle as:

$$\tilde{\psi} = \text{atan2}\left(\frac{B_z^b S_{\hat{\phi}} - B_y^b C_{\hat{\phi}}}{B_x^b C_{\hat{\theta}} + B_y^b S_{\hat{\phi}} + B_z^b C_{\hat{\phi}} C_{\hat{\theta}}}\right) \tag{8.9}$$

8.5 EKF's Algorithm

To estimate the three angles, two observers are used: the first one handles the estimation of roll and pitch, and the second one estimates the yaw angle. This division comes directly from the computation method of the angles (section 8.4), where it is shown that the yaw angle requires knowledge of the other two angles for its computation.

8.5.1 System Model - Roll and Pitch

As anticipated in the Introduction (section 8.1), the Observer relies on the kinematics equation between the angular velocity and the time derivative of the angles, here the expression:

$$\dot{\eta} = W(\eta)\omega^b \quad \Rightarrow \quad \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & S_\phi T_\theta & C_\phi T_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & \frac{S_\phi}{C_\theta} & \frac{C_\phi}{C_\theta} \end{bmatrix} \cdot \omega^b \tag{8.10}$$

From Equation 8.10 it is possible to notice that the first two rows depend only on roll and pitch angles and they can be used to design an Extended Kalman Filter (EKF) for this two angles.
Equation 8.11 shows the "nonlinear system model" on which the EKF is implemented:

$$S_{\phi,\theta} : \begin{cases} \dot{\boldsymbol{x}}_{\boldsymbol{\phi},\boldsymbol{\theta}}(t) = f(\boldsymbol{x}_{\boldsymbol{\phi},\boldsymbol{\theta}}(t), \boldsymbol{\omega}^b(t)) & +\boldsymbol{v}_{\boldsymbol{x}}(t) \\ \boldsymbol{y}(t) = g(\boldsymbol{x}_{\boldsymbol{\phi},\boldsymbol{\theta}}(t)) & +\boldsymbol{v}_{\boldsymbol{y}}(t) \end{cases} \tag{8.11}$$

where, for this specific system $S_{\phi,\theta}$:

- $\boldsymbol{x}_{\boldsymbol{\phi},\boldsymbol{\theta}} = \begin{bmatrix} \phi & \theta & b_x & b_y & b_z \end{bmatrix}'$ represents the state of the EKF, in which the first two variables are the angles to be estimated, while the other three represent the bias of the gyroscope. The estimation of the bias in important to reduce error in the angular velocity measurement.
- $\boldsymbol{y} = \begin{bmatrix} \tilde{\phi} & \tilde{\theta} \end{bmatrix}'$ represents the available measurement of $S_{\phi,\theta}$ system, which are the roll and pitch angles, computed with equation in section 8.4.
- $\boldsymbol{\omega}^b = \begin{bmatrix} \omega_x^b & \omega_y^b & \omega_z^b \end{bmatrix}'$ represents the angular velocity of the quadcopter in F_b frame, and in $S_{\phi,\theta}$ plays the role of input signal to it.

- $f(\boldsymbol{x_{\phi,\theta}}(t), \boldsymbol{\omega}^b(t))$ represents the vector field of $S_{\phi,\theta}$ shown in Equation 8.12. The first two rows are related to Equation 8.10, with additional correction of the gyroscope's bias, while the other three are fictitious and used to estimate the slowly varying bias of the gyroscope:

$$f(\boldsymbol{x_{\phi,\theta}}, \boldsymbol{\omega}^b) = \begin{bmatrix} 1 & S_\phi T_\theta & C_\phi T_\theta & 0 & 0 & 0 \\ 0 & C_\phi & -S_\phi & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \omega_x^b - b_x \\ \omega_y^b - b_y \\ \omega_z^b - b_z \\ b_x \\ b_y \\ b_z \end{bmatrix} \tag{8.12}$$

- $g(\boldsymbol{x_{\phi,\theta}}(t))$ represents the output equation of $S_{\phi,\theta}$ and it consists in the direct measurements of roll and pitch (Equation 8.8):

$$g(\boldsymbol{x_{\phi,\theta}}(t)) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \boldsymbol{x_{\phi,\theta}} \tag{8.13}$$

- $\boldsymbol{v_x}(t) \sim WN(0, \tilde{Q}_{\phi\theta})$ and $\boldsymbol{v_y}(t) \sim WN(0, \tilde{R}_{\phi\theta})$ represent the state and the output noise respectively, where $\tilde{Q}_{\phi\theta} \in \mathbb{R}^{5\times5}$ and $\tilde{R}_{\phi\theta} \in \mathbb{R}^{2\times2}$ are both diagonal matrices to be later tuned.

On this system model $S_{\phi,\theta}$, an EKF is implemented.

8.5.2 System Model - Yaw

The same approach is used to define the system model for the EKF on the yaw, but with the difference that, in this case, roll and pitch angles do not come from "measurements" (Equation 8.8), but directly from an estimate provided by the previous Observer.
The structure of the system model S_ψ is the same of Equation 8.11:

$$S_\psi : \begin{cases} \dot{\boldsymbol{x}}_{\boldsymbol{\psi}}(t) = f(\boldsymbol{x_\psi}(t), \boldsymbol{\omega}^b(t)) & +\boldsymbol{v_x}(t) \\ \boldsymbol{y}(t) = g(\boldsymbol{x_\psi}(t)) & +\boldsymbol{v_y}(t) \end{cases} \tag{8.14}$$

But, in this application, the fields are the following:

- $\boldsymbol{x_\psi} = \begin{bmatrix} \psi \end{bmatrix}$ is the state and contains only the yaw angle.
- $\boldsymbol{y} = \begin{bmatrix} \tilde{\psi} \end{bmatrix}$ represents the measured variable of system S_ψ.
- $\boldsymbol{\omega}^b = \begin{bmatrix} \omega_x^b & \omega_y^b & \omega_z^b \end{bmatrix}'$ represents the angular velocity of the quadcopter in F_b frame, and in S_ψ plays the role of input signal to it.

- $f(\boldsymbol{x_\psi}(t), \boldsymbol{\omega}^b(t))$ is the vector field of S_ψ and it is made by the third row of Equation 8.10 with the usual gyroscope's bias correction:

$$f(\boldsymbol{x_\psi}(t), \boldsymbol{\omega}^b(t) = \begin{bmatrix} 0 & \frac{S_{\hat{\phi}}}{C_{\hat{\theta}}} & \frac{C_{\hat{\phi}}}{C_{\hat{\theta}}} \end{bmatrix} \cdot \begin{bmatrix} \omega_x^b - b_x \\ \omega_y^b - b_y \\ \omega_z^b - b_z \end{bmatrix} \tag{8.15}$$

(observation: $\hat{\phi}$ and $\hat{\theta}$ are estimate coming from the previous EKF)

- $g(\boldsymbol{x_\psi}(t))$ represents the output equation of $S_{\phi,\theta}$, and it models the direct measurements of yaw by Equation 8.9):

$$g(\boldsymbol{x_\psi}(t)) = \begin{bmatrix} 1 \end{bmatrix} \cdot \boldsymbol{x_\psi} \tag{8.16}$$

- $\boldsymbol{v_x}(t) \sim WN(0, \tilde{Q}_\psi)$ and $\boldsymbol{v_y}(t) \sim WN(0, \tilde{R}_\psi)$ represent the state and the output noise respectively, where $\tilde{Q}_\psi \in \mathbb{R}$ and $\tilde{R}_\psi \in \mathbb{R}$ are tuning parameters.

On this system model S_ψ, an EKF is implemented.

8.6 Implementation

Once the two system models are defined, the implementation of both EKF is carried out. Due to the intrinsic structure of the whole system, which is composed of continuous systems ($S_{\phi,\theta}$ and S_ψ) with discrete measurements, a *Continuous Extended Kalman Filter with discrete measurements* is adopted. This kind of Observer is suitable when the dynamic model of the system is in continuous-time ($t \in \mathbb{R}$), while the estimate algorithm is implemented in discrete-time ($k \in \mathbb{Z}$) on a micro-controller with a polling of the output $\boldsymbol{y}$ variable fixed in time.
The algorithm is the same for both $S_{\phi,\theta}$ and S_ψ, hence here is reported for a general system with the same structure of the dynamic equation:

$$S : \begin{cases} \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{\omega}^b(t)) & +\boldsymbol{v_x}(t) \quad \boldsymbol{v_x} \sim WN(0, \tilde{Q}) \\ \boldsymbol{y}(t) = g(\boldsymbol{x}(t)) & +\boldsymbol{v_y}(t) \quad \boldsymbol{v_y} \sim WN(0, \tilde{R}) \end{cases} \tag{8.17}$$

The algorithm consists of two steps.

Predictor Step

1. Initial condition:

$$\hat{\boldsymbol{x}}(k-1), \;\; \boldsymbol{P}(k-1) \tag{8.18}$$

2. Predicted state and error covariance matrix computation:

$$\begin{aligned} \dot{\hat{\boldsymbol{x}}}(t) &= f(\hat{\boldsymbol{x}}(t), \boldsymbol{\omega}^b(t)) \\ \dot{\boldsymbol{P}}(t) &= \boldsymbol{A}(t)\boldsymbol{P}(t) + \boldsymbol{P}(t)\boldsymbol{A}(t) + \tilde{\boldsymbol{Q}} \qquad \boldsymbol{A}(t) = \left.\frac{\partial f(\boldsymbol{x}(t), \boldsymbol{\omega^b}(t))}{\partial \boldsymbol{x}(t)}\right|_{\hat{\boldsymbol{x}}(t), \boldsymbol{\omega^b}(t)} \end{aligned} \tag{8.19}$$

This step gives in output: $\hat{\boldsymbol{x}}_t$ and $\boldsymbol{P}_t$.

Update Step

1. Compute the Kalman Gain:

$$\boldsymbol{L}(k) = \boldsymbol{P}_t\boldsymbol{C}(k)'(\boldsymbol{C}(k)\boldsymbol{P}_t\boldsymbol{C}(k)' + \tilde{\boldsymbol{R}})^{-1} \qquad \boldsymbol{C}(k) = \left.\frac{\partial g(\boldsymbol{x}(t))}{\partial \boldsymbol{x}(t)}\right|_{\hat{\boldsymbol{x}}(t), \boldsymbol{\omega^b}(t)} \tag{8.20}$$

2. Compute the state estimate:

$$\hat{\boldsymbol{x}}(k) = \hat{\boldsymbol{x}}_t + \boldsymbol{L}(k)\left(\tilde{\boldsymbol{y}}(k) - g(\hat{\boldsymbol{x}}_t)\right) \tag{8.21}$$

3. Compute the covariance matrix:

$$\boldsymbol{P}(k) = \left(I + \boldsymbol{L}(k)\boldsymbol{C}(k)\right)\boldsymbol{P}_t \tag{8.22}$$

Prediction Step Starting from the state estimate $x(k-1)$, error covariance matrix $\boldsymbol{P}(k-1)$ and input $\boldsymbol{\omega^b}(k-1)$ of the previous time instant *k-1*, the state equation of system S and the one of the estimated error covariance matrix $\boldsymbol{P}$ are integrated over time (a Forward Euler Integration is performed with a sampling time $T_{predic} = 0.001[s]$). The result of this first part is a prediction of the state $\hat{\boldsymbol{x}}_t$ and a value of covariance matrix equal to $\boldsymbol{P}_t$.

Update Step With matrix $\boldsymbol{P_t}$, the Observer gain $\boldsymbol{L_k}$ is updated; then, given the prediction $\hat{\boldsymbol{x}}_{\boldsymbol{t}}$, an estimate of the state $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$ is computed making a weighted sum (the weight is the gain $\boldsymbol{L_k}$) of the state prediction with the one-step ahead prediction error. Finally, the covariance matrix $\boldsymbol{P}(k)$ is updated with the new gain $\boldsymbol{L}(k)$ and matrix $\boldsymbol{P}_t$.

8.7 Validation and Simulation

The challenging part of an EKF implementation is the tuning of the matrices $\tilde{R}$ and $\tilde{Q}$. The latter represents the variance of the measurement noise, whose values are computed during the sensor analysis (chapter 4.1.7). The tuning of the state noise matrix, however, requires an exhaustive trial and error process. Due to the high computational power required to start each simulation, along with the additional time needed to upload the

script onto the quadcopter's onboard micro-controller, the number of simulations per laboratory session is insufficient to complete this task.

Therefore, the solution involves recording tests where the quadcopter is moved with the motors off. During these tests, the IMU sensor signals and the state estimates provided by the quadcopter's Internal State Estimator are logged. In Simulink, the EKF is then implemented and fed with the same sensor signals, allowing its state estimate to be compared with that provided by the Internal State Estimator. This setup allows for extensive testing of the EKF and proper tuning of the matrices involved in the estimation. The results are shown in Figures 8.6.

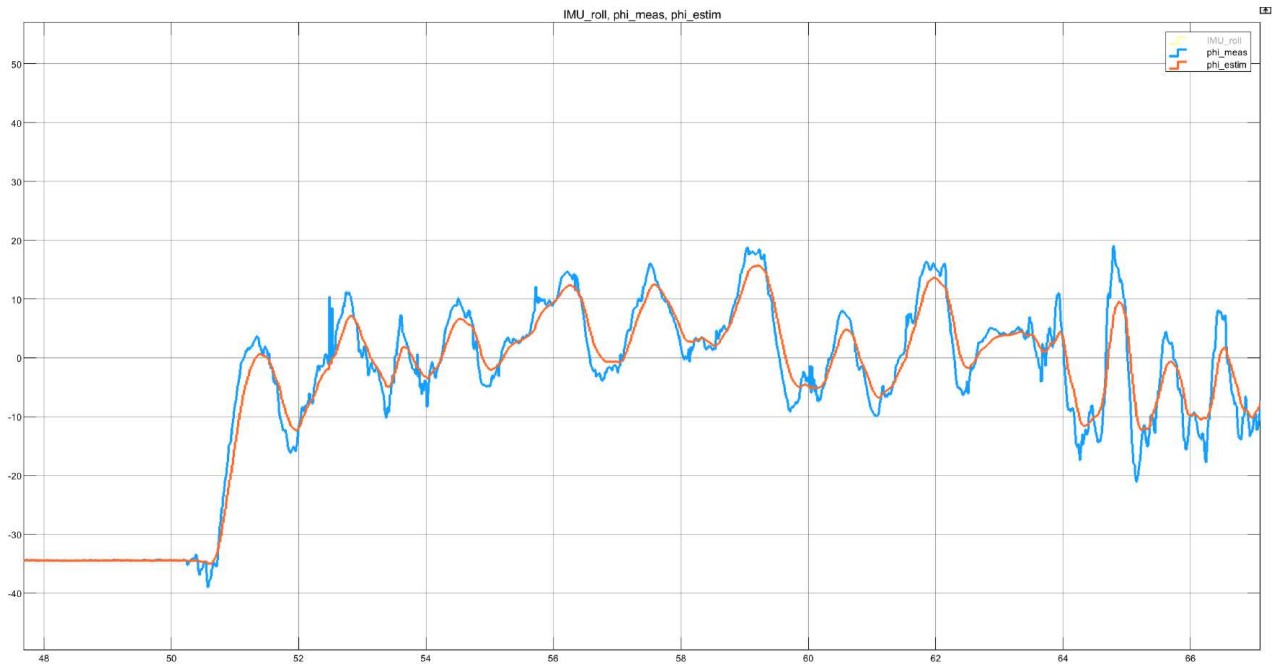

Figure 8.6: Observers Comparison: EKF (blue) vs quadcopter's Internal State Estimator (red)

9 Implementation on the Real System

9.1 Tests Setup

All the controls described in this chapter act on just 1 degree of freedom, while the quadrotor is a 3-dof system. To allow the testing, the quadrotor was physically constrained so that just 1 degree of freedom was left, as depicted in Figure 9.1.

In case of schemes aiming to control the roll angle ϕ, the quadcopter arms corresponding to motors 1 and 2 were tied with strings to the support. In this way, pitch and yaw movement were not possible and the corresponding angles were fixed at around 0°. Viceversa, in case of schemes aiming to control the pitch angle θ, the arms corresponding to motors 3 and 4 were tied, blocking roll and yaw.

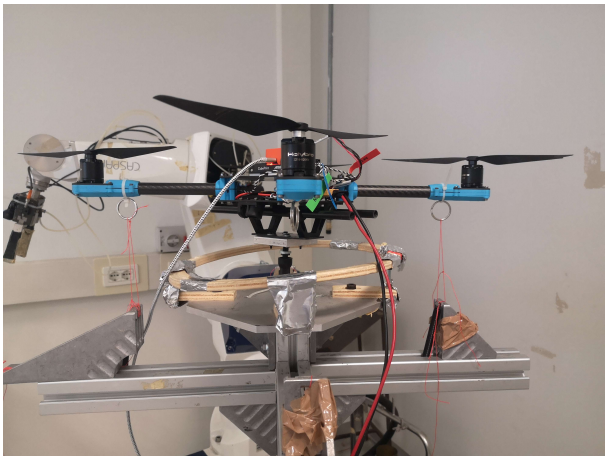

Figure 9.1: Tests Setup

(Note: in the control schemes the custom designed EKF - see 8.5 -was used to retrieve the measures of angles and their correspondent rates.)

9.2 External disturbances

The state of the art of drone attitude control would suggest to include the aerodynamic disturbance effects directly in the linear motion Euler-Lagrange equations. Since, in this case, the quadcopter is constrained and cannot change its position in space, just orientation, this approach could not be pursued. Nevertheless, the designed control schemes are still able to reject external disturbances by an accurate tuning of their parameters.

9.3 1-dof PID Control

After testing some of the previously simulated controllers, which resulted in a negative outcome, a PID controller for the 1-dof system has been implemented, mainly because of

the more intuitive possibilities of tuning it provided.
Roll and pitch control are akin to each other, and they yield similar results.

9.3.1 1-dof PID Control - Roll

The tuning is realized on the model of the system represented by the state space equations 6.1.
The transfer function of the system can thus be written as

$$G(s) = \frac{1}{I_{xx}s^2} \tag{9.1}$$

where $I_{xx} = 0.07$. As explained in Chapter 5, the inertia value has been increased while designing the controller, in order to empirically add the effect of friction to our initial estimation. A serial PID was opted for, in the form of:

$$K_{PID,serial}(s) = \frac{k_c(\tau_i s + 1)(\tau_d s + 1)}{\tau_i s} \tag{9.2}$$

where k_c, τ_i, τ_d are the controller gain, integral time and derivative time.

This form allowed for a quick visualization of how certain properties of the feedback system (like cutting frequency, position of closed-loop poles and zeros, attenuation/amplification of signals at certain frequencies) changed with the variation of the parameters.

The PID was tailored to the system and turned into

$$K_{PID}(s) = \frac{k_c(\tau_i s + 1)(\tau_d s + 1)}{s(\tau_{Hf} s + 1)} \tag{9.3}$$

by adding a high frequency pole to make the controller realizable and by removing τ_i from the denominator, including it now into the PID gain k_c.

The PID gain was initially set as

$$k_c = k_c' I_{xx} \tag{9.4}$$

to cancel out the supposed gain of the system, where $k_c' = 35$ is set as final value, both through the study of the Bode diagram of the transfer function

$$L(s) = G(s) \cdot K_{PID}(s) \tag{9.5}$$

and through a trial-and-error procedure on the real system.
The value of k_c' has a role in defining ω_c of the feedback system, together with zeros, which are set such that $\tau_i = 0.5$ and $\tau_d = 0.5$.

The pole is set such that $\tau_{Hf} = \frac{1}{130}$.
Its function is to lower the magnitude of the transfer function L(s) at high frequencies, thus attenuating noises in that range.

To allow the PID implementation on the real system, the saturation of the control variable and an anti-windup scheme are needed, as shown in Figure 9.3.1.

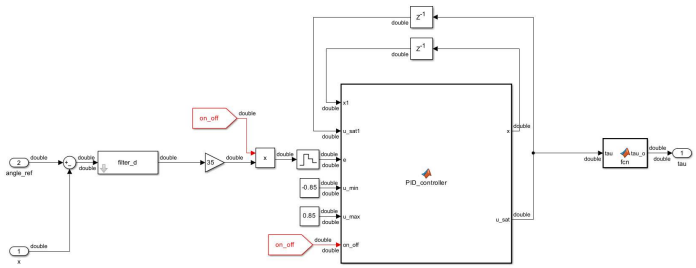

Figure 9.2: PID control for the real 1-dof system - Roll

The anti-windup method proposed by Hanus is the one applied in the closed-loop schemes. In fact, it can be employed with any general structure of a controller, while other schemes are only specific for a PID.
Firstly, the regulator is converted in discrete time and rewritten in state-space form, obtaining its state-space representation:

$$R_{pid} : \begin{cases} \dot{x}_{pid}(k+1) = A_{pid}x_{pid}(k) + B_{pid}u_{sat}(k) \\ u(k) = C_{pid}x_{pid}(k) + D_{pid}e(k) \\ u_{sat}(k) = max(u_{min}, min(u_{max}, u(k))) \end{cases} \tag{9.6}$$

Secondly, the controller R_{pid} is implemented in its "conditioned" form (with the anti-windup scheme integrated):

$$R_{pid}^{cond} : \begin{cases} x_{pid}(k+1) = A_{cond}x_{pid}(k) + B_{cond}u_{sat}(k) \\ u(k) = C_{pid}x_{pid}(k) + D_{pid}e(k) \\ u_{sat}(k) = max(u_{min}, min(u_{max}, u(k))) \end{cases} \tag{9.7}$$

where the conditioned matrices are:

$$\begin{aligned} A_{COND} &= A_{pid} - B_{pid} \cdot inv(D_{pid}) \cdot C_{pid} \\ B_{COND} &= B_{pid} \cdot inv(D_{pid}) \end{aligned} \tag{9.8}$$

The idea behind this method is to limit the action of the error e in the controller state-equation while it is in a saturation condition, keeping the input, state and output variables of the controller always consistent.

Four things to note:

1. The roll angle error is filtered through a low-pass filtered in the form of

$$Filter(s) = \frac{1}{1 + \frac{s}{120}} \tag{9.9}$$

   which is then discretized. Its function is to filter the unwanted sensor measurements noises at high frequencies, thus reducing oscillations of the system response.
2. The gain $k_c' = 35$ is set outside of the PID block: this allows for quick retuning during the testing on the system.
3. The on_off command allows to initialize the PID block state and output to 0 when the system is turned on.
4. A MATLAB function is used to create the vector of control variables for the 3-dof system, starting from the computed 1-dof control action:

$$\tau = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} \tau_\phi \\ 0 \\ 0 \end{bmatrix} \tag{9.10}$$

The testing on the real system was realized with step references and took into account the possibility of input disturbances, as shown in Figure 9.3.1 and Video 1 and 2.

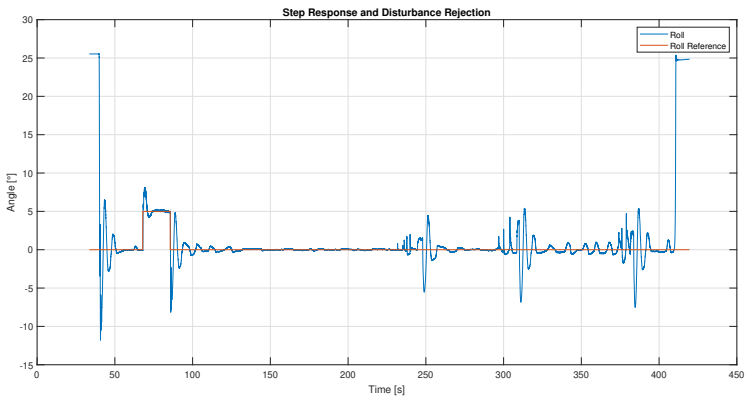

Figure 9.3: 1-dof PID - Roll angle tracking

The quadrotor is initially powered off and it is laying onto the wooden support. First, the quadcopter is brought to the equilibrium position of $\phi = 0°$, then a step reference of 5° is given. Both roll references are followed, after a transitory characterized by an initial sovraelongation and damped oscillations.
After that, the system is brought back to the equilibrium, and input disturbance rejection is tested by manually interfering three times with the quadrotor torques. The system reacts with a damped oscillatory motion and, in the case of an impulsive disturbance,

manages to go back to $\phi = 0°$.
Finally, the quadrotor is powered off again.

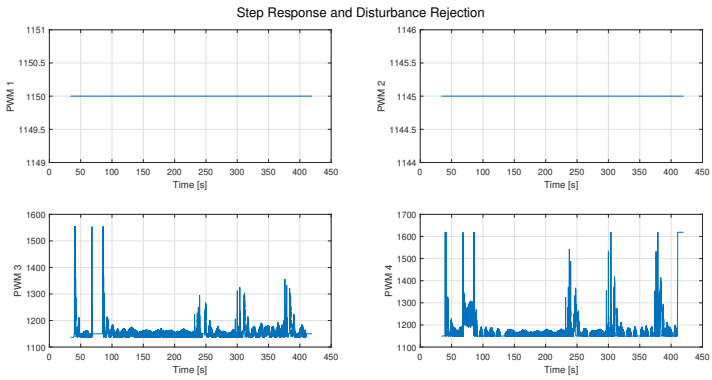

Figure 9.4: 1-dof PID - PWM (Roll)

The PWM of the motors are shown in Figure 9.3.1. From the plots it is visible how PWM1 and PWM2, which are not involved with roll motion, are kept at their minimum value, while PWM3 and PWM4 vary according to the given reference or to counteract disturbances.

9.3.2 1-dof PID Control - Pitch

The same 1-dof control structure was applied on the system with pitch angle θ as controlled variable.
In Figure 9.5, step reference tracking and disturbance rejection are shown, with results similar to the roll case. In Figure 9.6 the PWM are shown: contrary to roll control, in pitch control PWM_1 and PWM_2 vary accordingly to the control action, while PWM_3 and PWM_4 are kept to their minimum value.

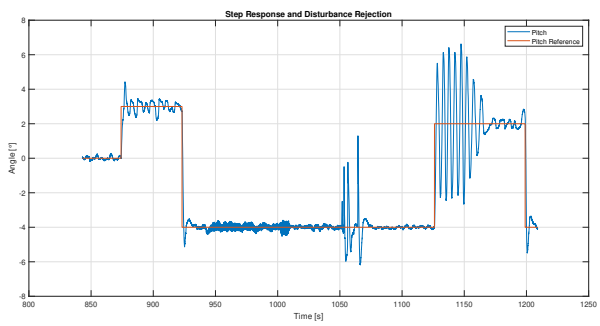

Figure 9.5: 1-dof PID - Pitch angle tracking

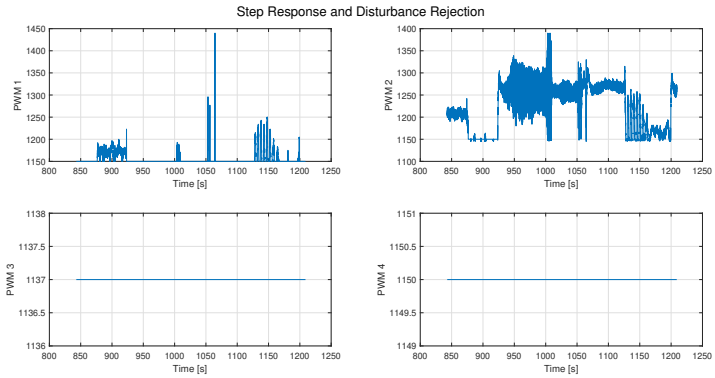

Figure 9.6: 1-dof PID - PWM (Pitch)

9.3.3 Allocation Logic

The control schemes are equipped with an Allocation Logic Block, which computes the corresponding control command $\text{PWM} = [\text{PWM}_1\ \text{PWM}_2\ \text{PWM}_3\ \text{PWM}_4]'$, given the torque command $\tau = [\tau_\phi\ \tau_\theta\ \tau_\psi]'$.
This particular type of allocation only works for 1-dof and 2-dof control and is equipped on every implemented scheme.
This is based on the fact that the torque and the motors angular velocity vectors are connected by the relation

$$\tau = M_{\text{PWM}\to\tau} \cdot \text{PWM} = \begin{bmatrix} 0 & 0 & l \cdot k_3 & -l \cdot k_4 \\ -l \cdot k_1 & l \cdot k_2 & 0 & 0 \\ -b & -b & b & b \end{bmatrix} \cdot \text{PWM} \tag{9.11}$$

By looking at the first two rows of matrix $M_{PWM\to\tau}$, one can see that τ_ϕ and τ_θ are related only to PWM_3, PWM_4, and PWM_1, PWM_2 respectively. By setting PWM_{max} and PWM_{min}, maximum and minimum values allowed on the motors velocities, it is possible to allocate the values of PWM during the roll and pitch movements.

In particular, the logic is:

$$\begin{array}{ll} \text{if } \tau_\phi \geq 0, \text{ then:} & \text{otherwise:} \\ \begin{cases} \text{PWM}_4 = \text{PWM}_{min} \\ \text{PWM}_3 = \dfrac{\tau_\phi + l \cdot k_4 \cdot \text{PWM}_4}{l \cdot k_3} \end{cases} & \begin{cases} \text{PWM}_3 = \text{PWM}_{min} \\ \text{PWM}_4 = \dfrac{-\tau_\phi + l \cdot k_3 \cdot \text{PWM}_3}{l \cdot k_4} \end{cases} \end{array} \tag{9.12}$$

$$\begin{array}{ll} \text{if } \tau_\theta \geq 0, \text{ then:} & \text{otherwise:} \\ \begin{cases} \text{PWM}_1 = \text{PWM}_{min} \\ \text{PWM}_2 = \dfrac{\tau_\theta + l \cdot k_1 \cdot \text{PWM}_1}{l \cdot k_2} \end{cases} & \begin{cases} \text{PWM}_2 = \text{PWM}_{min} \\ \text{PWM}_1 = \dfrac{-\tau_\theta + l \cdot k_2 \cdot \text{PWM}_2}{l \cdot k_1} \end{cases} \end{array} \tag{9.13}$$

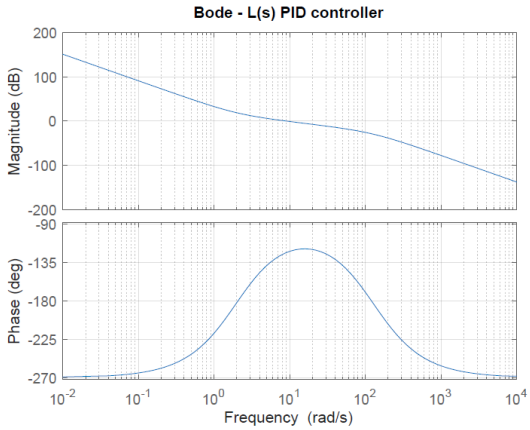

Figure 9.7: Bode - PID controller

To keep the control loop as linear as possible, it was decided to constrain the control action between $\text{PWM}_{min} = 150$ and $\text{PWM}_{max} = 700$, since, from the Thrust-PWM identification, 700 was the end of the linear region.

9.3.4 L(s) analysis

By looking at the Simulink scheme developed in Figure 9.3.1, the following open loop transfer function of the PID controller has been computed:

$$L(s) = G(s) \cdot K_{PID}(s) \cdot Filter(s) \tag{9.14}$$

The corresponding Bode diagram is represented in Figure 9.7 and it is characterized by a margin phase of 56.9° at a cutting frequency of 9.12 rad/s. Notice that the low value of the margin phase is responsible for the oscillations that have been observed during the testing phase of this controller on the real system.

9.3.5 Closed loop Identification

The step response of the overall controlled system was fitted with a second order transfer function with delay, with the aim of verifying the declared characteristics of the controller. This optimization was performed by exploiting Python Scipy library, which allows to fit complicated transfer functions to a function of the form:

$$F(s) = \frac{Ke^{-\theta s}}{(\tau_1 s + 1)(\tau_2 s + 1)} \tag{9.15}$$

The fitting problem gave the results illustrated in Table 9.1 and Figure 9.8: The cut off frequency of this new simplified transfer function is 1.0193 rad/s, which is quite close to the desired one.

Parameter	Value
K	0.08777160165775438
τ_1	0.768362201062388
τ_2	0.4773486513279449
θ	4.391660581719733

Table 9.1: PID Fitting Results

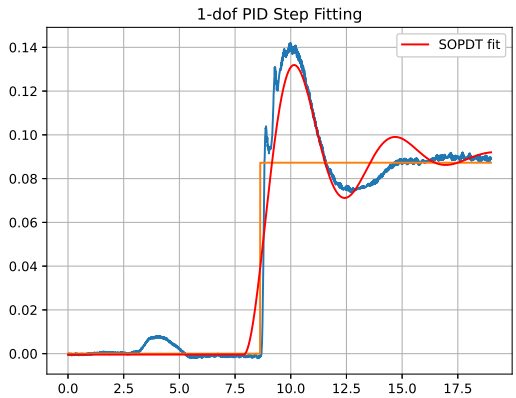

Figure 9.8: PID Step Fitting

9.4 1-dof Pole Placement Control - Pitch

After creating a PID controller for both pitch and roll, the information obtained were used to develop a pole placement controller through the transfer function method. Firstly, the closed loop poles were computed by simply obtaining the roots of the polynomial P(s), given by the sum of the numerator of L(s) with its denominator. These roots were then used as the free design parameters needed for the pole placement method.

This method uses the scheme in Figure 9.9 where the first block represents the regu-

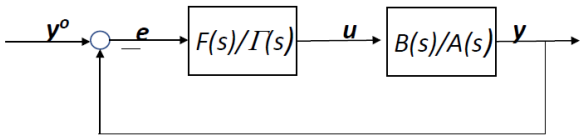

Figure 9.9: Pole Placement implementation scheme

lator, while the second block represents the system. In particular, it is possible to include an integral action into the regulator by enlarging the system and considering the system with transfer function:

$$\tilde{G}(s) = G(s)\frac{1}{s} \tag{9.16}$$

inside the scheme of Figure 9.10:

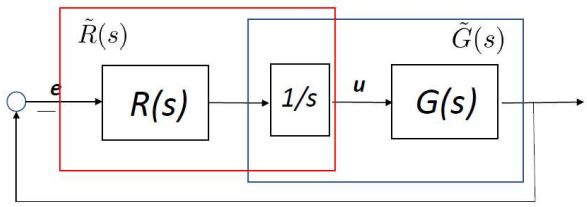

Figure 9.10: Pole Placement enlarged scheme

With the regulator written in the form:

$$R(s) = \frac{F(s)}{\Gamma(s)} = \frac{f_2 \cdot s^2 + f_1 \cdot s + f_0}{\gamma_2 \cdot s^2 + \gamma_1 \cdot s + \gamma_0} \tag{9.17}$$

the Diophantine Equation was solved

$$A(s) \cdot \Gamma(s) + B(s) \cdot F(s) = P(s) \tag{9.18}$$

thus obtaining the coefficients of $R(S)$.
Since the enlarged scheme is considered, the actual regulator is:

$$\tilde{R}(s) = R(s)\frac{1}{s} \tag{9.19}$$

implemented on $G(s)$.

To check that the system with the new implemented regulator had the same properties of the PID case, the bode plot was computed and is reported in Figure 9.11.

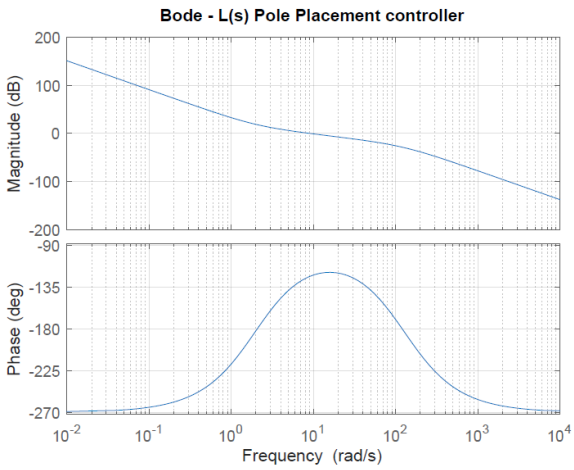

Figure 9.11: Bode - Pole Placement controller

From the picture it is possible to claim that the margin phase and the cutoff frequency are exactly the same of the PID case, thus also the response of the system in closed loop will be the same.

Before implementing the Simulink scheme to check this result, a last step was done. The regulator obtained could be considered as the product of:

$$\tilde{R}(s) = R_{true}(s) \cdot k_c' \cdot Filter(s) \tag{9.20}$$

where

- R_{true} is the actual regulator used, and also the one on which the saturation of the control variable and an anti-windup scheme are implemented as in the PID case. In particular, its expression is the following:

$$R_{true}(s) = \frac{9.555 \cdot s^3 + 1.185e06 \cdot s^2 + 4.625e06 \cdot s + 4.586e06}{4200 \cdot s^3 + 1.05e06 \cdot s^2 + 6.552e07 \cdot s} \tag{9.21}$$

- $k_c' = 35$ is taken from the PID case and here again used externally to allow an eventual a posteriori tuning of the system during testing.
- $Filter(s)$ is the same filter used in PID.

The final scheme implemented in Simulink has the very same structure of the PID one (9.3.1) but with the pole placement regulator inside the Controller block.

The testing on the real system was realized with step references as in PID case, and took into account the possibility of input disturbances, as shown in Figure 9.12 and Video 3 and 4.

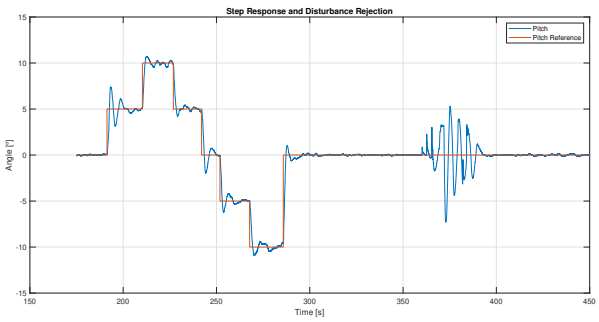

Figure 9.12: 1-dof Pole Placement - Pitch angle tracking

Starting from the equilibrium position, the following steps were given to the quadcopter: +5, +10, +5, 0, -5, -10, 0. It is possible to see that all this references are correctly followed, with very small sovraelongations when the step is given.
The system is then kept at the equilibrium position for a certain time, after which it is externally perturbed with forces of different intensities. Despite this, the system always tries and manages to stay around θ=0°.

For what concerns the PWM values reported in Figure 9.13, it is possible to appreciate

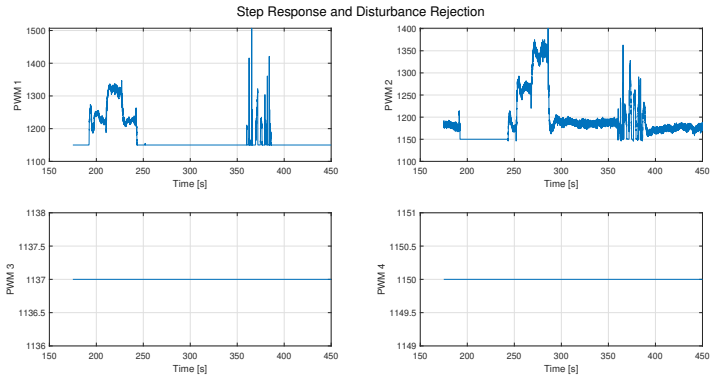

Figure 9.13: 1-dof Pole Placement - PWM

the fact that, since the roll and yaw motions of the quadcopter are blocked, the PWM related to motors 3 and 4 are set to their minimum, while the ones related to motors 1 and 2 vary depending on the references and/or inputs given to the system.

9.4.1 Closed loop Identification

The system identification previously performed on PID has been recomputed for the pole placement controller, by considering 9.15. The cut off frequency of the simplified system is 1.2921 [rad/s], which is close to the theoretical one.
Fitting coefficients are displayed in Table 9.2 ans the results are in Figure 9.14.

Parameter	Value
K	0.09071944373419887
τ_1	0.7006324044217673
τ_2	0.24049058356078076
θ	7.884099504481733

Table 9.2: Pole Placement Fitting Results

9.5 1-dof H2 Control - Pitch

With the same approach of the pole placement control, the H2 control was implemented starting from the knowledge of the open loop transfer function $L(s)$ that already had guaranteed the stabilization of the corresponding closed loop system. Moreover, the implementation was also based on some results obtained from the simulations done in Chapter 7.
Regarding this, the stabilized system was represented by the transfer function previously called $G_{stable,1dof}$, which corresponds to the system stabilized by a state feedback control

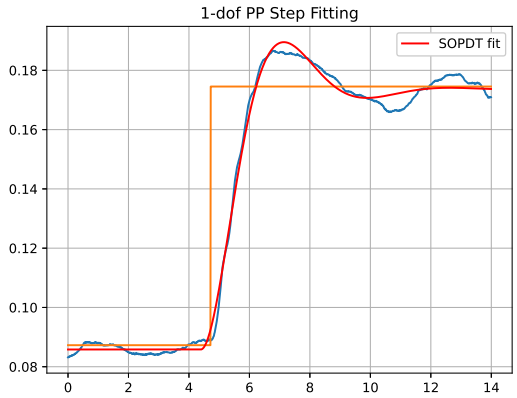

Figure 9.14: Pole Placement Step Fitting

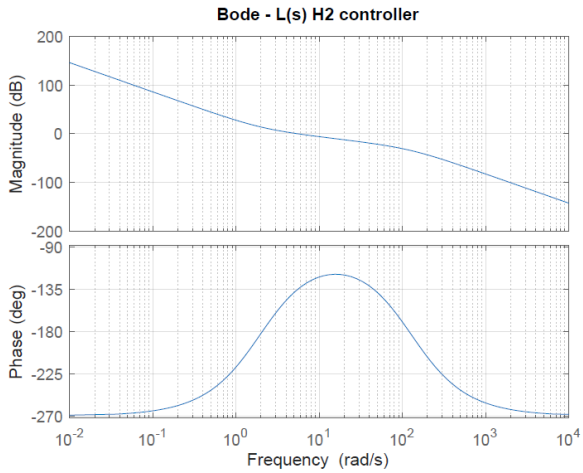

Figure 9.15: Bode - H2 controller

law. Knowing this, the regulator was found by:

$$R(s) = \frac{L(s)}{G_{stable,1dof}} \tag{9.22}$$

To check that the system controlled by this controller would have a similar behaviour to the one controlled by PID or pole placement, the open loop transfer function was again re-computed by inverting the above relationship 9.22. This was done because of the computational precision of MATLAB, which is not able to cancel pole and zeros even when they are the same. Due to this, the order of both the numerator and denominator of the regulator appeared to be higher with respect to the one obtained by manual computations.

The main consequence was that the bode plot of $L(s)$ was similar to the ones of PID and pole placement but with a phase margin of 45.1° at a cutting frequency of 5.53 rad/s, as shown in Figure 9.15

To counteract this effect the open loop transfer function could be read as:

$$L(s) = R_{true}(s) \cdot k_c' \cdot Filter(s) \cdot G_{stable,1dof} \tag{9.23}$$

In this way, by setting k_c' to the value of 62.5, instead of the previous 35, it was possible to re-obtain the same phase margin at the same cutting frequency for $L(s)$. To understand which would have been the behaviour of the quadcopter with these two different gains, the Simulink scheme for H2 was implemented, together with the saturation of the control variable and anti-windup schemes for $R_{true}(s)$, whose final expression is:

$$R_{true}(s) = \frac{(s+2)^2 \cdot (s^2 + 3.279 \cdot s + 3.211)}{s^3 \cdot (s + 130)} \tag{9.24}$$

The Simulink scheme for H2 has the same main structure as the PID one 9.3.1.
Inside the H2 controller block, at every sampling instant, the state of the controller is computed as done in the PID case, with the difference that the output of the regulator is not the control variable, but one of its two components called γ. As done for the PID case, the H2 controller is implemented in its conditioned form to avoid windup phenomenon.
The original control action is at first discretized over time, and then its state-space representation is computed, obtaining:

$$\begin{cases} x_{H2}(k+1) = A_{H2}x_{H2}(k) + B_{H2}e(k) \\ \gamma(k) = C_{H2}x_{H2}(k) + D_{H2}e(k) \\ u(k) = \gamma(k) + K_{H2}x_{sys}(k) \\ u_{sat}(k) = max(u_{min}, min(u_{max}, u(k)) \\ \gamma_{sat} = u_{sat}(k) - K_{H2}x_{sys}(k) \end{cases} \tag{9.25}$$

Then its conditioned form is implemented:

$$\begin{cases} x_{H2}(k+1) = A_{cond}x_{H2}(k) + B_{cond}\gamma_{sat}(k) \\ \gamma(k) = C_{H2}x_{H2}(k) + D_{H2}e(k) \\ u(k) = \gamma(k) + K_{H2}x_{sys}(k) \\ u_{sat}(k) = max(u_{min}, min(u_{max}, u(k)) \\ \gamma_{sat} = u_{sat}(k) - K_{H2}x_{sys}(k) \end{cases} \tag{9.26}$$

with the usual conditioned matrices 9.8. K_{H2} is the gain vector obtained in Chapter 7 and x_{sys} is the state vector of the system. u is then saturated as always between -0.85 and 0.85.

The four notes made for the PID scheme are still valid here. The most important is the one related to the gain, which was beneficial to tuning the controller in order to obtain

the desired performances. In particular, it was noticed that the system required two different gains, depending on the task that the quadcopter had to fulfill. When covering large angles, for instance to bring the quadcopter from the resting position (when laying on the wooden support) to the equilibrium position, smaller gains were needed, around the value of k_c'=35. When the quadcopter was requested instead to remain around the equilibrium position without oscillating, larger gains were needed, around the value of k_c'=62.5.

Because of the different gains, and so different regulators, needed to control the system, it is possible to rename this control as "Switching H2" and thus consider it like a nonlinear control technique. To deal with it, a gain scheduling was designed: when the pitch angle remained inside the range (-5°,+5°) for more than 5 seconds the gain was switched to k_c'=62.5. For easiness of reading, a simplified scheme is shown in Figure 9.16 to show where the gain scheduling block has been inserted with respect to the original Simulink 9.3.1.

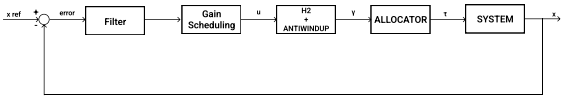

Figure 9.16: H2 Control - Simplified Scheme

The testing on the quadcopter is reported here. Initially the reference is set to the equilibrium position. After a certain time, a step of 5° is imposed, followed by a change of reference of 15° correctly tracked by the system. Then two consecutive steps of 10°, one of 5° and another one of 10° are set.
It is possible to see in Figure 9.17 that sometimes the response of the system shows some damped oscillations, but this happens only when the reference dramatically changes.

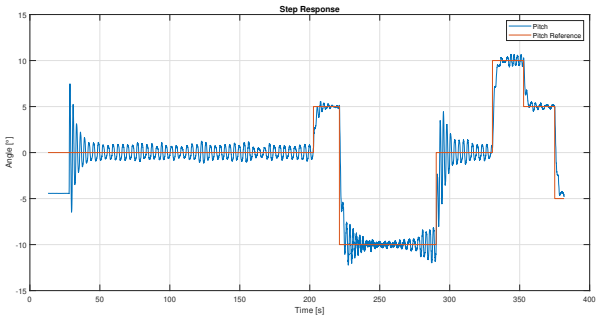

Figure 9.17: 1-dof H2 - Pitch angle tracking

Figure 9.18, instead, represents the PWM values during the above testing. As expected, PWM3 and PWM4 are set around to their minimum while motors 1 and 2 are used to keep the system as stable as possible.
The system was then manually disturbed with impulses of several intensities and also with other random inputs. The results are illustrated in the Figures 9.19, 9.20.

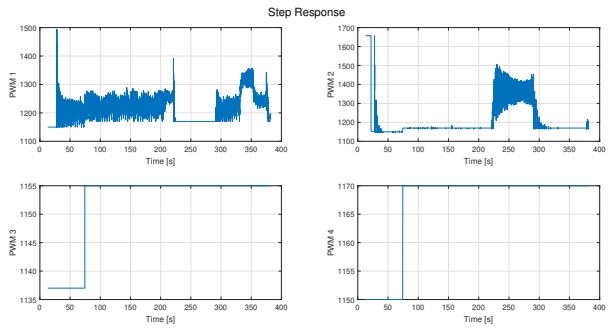

Figure 9.18: 1-dof H2 - PWM

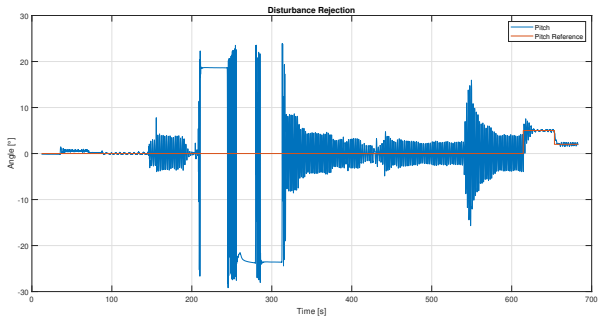

Figure 9.19: 1-dof H2 - Disturbance rejection

The control testing can be seen in Video 5 and 6 too.

9.5.1 Closed loop identification

Also for the H2 controller, the function fitting on 9.15 was performed. This time, with poor results. This was entirely due to the fact that the available acquisitions were not leaving enough time between two subsequent step references. The quadcopter, in fact, was still oscillating at the beginning of the steps and so the optimization algorithm could not converge to a realistic value, as shown in Figure 9.21.

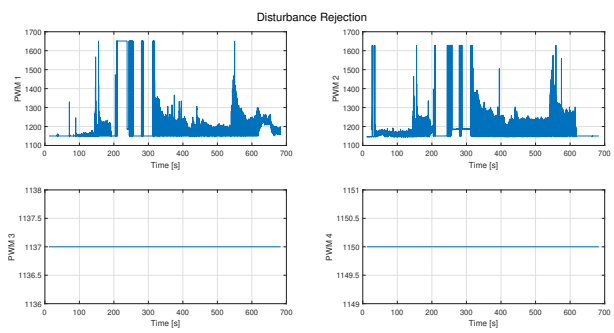

Figure 9.20: 1-dof H2 - PWM (disturbance rejection)

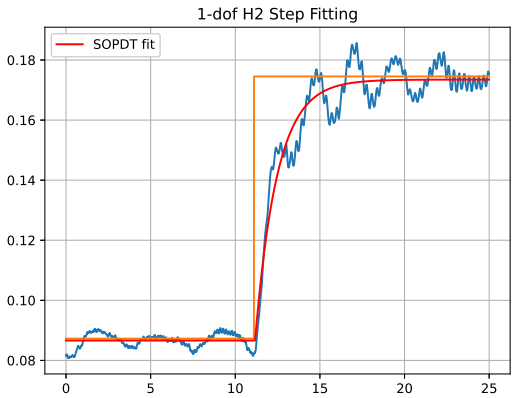

Figure 9.21: H2 Step Fitting

10 Final analysis and considerations

10.1 Comparison between the implemented controllers

In this section, the three implemented controllers are compared with one another to highlight their differences and/or similarities. This is done examining the capability of the system in closed loop to follow a step from 0° to 5° and also to reject random external disturbances.

For what concerns the step tracking case, the different responses of the system are illustrated in Figure 10.1. It is possible to claim that the H2 controller shows a better behaviour with respect to the PID regulator, characterized by a sovraelongation, and the pole placement case, which presents some damped oscillations. In fact, the system with H2 goes nearly straight to the new reference of 5°.

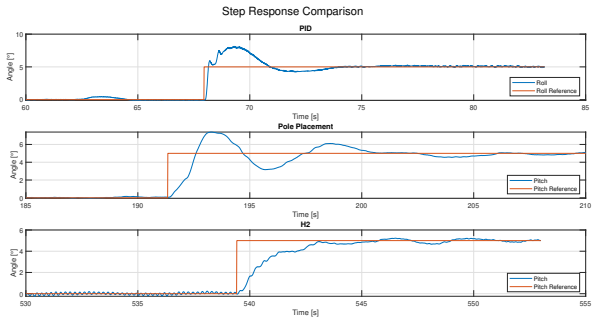

Figure 10.1: Step Response Comparison - 0° to 5° angle

Regarding the external disturbances' rejection, it is more difficult to make a comparison among the three controllers since the external inputs were random in time and in magnitude. By looking at Figure 10.2 the most robust controller seems to be the PID. After that an input is given to the quadcopter, the system reacts with a single oscillation before going back to the original equilibrium position. The pole placement regulator, instead, shows damped oscillations while the H2 controller introduces periodic oscillations.

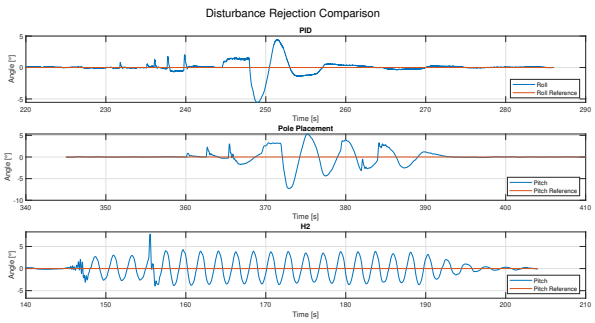

Figure 10.2: Disturbance Rejection Comparison

10.2 Other controllers

For what concerns the other two types of nonlinear controllers not discussed in Chapter 9, the inverse dynamics control was not implemented in the real system because of the uncertainty on the inertia values that in this type of controllers is fundamental not to have.
The VSC, despite its robustness, was tested on the quadcopter, but with a negative outcome. Moreover, due to its particular structure, it was not possible to follow for it the same procedure done in the previous implemented controllers based on the knowledge of the open-loop transfer function.

10.3 Challenges for 2-dof and 3-dof controllers on the real system

After the tuning of the 1-dof PID controllers, respectively for the roll and pitch motions, the results were used to develop a 2-dof PID controller. In order to design the 2-dof controller, a yaw controller would have also been needed to keep the drone as still as possible around the z-axis. It was not possible to design a yaw controller mainly for two reasons:

Magnetometer errors: the main problem in implementing a controller on the yaw dynamics with the available setup is that the main z-orientation sensor, the magnetometer, was not precise enough to reconstruct information about the yaw angle. This was mainly due to the fact that this sensor was heavily influenced by the magnetic disturbances generated by the motors and the other electric apparatuses of the laboratory. The bad performance of this sensor made it impossible to have a good yaw estimation from the EKF.

Inertia Yaw Estimation: as mentioned in Chapter 5, it was not possible to perform an identification experiment on yaw inertia I_{zz} with the available setup.

Not constraining the yaw is not a viable solution. In fact, it would lead to high accelerations, which deteriorate the performances of the EKF estimation, due to the kinematic nature of its equations. A tentative solution was trying to mechanically block the yaw position, which was impossible in the actual setup without constraining also roll and pitch movements. This problem could be overcome by changing the 3D joint constraining the drone to the support. A universal-joint, in fact, would limit the possible movements to just roll and pitch. It is straightforward that for the 3-dof controller the same problem holds.

List of Videos

1. 1-dof PID Control (Roll) - Step Response
2. 1-dof PID Control (Roll) - Disturbance Rejection
3. 1-dof Pole Placement Control (Pitch) - Step Response
4. 1-dof Pole Placement Control (Pitch) - Disturbance Rejection
5. 1-dof H2 Control (Pitch) - Step Response
6. 1-dof H2 Control (Pitch) - Disturbance Rejection

Bibliography

[1] Drone developement kits: X500 v2 kits. https://holybro.com/products/x500-v2-kits. Accessed: 14/03/2024.

[2] M. Farina. Lecture notes of the course: Networked control. Unpublished.

[3] R. Scattolini L. Magni. *Advanced and multivariable control.* Pitagora Editrice Bologna, 2014.

[4] T. Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics*, August 2011.

[5] N. Schiavoni P. Bolzern, R. Scattolini. *Fondamenti di controlli automatici.* McGraw-Hill Education, 2015.

[6] M. Prandini. Lecture notes of the course: Nonlinear control. Unpublished.