Model identification and flight control design for the Prometheus mapping drone

Candidate: Nicola Dal Lago

Advisor: Prof. Luca Schenato
Advisor: Prof. George Nikolakopoulos
Advisor: Emil?

Master in Control Engineering
Department of Information Engineering
2016

Abstract

Write here the abstract

Contents

1
Introduction

In these last years, a growing interest has been shown in robotics, In fact, several industries (automotive, medical, manufacturing, space, etc.), require robots to replace men in dangerous, repetitive or onerous situations. A wide area of this research is dedicated to Unmaned Aerial Vehicle (UAV) and especially the one of having the capability of Vertical TakeOff and Landing (VTOL) [1]. This kind of vehicle can be use in a variety of different scenario, for its reasonable price, small dimensions and great sensors capability. In particular, nowdays intensive research as been accomplish in the area of enviroment monitoring and exploration, accomplish with different strategies and sensors.

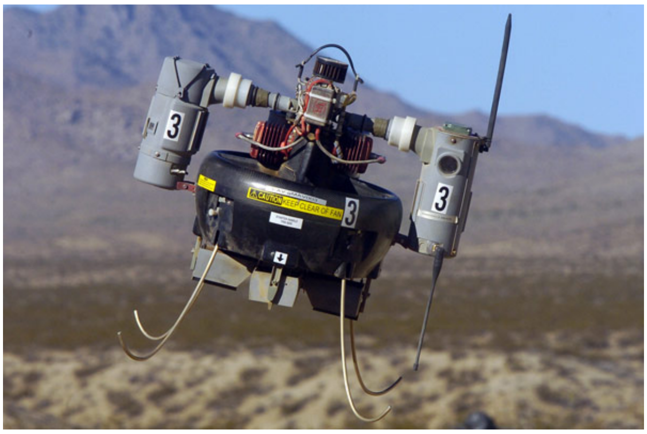

Figure 1.1: An example of UAV. T-Hawk, a US-made UAV, commonly used to search for roadside bombs in Iraq, made its debut when it photographed the Fukushima nuclear plant from above, providing a detailed look at the interior damage.

Many typse of UAVs have been developed over the last years, in particular the quadrotor type [2]. The aim of this thesis is to contribute to the develop of the so called *Prometheus project*, a fully autonomus vertical takeoff and landing vehicle, able to perform indoor enviroment exploration and mapping. To do this, we were inspired from the film Prometheus, where drones are able to map an indoor cave. Of course, due to technology and budget limitations, the vehicle will not have the same performance, but will have in theory the same capabilities. As previously said, this thesis is only a part of the project, that has been divided in three main parts:

- mechanical design and building of the UAV [3];
- mathematical model, system identification and control;
- usage of the sensors, mapping and navigation algorithms.

This thesis will focus on the second point, but briefly introductions will give also in the other two points, in particular in the mechanical design, necessary for develop a mathematical model.

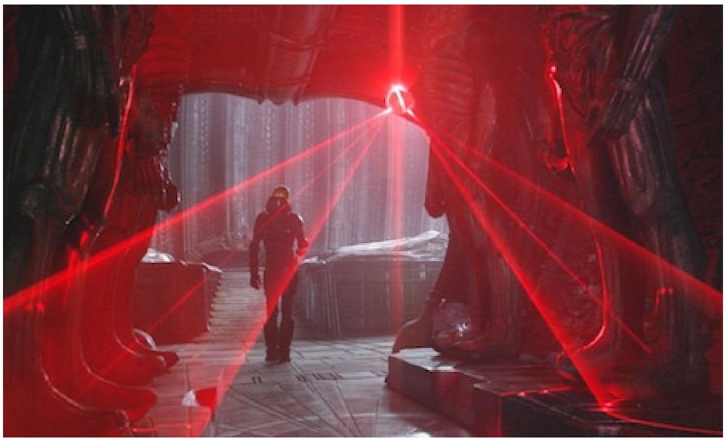

Figure 1.2: Frame of the prometheus movie, where the drone is performing the exploration and mapping of the cave.

Description of the varius chapters...

2
Design and model

In this chapter we will focus in the description of the mechanical model of the UAV and the sensor system and, from these, a mathematical model will be derived, necessary for build and simulate a control law, and to perform system identification.

2.1 Mechanical design

The overall objective of the Prometheus project is navigation and mapping, for these we mean obtain a 3D reconstruction of an unknown indoor physical environment, using a 360 degrees *Lidar* laser scanner, which, coupled to a quadrotor type UAV, will explore in a autonomus way. Lidar is a surveying technology that measures distance by illuminating a target with a laser light. Lidar is an acronym of Light Detection And Ranging, (sometimes Light Imaging, Detection, And Ranging).

Figure 2.1: Lidar laser scanner, it is able to perform a 360 degrees mapping.

Lidar is popularly used as a technology to make high-resolution maps, with applications in geodesy, geomatics, archaeology, geography, geology, geomorphology, seismology, forestry, atmospheric physics and so on. What is known as Lidar is sometimes simply referred to as laser scanning or 3D scanning, with terrestrial, airborne and mobile applications [1]. The specific Lidar laser scanner used in this project is depict in figure 2.1, where is possible to see the rotating structure moved by a motor attached in the bottom of the frame. However, this sensor is only able to perform 2D mapping and, attached to a drone, make it practically impossible to perform a complete 3D mapping. To solve this problem, several approaches could be adopted, such as use a more complicated and more expensive sensor, that can maps direclty in 3D, or just by simply use more than one Lidar. However, the solution adopted in this project is again inspired from the movie Prometheus, where the sensor is also rotating around the UAV. In such a way, the Lidar has three degrees of freedom in the movement and a 3D mapping can be performed. This solution comports, of course,the usage of only one laser scanner, but require a rotating structure that can move the sensor.

Add renders here.

In figure [renders] is possible to see clearly the platform, made of two lightweight rings, and the cart that provides the circular movement of the sensor. An important choice was also the selection of the UAV, that has to guarantee to flight also with the weight of the mechanical structure, sensor and all the electronics needed to fly and control the movement of the cart.

2.2 Mathematical model

Is pretty much clear from the previous section that this UAV is different from almost every other vehicle that is possible to buy, this of course require a complete and detailed study to characterize the mathematical model. For characterize the model, is before necessary to provide some definitions, that are also valid for standar commercial quadrotors.

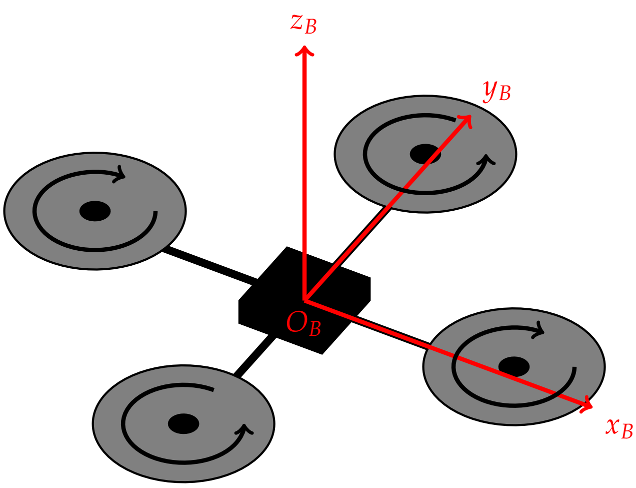

Figure 2.2: Sketch of a standard quadrotor with its body frame attach.

A quadrotor helicopter is made of a central frame and four propellers that are attach to the frame with respectively four arms. Moreover, the propellers' rotation direction

[1] https://en.wikipedia.org/wiki/Lidar

must be opposite in pairs, like illustrated in figure 2.2.
Furthermore, is necessary to define two frames, the world fixed frame and the body frame attached to the vehicle.

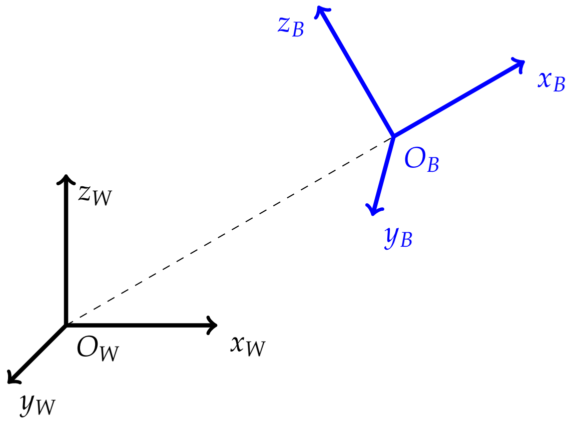

Figure 2.3: Illustration of the world and body frames. Maybe change the inclination

In figure 2.3 is possible to see the two frames, the world frame, in black, is fixed to a point and it can't be moved, the body frame, in blue, instead is attached to the quadrotor and can move with six degrees of freedom, that are position and orientation. In this, we are interesting in knowing the traslation and rotation of the body frame in respect to the world frame. For rappresent the traslation, a three dimension vector $\mathbf{x}$ is enough, that actually indicate the position of the quadrotor in the space. Instead, for the rotation, we used quaternions [4], that will be introduced in the following section.

2.2.1 Quaternion math

A quaternion is a hyper complex number of rank 4, which can be represented as follow

$$\mathbf{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T \tag{2.1}$$

The quaternion units from q_1 to q_3 are called the vector part of the quaternion, while q_0 is the scalar part [5]. Multiplication of two quaternions $\mathbf{p}$ and $\mathbf{q}$, is being performed by the Kronecker product, denoted as $\otimes$. If $\mathbf{p}$ represents one rotation and $\mathbf{q}$ represents another rotation, then $\mathbf{p} \otimes \mathbf{q}$ represents the combined rotation.

$$\mathbf{p} \otimes \mathbf{q} = \begin{bmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ p_0q_1 + p_1q_0 + p_2q_3 - p_3q_2 \\ p_0q_2 - p_1q_3 + p_2q_0 + p_3q_1 \\ p_0q_3 + p_1q_2 - p_2q_1 + p_3q_0 \end{bmatrix} \tag{2.2}$$

$$= Q(\mathbf{p})\mathbf{q} = \begin{bmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p2 & p_1 & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{2.3}$$

$$= \bar{Q}(\mathbf{q})\mathbf{p} = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \tag{2.4}$$

The norm of a quaternion is defined as

$$||\mathbf{q}|| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{2.5}$$

If the norm of the quaternion is equal to 1, then the quaternion is called unit quaternion. The complex conjugate of a quaternion has the same definition as normal complex numbers.

$$\mathbf{q}^* = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \end{bmatrix}^T \tag{2.6}$$

The inverse of a quaternion is defined as a normal inverse of a complex number.

$$\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{||\mathbf{q}||^2} \tag{2.7}$$

The time derivative of the unit quaternion is the vector of quaternion rates [6]. It requires some algebraic manipulation but is important to notice that the quaternion rates, $\dot{\mathbf{q}}$, are related to the angular velocity $\boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$. It can be represented in two ways:

- as in equation (2.8) in case that the angular velocity is in the world frame

$$\dot{\mathbf{q}}_w(\mathbf{q}, w) = \frac{1}{2}\mathbf{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} = \frac{1}{2}Q(\mathbf{q}) \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \tag{2.8}$$

- as in equation (2.9) if the angular velocity vector is in the body frame of reference.

$$\dot{\mathbf{q}}_{w'}(\mathbf{q}, w') = \frac{1}{2} \begin{bmatrix} 0 \\ \boldsymbol{\omega}' \end{bmatrix} \otimes \mathbf{q} = \frac{1}{2}\bar{Q}(\mathbf{q}) \begin{bmatrix} 0 \\ \boldsymbol{\omega}' \end{bmatrix} \tag{2.9}$$

A unit quaternion can be used also as a rotation operator, however the transformation requires both the quaternion and its conjugate, as show in equation (2.10). This rotates the vector $\mathbf{v}$ from the world frame to the body frame represented by $\mathbf{q}$.

$$\omega = \mathbf{q} \otimes \begin{bmatrix} 0 \\ \mathbf{v} \end{bmatrix} \otimes \mathbf{q}^* \tag{2.10}$$

Unit quaternion can be use also to represents rotation matrixes. Consider a vector $\mathbf{v}$ in the world frame. If $\mathbf{v}'$ is the same vector in the body coordinates, the the following relations hold

$$\begin{aligned} \begin{bmatrix} 0 \\ \mathbf{v}' \end{bmatrix} &= \mathbf{q} \cdot \begin{bmatrix} 0 \\ \mathbf{v} \end{bmatrix} \cdot \mathbf{q}^* & (2.11) \\ &= \bar{Q}(\mathbf{q})^T Q(\mathbf{q}) \begin{bmatrix} 0 \\ \mathbf{v} \end{bmatrix} & (2.12) \\ &= \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & R_{\mathbf{q}}(\mathbf{q}) \end{bmatrix} \begin{bmatrix} 0 \\ \mathbf{v} \end{bmatrix} & (2.13) \end{aligned}$$

where

$$R_{\mathbf{q}}(\mathbf{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{2.14}$$

That is,

$$\begin{aligned} \mathbf{v}' &= R_{\mathbf{q}}(\mathbf{q})\mathbf{v} & (2.15) \\ \mathbf{v} &= R_{\mathbf{q}}(\mathbf{q})^T \mathbf{v}' & (2.16) \end{aligned}$$

Just as with rotation matrices, sequences of rotations are represented by products of quaternions. That is, for unit quaternions $\mathbf{q}$ and $\mathbf{p}$, it holds that

$$R_{\mathbf{q}}(\mathbf{q} \cdot \mathbf{p}) = R_{\mathbf{q}}(\mathbf{q}) R_{\mathbf{q}}(\mathbf{p}) \tag{2.17}$$

Finally, for representing quaternion rotations in a more intuitive manner, the conversion from Euler angles (roll ϕ, pith θ and yaw ψ) to quaternion and viceversa can be performed by utilizing the following two equations respectively.

$$\begin{aligned} q &= \begin{bmatrix} \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix} & (2.18) \\ \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} &= \begin{bmatrix} \mathrm{atan2}(2(q_0q_1 + q_2q_3), q_0^2 - q_1^2 - q_2^2 + q_3^2) \\ \mathrm{asin}(2(q_0q_2 - q_3q_1)) \\ \mathrm{atan2}(2(q_0q_3 + q_1q_2), q_0^2 + q_1^2 - q_2^2 - q_3^2) \end{bmatrix} & (2.19) \end{aligned}$$

2.2.2 Quadrotor modelling

We consider first a standard quadrotor, without a rotating platform, like in figure 2.4.

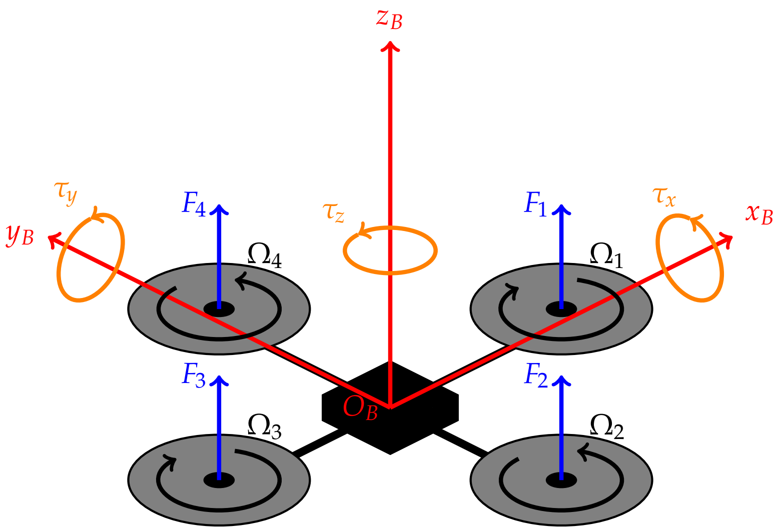

Figure 2.4: Sketch of a standard quadrotor.

In fiure 2.4 are also impres the force vectors F_i generate from each motor-propeller, the torques vectors τ_x, τ_y and τ_z about the three axis and the propeller's speed Ω_i. Now, for modeling the rigid body of a multirotor, the standard Newton-Euler kinematics equations can be utilized [7].

$$\begin{bmatrix} \mathbf{F} \\ \boldsymbol{\tau} \end{bmatrix} = \begin{bmatrix} m \cdot I_{3\times3} & \mathbf{0} \\ \mathbf{0}^T & I_{cm} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{x}}_\mathbf{B} \\ \dot{\boldsymbol{\omega}}_\mathbf{B} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\omega}_\mathbf{B} \times I_{cm} \cdot \boldsymbol{\omega}_\mathbf{B} \end{bmatrix} \tag{2.20}$$

Where $\mathbf{F} = \begin{bmatrix} F_x & F_y & F_z \end{bmatrix}^T$ is the vector of the total force, $\boldsymbol{\tau} = \begin{bmatrix} \tau_x & \tau_y & \tau_z \end{bmatrix}^T$ is the total torque, m is the mass of the quadrtotor, I_{cm} is the matrix of inertia related to the center of mass, $\ddot{\mathbf{x}}_\mathbf{B}$ is the acceleration of the quadrotor center of mass related to the body frame and $\boldsymbol{\omega}_\mathbf{B} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ is the rotational rates in the body frame.
Before deriving the torque relationship, the motors' models from the input signal to the thrust force are needed. In specific, the four input signals are the speed of the propellers u_i, map between 0 (no throttle) and 1 (full throttle). Then, the thrust for each propeller can be simply derive as follow

$$F_i(t) = A_{F,i}\Omega_i^2 = A_{F,i}\Omega_{max,i}^2 u_i(t)^2 \tag{2.21}$$

where $A_{F,i} \in \mathbb{R}_+$ are the thrust constants of the motor-propeller combination, $\Omega_{max,i} \in \mathbb{R}_+$ are the maximum rotational speed of the motors and $u_i(t)$ are the motors' signals. What is missing in equation (2.21) is the model of the DC motors and in particular, a map between the input signal $u_i(t)$ and the control signal $u_{in,i}(t)$. To keep the model simple but still accurate [2], the motor has been modeled like a delay, like in equation

[2] http://pi19404.github.io/pyVision/2015/04/10/25/

(2.22).

$$u_i(t) \approx \frac{1}{\tau_i s + 1} u_{in.i}(t) \tag{2.22}$$

This approach is very common [8], since all the parameters of a motor are not provide from datasheet, especially from cheap motors that is possible to find quite often in a commercial quadrotor. Furthermore, to represent the direction of the thrust from a motor it should be considered that

$$\mathbf{F}_i(t) = A_{F,i}\Omega^2_{max,i} u_i(t)\mathbf{n}_i \tag{2.23}$$

$$\mathbf{n}_i = R_i \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \tag{2.24}$$

Where, in this case, $\mathbf{F}_i(t)$ is the force vector for each propeller and R_i is the rotational matrix encoding the direction of the thrust and torque vector. Then the torque rappresentation is given by

$$\boldsymbol{\tau}_i(t) = -\operatorname{sgn}(\Omega_i) B_{F,i}\Omega^2_{max,i} u_i(t)^2 \mathbf{n}_i \tag{2.25}$$

where $B_{F,i} \in \mathbb{R}_+$ is the torque constant.
Now, by defining the vector $\mathbf{l}_i = \begin{bmatrix} l_{x,i} & l_{y,i} & l_{z,i} \end{bmatrix}^T$ the distance between the center of mass and the position where the propeller i is attached, combining equations (2.23), (2.24) and (2.25) is possible to obtain equation (2.26) as in the work [9].

$$\begin{bmatrix} \mathbf{F}_{total} \\ \boldsymbol{\tau}_{total} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{4} \mathbf{F}_i(u_i^2) \\ \sum_{i=1}^{4} \mathbf{l}_i \times \mathbf{F}_i(u_i^2) + \boldsymbol{\tau}_i(u_i^2) \end{bmatrix} \tag{2.26}$$

This combined with the Newton-Euler kinematics of equation (2.20) gives the final model, from control signal to acceleration and angular acceleration, as depicted in equations (2.27) and (2.28).

$$\begin{aligned} \begin{bmatrix} \ddot{\mathbf{x}}_B \\ \dot{\boldsymbol{\omega}}_B \end{bmatrix} = & \begin{bmatrix} \dots & \frac{A_{F,i}\Omega^2_{max,i}\mathbf{n}_i}{m} & \dots \\ \dots & I_{cm}^{-1}\left[(\mathbf{l}_i + \boldsymbol{\Delta l}) \times A_{F,i}\Omega^2_{max,i}\mathbf{n}_i - \operatorname{sgn}(\Omega_i) B_{F,i}\Omega^2_{max,i}\mathbf{n}_i\right] & \dots \end{bmatrix} \begin{bmatrix} \vdots \\ u_i^2 \\ \vdots \end{bmatrix} + \\ & + \begin{bmatrix} \mathbf{0} \\ I_{cm}^{-1}(\boldsymbol{\omega}_B \times I_{cm}\boldsymbol{\omega}_B) \end{bmatrix} \end{aligned} \tag{2.27}$$

$$u_i = \frac{1}{\tau_i s + 1} u_{in,1} \tag{2.28}$$

Where $\boldsymbol{\Delta l}$ is the offset vector of the center of gravity (CoG) in the body frame of reference. From the model (2.27) the linear and angular accelerations are given, is then necessary to convert those from the body frame and integrate to obtain the position $\mathbf{x}_W$ and orientation $\mathbf{q}_W$ of the quadrotor with the respect to the world frame. Then, by adding the gravity term we have

$$\ddot{\mathbf{x}}_{B,g} = R_{\mathbf{q}_W}(\mathbf{q}_W)^T \cdot \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \ddot{\mathbf{x}}_B \tag{2.29}$$

where g is the gravity costant, about 9.81, and $R_{\mathbf{q}_W}(\mathbf{q}_W)$ is the rotation matrix built from equation (2.14). To derive the velocity $\dot{\mathbf{x}}_W$ in the world frame, once again by using the rotation matrix we obtain

$$\dot{\mathbf{x}}_W = R_{\mathbf{q}_W}(\mathbf{q}_W) \cdot \dot{\mathbf{x}}_B \tag{2.30}$$

Instead, for the orientation, we use the results from the paragraph 2.2.1 and we get

$$\dot{\mathbf{q}} = \frac{1}{2} \cdot Q(\boldsymbol{\omega}) \cdot \mathbf{q} \tag{2.31}$$

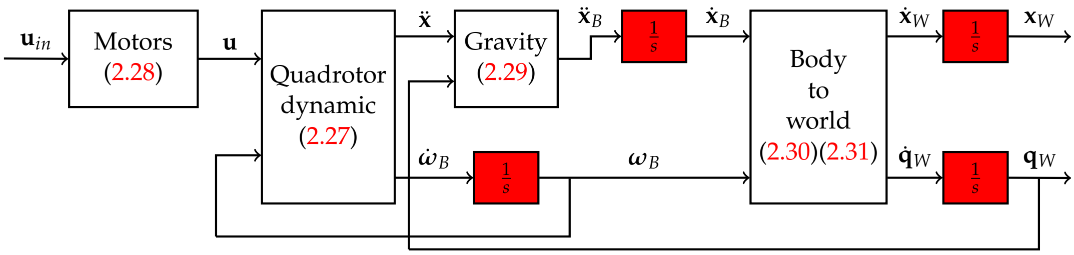

Figure 2.5: Block diagram of the quadrotor's dynamic.

In figure 2.5 is depicted a block diagram of the quadrotor dynamic, from the inputs $\mathbf{u}_{in}$, to position $\mathbf{x}_W$ and orientation $\mathbf{q}_W$ in the world frame.

2.2.3 Adding the rotating platform

Till now, all the model was designed for a standard quadrotor vehicle, what we want to do in this section is to add the model of the rotating platform, necessary for deduce a controller and simulate it.

The movement of the platform, introduces a time variant center of gravity, that is simply modelled with time variant vectors $\mathbf{l}_i(t)$, that identify the displacement of the center of the propeller i with the respect of the CoG. If we know precisely the position of the CoG of the quadrotor (without the moving cart) and the position of the CoG of the cart, the result position can be compute.

In figure 2.6 is illustrated how the resulting CoG change with the position of the cart, is possible to see also the four $\mathbf{l}_i(t)$ vectors in black dashed line. Then the position of the CoG is

$$\mathbf{p} = \frac{1}{m} \cdot \left(m_{quad}\mathbf{p}_{quad} + m_{cart}\mathbf{p}_{cart}\right) \tag{2.32}$$

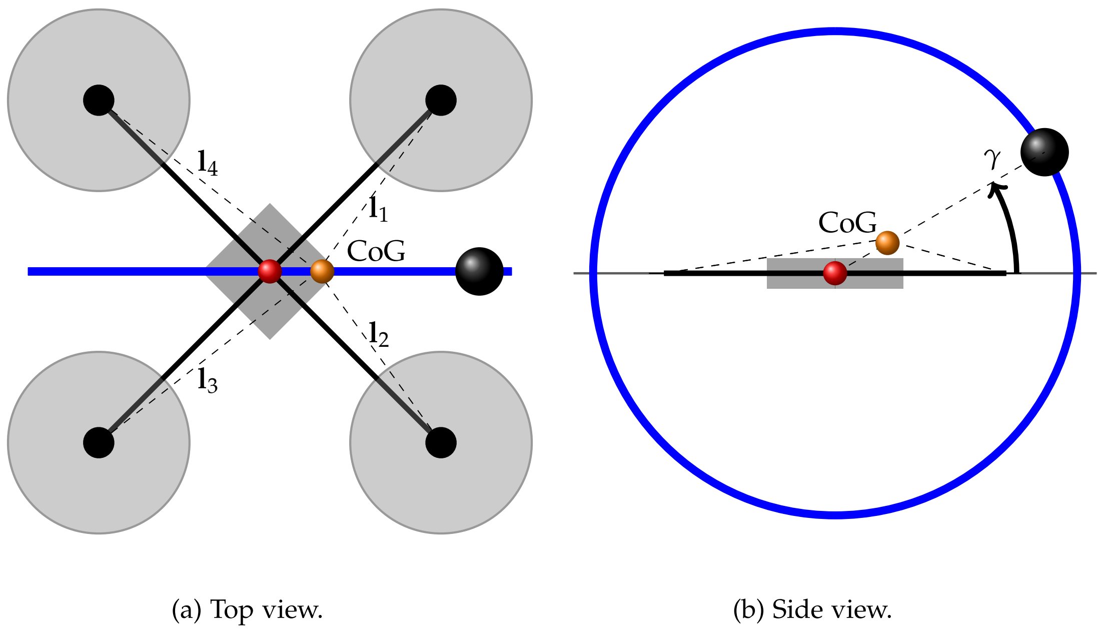

Figure 2.6: Quadrotor with the rotating platform in blue, in red the CoG of the quadrotor and in orange the resulting CoG.

where $m = m_{quad} + m_{cart}$ is the sum of the mass of the quadrotor plus the mass of the moving cart, then the total mass, $\mathbf{p}_{quad}$ is the position of the center of gravity of the quadrotor without the cart with the respect to the origin of the body frame (in general the quadrotor frame is not symmetrical) and $\mathbf{p}_{cart}$ is the position of the CoG of the cart with the respect to the body frame. Then the vectors $\mathbf{l}_i$ are just the distance between the center of the propeller i and $\mathbf{p}$.
Another difference in using the rotating platform is that the moment of inertia I_{cm} is not constant, but depend from the position γ of the cart, like in figure 2.6b. This problem can be solved by using the detailed CAD model of the entire vehicle, provided in [3]. From this is possible to deduce the inertia for various position, and then create a simple piecewise model.
Add some graphs as soon as carlos send me the CAD.
The movement of the sensor introduced also a centrifugal force in the vheicle. In particular, if $\mathbf{p}_{cart}$ is the vector that encode the position of the cart with the respect of the body frame, the Newton's law of motion for the cart in vector form is

$$\mathbf{f} = m_{cart}\mathbf{a} = m_{cart}\frac{d^2\mathbf{p}_{cart}}{dt^2} \tag{2.33}$$

By twice applying the transformation above from the stationary to the rotating frame, the absolute acceleration of the particle can be written as [10]

$$\begin{aligned}\frac{d^2\mathbf{p}_{cart}}{dt^2} &= \frac{\partial}{\partial t}\left(\frac{d\mathbf{p}_{cart}}{dt}\right) + \boldsymbol{\omega} \times \left(\frac{d\mathbf{p}_{cart}}{dt}\right) \\ &= \frac{\partial}{\partial t}\left(\frac{\partial\mathbf{p}_{cart}}{\partial t} + \boldsymbol{\omega} \times \mathbf{p}_{cart}\right) + \boldsymbol{\omega} \times \left(\frac{\partial\mathbf{p}_{cart}}{\partial t} + \boldsymbol{\omega} \times \mathbf{p}_{cart}\right)\end{aligned} \tag{2.34}$$

where in this case ω is the angular velocity of the cart with the respect to the body frame. Expanding expression (2.34), noting that the chain rule applies to differentiation of cross products, that the cross product is distributive over addition, and coupling with equation (2.33), we have

$$\mathbf{f} = m_{cart}\frac{\partial^2 \mathbf{p}_{cart}}{\partial t^2} + \underbrace{m_{cart}\frac{d\boldsymbol{\omega}}{dt} \times \mathbf{p}_{cart}}_{\text{Euler force}} + \underbrace{2m_{cart}\boldsymbol{\omega} \times \frac{\partial \mathbf{p}_{cart}}{\partial t}}_{\text{Coriolis force}} + \underbrace{m_{cart}\boldsymbol{\omega} \times \left(\boldsymbol{\omega} \times \mathbf{p}_{cart}\right)}_{\text{centrifugal force}} \tag{2.35}$$

That describe the so called Euler, Coriolis and centrifugal force of the moving platform. To add this to the main model, we just simply need to sum up the vector f, divide by m_{cart} to the equation (2.27), in the first three rows of the matrix, that reguard the acceleration of the body frame

$$\begin{bmatrix} \ddot{\mathbf{x}}_B \\ \dot{\boldsymbol{\omega}}_B \end{bmatrix} = \begin{bmatrix} \cdots & \frac{A_{F,i}\Omega^2_{max,i}\mathbf{n}_i}{m} & \cdots \\ \cdots & I_{cm}^{-1}\left[(\mathbf{l}_i + \boldsymbol{\Delta l}) \times A_{F,i}\Omega^2_{max,i}\mathbf{n}_i - \operatorname{sgn}(\Omega_i)B_{F,i}\Omega^2_{max,i}\mathbf{n}_i\right] & \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ u_i^2 \\ \vdots \end{bmatrix} + \\ + \begin{bmatrix} \mathbf{0} \\ I_{cm}^{-1}\left(\boldsymbol{\omega}_B \times I_{cm}\boldsymbol{\omega}_B\right) \end{bmatrix} + \frac{1}{m_{cart}}\begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} \tag{2.36}$$

2.3 Experimental setup

Description of all the systems, ROS, vicon, etcetera.

3
System identification

In this chapter,is about to be addressed an important part of this project. Since the model of the previous chapter is depending from many parameters, is necessary to identificate them, to be able to design an appropriate controller. A Kalman Filter approach will be used, based from the work [9].

3.1 System simplification and linear approxmation

Starting from the model deducted in section 2.2.2

$$\begin{bmatrix} \ddot{\mathbf{x}}_B \\ \dot{\boldsymbol{\omega}}_B \end{bmatrix} = \begin{bmatrix} \dots & \frac{A_{F,i}\Omega_{max,i}^2 \mathbf{n}_i}{m} & \dots \\ \dots & I_{cm}^{-1}\left[(\mathbf{l}_i + \boldsymbol{\Delta l}) \times A_{F,i}\Omega_{max,i}^2 \mathbf{n}_i - \operatorname{sgn}(\Omega_i) B_{F,i}\Omega_{max,i}^2 \mathbf{n}_i\right] & \dots \end{bmatrix} \begin{bmatrix} \vdots \\ u_i^2 \\ \vdots \end{bmatrix} + \\ + \begin{bmatrix} \mathbf{0} \\ I_{cm}^{-1}(\boldsymbol{\omega}_B \times I_{cm}\boldsymbol{\omega}_B) \end{bmatrix} \tag{3.1}$$

$$u_i = \frac{1}{\tau_i s + 1} u_{in,1} \tag{3.2}$$

we need to do some simplification. In particular, by assuming that all engines have the same parameters, is possible to rewrite these parameters as follows

$$A_{F,i}\Omega_{max,i}^2 \approx A_F \tag{3.3}$$
$$B_{F,i}\Omega_{max,i}^2 \approx B_F \tag{3.4}$$
$$\tau_i \approx \tau \tag{3.5}$$

Moreover the term $I_{cm}^{-1}(\boldsymbol{\omega} \times I_{cm}\boldsymbol{\omega}_B)$ can be neglected [9]. This can be easily seen simply by simulating the mathematical model with and without the term, the differences are very small. Add a graph with the simulation?
Another simplification, is that the inertia matrix I_{cm} is a diagonal matrix, $I_{cm} = \text{diag}(I_{xx}, I_{yy}, I_{zz})$. This is generally true in a standard quadrotor but is not so immediate for the vehicle of this project. However, if we align the x axis with the orinetation of the circular structure, we obtain a inertia matrix almost diagonal; what makes the matrix "less diagonal" is the position of the cart. However, the mass of the sensor is not sufficently big to modify enough the matrix and this assumption is valid also here. Add numbers with the CAD. Of course, in different applications, where the mass of the quadrotor and the mass of the sensor are more similar, a different approach is required.
Another non linearity is in the inputs, since the model require the square of these. A solution of this problem proposed in [9] is to rewrite equation (3.2) with the square of the control inputs. This effectively moves the squared control signal from the force and torque equations to the input. This representation keeps the static relationship but will affect the dynamics of the first order system, but is assumed that a first order system still captures the majority of the dynamics. Maybe explain it a bit better
In conclusion, the approximate linear model is

$$\begin{bmatrix} \ddot{\mathbf{x}}_B \\ \dot{\boldsymbol{\omega}}_B \end{bmatrix} = \begin{bmatrix} \cdots & \frac{A_F \mathbf{e}_3}{m} & \cdots \\ \cdots & I_{cm}^{-1}\left[(\mathbf{l}_i + \boldsymbol{\Delta l}) \times A_F \mathbf{e}_3 - \text{sgn}(\Omega_i) B_F \mathbf{e}_3\right] & \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ u_i \\ \vdots \end{bmatrix} \tag{3.6}$$

$$u_i = \frac{1}{\tau s + 1} u_{in,i}^2 \tag{3.7}$$

where instead of $\mathbf{n}_i$ there is $\mathbf{e}_3$ because in the structure of this particular vehicle, the propellers are mounted parallel to the ground and then with a force vector aligned to $\mathbf{e}_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$.

3.2 Quadrotor parameters

From the simplified model of equations (3.6) and (3.7), the identifiable parameters are

$$\boldsymbol{\beta} = \begin{bmatrix} \frac{A_F}{m} & \frac{A_F}{I_{xx}} & \frac{A_F}{I_{yy}} & \frac{A_F}{I_{zz}} & \frac{B_F}{I_{zz}} & \Delta l_x & \Delta l_y \end{bmatrix}^T, \quad \tau \tag{3.8}$$

Then, is possible to rewrite the linear model in a more compact form:

$$\begin{bmatrix} \ddot{\mathbf{x}}_B \\ \dot{\boldsymbol{\omega}}_B \end{bmatrix} = \begin{bmatrix} L(\beta_1) \\ A(\boldsymbol{\beta}) \end{bmatrix} \mathbf{u} \tag{3.9}$$

Under the assumption of the sampling rate to be much faster than the dynamics [1], equation (3.7) is implemented as discrete-time first order system, and the parameters are modeled as integrated white noise, which gives the following prediction equations

[1] in this case, thanks to the performance of the onboard electronics, the sampling rate is equal to 222 Herz.

$$\boldsymbol{\omega}_k = \boldsymbol{\omega}_{k-1} + \Delta t A(\boldsymbol{\beta}_{k-1})\mathbf{u}_{k-1} \tag{3.10}$$

$$\mathbf{u}_k = \frac{\tau_{k-1}}{\Delta t + \tau_{k-1}}\mathbf{u}_{k-1} + \frac{\Delta t}{\Delta t + \tau_{k-1}}\mathbf{u}_{in,k}^2 \tag{3.11}$$

$$\boldsymbol{\beta}_k = \boldsymbol{\beta}_{k-1} \tag{3.12}$$

$$\tau_k = \tau_{k-1} \tag{3.13}$$

where $\mathbf{u}_k$ and $\mathbf{u}_{in,k}$ are the inputs at time instant k expressed in a vectorial way, Δt is the sampling period and $\bullet^2$ is the element-wise square of a vector.

3.3 Kalman filter

A Kalman filter approach is chose for this project since it has good result in this kind of applications. Of course, better performances can be obtained with specific strategies for non linear systems [11], but these methods are in general much more complicated and require much more computational effort, especially if is necessary to estimate the parameters online.

Now, is possible to use the standard Kalman filter equations [12] to develop an online identification alghoritm as follow.

The augmented state $\mathbf{x}_{est}$ is

$$\mathbf{x}_{est} = \begin{bmatrix} \boldsymbol{\omega}_B & \mathbf{u}_{in} & \boldsymbol{\beta} & \tau \end{bmatrix}^T \in \mathbb{R}^{15} \tag{3.14}$$

The initial values of $\boldsymbol{\omega}_B$ and $\mathbf{u}_{in}$ are know, so the state is initialized with these. Moreover, due to the parameters $\boldsymbol{\beta}$ and τ having a constraint to being positive, they are implemented as $\exp(\boldsymbol{\beta})$ and $\exp(\tau)$ to force positive results from the estimation, while the $\boldsymbol{\Delta l}$ are constrained to be within the propellers (the length of the arms is set to be equal to one, since the correct length is not necessary for the identification) wich is implemented using a zero centered logistic function

$$\frac{2}{1 - \exp(-\Delta l)} - 1 \tag{3.15}$$

With the augmented state is possible to write a new state space system in discrete time with matrix A_{est} [2]

[2]For notation, $\mathbf{0}_{a\times b}$ is equal to a zero matrix with a rows and b columns,$\mathbf{1}_{a\times b}$ is equal to a ones matrix with a rows and b columns,I_a is the identity matrix of dimension $a \times a$, and the vector $\mathbf{x}_{est,a:b}$ are the entries from a to b of the augmented state (is implicit that is consider at time k)

$$
\begin{aligned}
A_{\omega,\mathbf{u}_{in}} &= \left[\begin{array}{c} 2\Delta t e^{\beta_2}(\Delta l_y - 1)\cdot \mathbf{u}^T \\ \hline -2\Delta t e^{\beta_3}(\Delta l_x + 1)\cdot \mathbf{u}^T \\ \hline -\operatorname{sgn}(\Omega_i) 2\Delta t e^{\beta_4}\cdot \mathbf{u}^T \end{array}\right] && \in \mathbb{R}^{3\times 4} \\
A_{\omega,\beta_1} &= \mathbf{0}_{3\times 1} && \in \mathbb{R}^{3\times 1} \\
A_{\omega,\beta_2} &= \begin{bmatrix} \Delta t e^{\beta_2}\left(\Delta l_y \sum_{i=1}^{4} u_i^2 - u_1^2 - u_2^2 + u_3^2 + u_4^2\right) & 0 & 0 \end{bmatrix}^T && \in \mathbb{R}^{3\times 1} \\
A_{\omega,\beta_3} &= \begin{bmatrix} 0 & -\Delta t e^{\beta_3}\left(\Delta l_y \sum_{i=1}^{4} u_i^2 + u_1^2 - u_2^2 - u_3^2 + u_4^2\right) & 0 \end{bmatrix}^T && \in \mathbb{R}^{3\times 1} \\
A_{\omega,\beta_4} &= \begin{bmatrix} 0 & 0 & -\Delta t e^{\beta_4} \sum_{i=1}^{4} \operatorname{sgn}(\Omega_i) u_i^2 \end{bmatrix}^T && \in \mathbb{R}^{3\times 1} \\
A_{\omega,\beta_5} &= \begin{bmatrix} 0 & 0 & \Delta t \end{bmatrix}^T && \in \mathbb{R}^{3\times 1} \\
A_{\omega,\beta_{6:7}} &= \begin{bmatrix} 0 & \Delta t e^{\beta_2} \sum_{i=1}^{4} u_i^2 \\ -\Delta t e^{\beta_3} \sum_{i=1}^{4} u_i^2 & 0 \\ 0 & 0 \end{bmatrix} && \in \mathbb{R}^{3\times 2} \\
A_{\omega,\beta} &= \left[\begin{array}{c|c|c|c|c|c} A_{\omega,\beta_1} & A_{\omega,\beta_2} & A_{\omega,\beta_3} & A_{\omega,\beta_4} & A_{\omega,\beta_5} & A_{\omega,\beta_{6:7}} \end{array}\right] && \in \mathbb{R}^{3\times 7} \\
A_{\mathbf{u}_{in}} &= \left(1 - \frac{\Delta t}{\Delta t + e^{\tau}}\right)\cdot I_4 && \in \mathbb{R}^{4\times 4} \\
A_{est,k} &= \left[\begin{array}{c|c|c|c} I_3 & A_{\omega,\mathbf{u}_{in}} & A_{\omega,\beta} & \mathbf{0}_{3\times 1} \\ \hline \mathbf{0}_{4\times 3} & A_{\mathbf{u}_{in}} & \multicolumn{2}{c}{\mathbf{0}_{4\times 8}} \\ \hline \multicolumn{2}{c|}{\mathbf{0}_{8\times 7}} & \multicolumn{2}{c}{I_8} \end{array}\right] && \in \mathbb{R}^{15\times 15}
\end{aligned}
$$

and then use the Kalman filter equations in a recursive way [12]

$$
\begin{aligned}
P_k &= A_{est,k}\cdot P_{k-1}\cdot A_{est,k}^T + Q && \in \mathbb{R}^{15\times 15} \\
H_k &= \left[\begin{array}{c|c|c|c|c} I_3 & \multicolumn{4}{c}{\mathbf{0}_{3\times 12}} \\ \hline \mathbf{0}_{1\times 3} & 2e^{\beta_1}\cdot \mathbf{u}^T & \mathbf{0}_{1\times 2} & e^{\beta_1}\sum_{i=1}^{4} u_i^2 & \mathbf{0}_{1\times 5} \end{array}\right] && \in \mathbb{R}^{4\times 15} \\
S_k &= H_k\cdot P_k\cdot H_k^T + R && \in \mathbb{R}^{4\times 4} \\
K_k &= P_k\cdot H_k^T\cdot S_k^{-1} && \in \mathbb{R}^{15\times 4} \\
P_k &= \left(I_{15} - K_k\cdot H_k\right)\cdot P_{k-1} && \in \mathbb{R}^{15\times 15} \\
\mathbf{x}_{est,k} &= \mathbf{x}_{est,k-1} + K_k\cdot\left(\begin{bmatrix} \boldsymbol{\omega} \\ \ddot{x}_z \end{bmatrix} - \begin{bmatrix} \mathbf{x}_{est,1:3} \\ e^{\beta_1}\cdot \mathbf{1}_{1\times 4}\cdot \mathbf{x}_{est,4:7}^2 \end{bmatrix}\right) && \in \mathbb{R}^{15\times 1}
\end{aligned}
$$

where P_k is the state update covariance matrix based on model, H_k maps the measurement to the states, S_k is the update measurement covariance, K_k is the update Kalman gain, $Q \in \mathbb{R}^{15\times 15}$ is the fixed covariance matrix and $R \in \mathbb{R}^{4\times 4}$ the fixed measurement covariance matrix. Both Q and R are diagonal matrices.

3.4 Results

The estimator needs to be setup with specific process and measurement covariance Q and R, and the starting state covariance P_0. The values for Q and P_0 where found simply with a trial and error procedure, while R was taken from the noise densities of the measured signals. In particular we measured a steady state position of the quadrotor, record the acceleration in the z axis $\ddot{x}_{B,z}$ and the angular rate $\boldsymbol{\omega}_B$, then by analyzing these data, a noise variance was extracted. Moreover, since in the augmented state $\mathbf{x}_{est}$ is present also an estimation of the angular rate $\hat{\boldsymbol{\omega}}$, to evaluate the quality of the estimation was also compare it with the measured angular rate. In this case the initial values of the state $\mathbf{x}_{est}$ were chose to be considerably different from a real value, just to show the performace of the etimator.

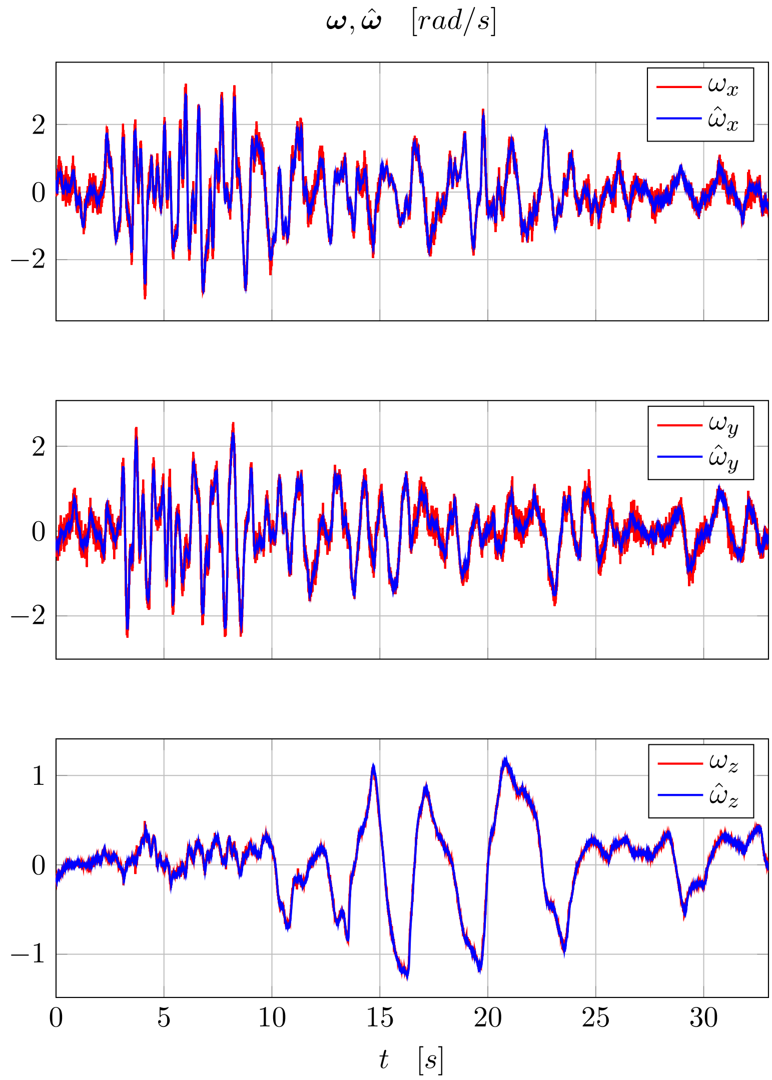

Figure 3.1: Measured and estimated angular rate, $\boldsymbol{\omega}_B$ and $\hat{\boldsymbol{\omega}}_B$.

As is possible to see in figure 3.1, the estimation of the angular rate works pretty well from the begining for all the three axis. In figure 3.2 are instead plotted the estimations of all parameters $\boldsymbol{\beta}$ and τ. The identification was performed with the cart fixed in one position. Is possible to see that for almost all parameters there is convergence after about 15 seconds, while for the parameter $\frac{B_F}{I_{zz}}$ is necessary more time. This can be explain by observing the angular rate, in particular note that ω_z is almost zero for about the first 10 seconds, due to the particular trajectory of the vehicle. Of course, is not possible to perform system identification without excitation of the system and thats why the parameters depending on ω_z require more time to converge.

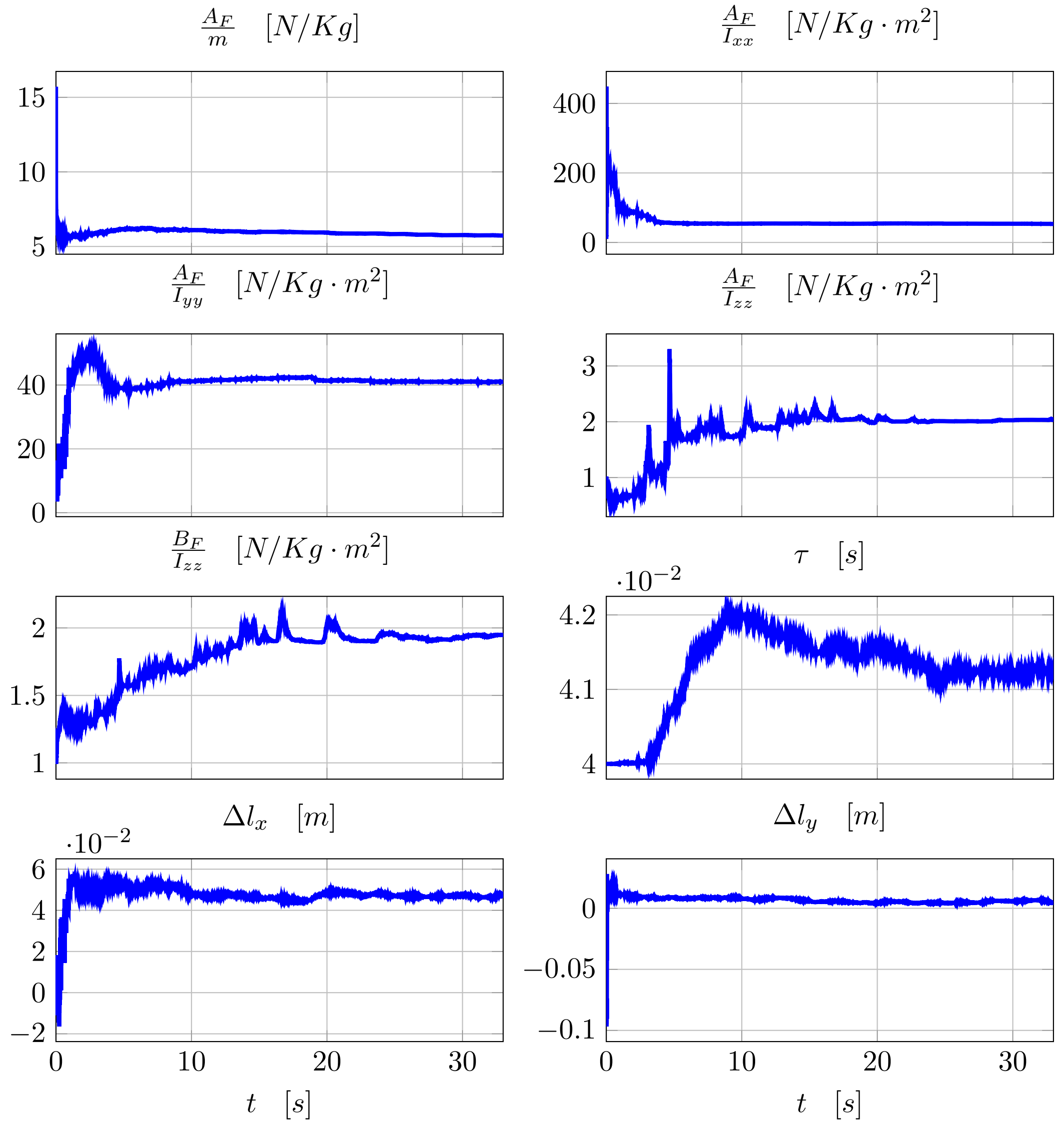

Figure 3.2: Estimated parameers $\boldsymbol{\beta}$ and τ.

In conclusion the alghoritm has been shown to work quite well, and for this application is not then necessary to use more sopisticated techniques.

4 Trajectories generator

An important aspect of this project is the path plannig, because the aim is to map and navigate in a enviroment without a priori information and in complete autonomous. The study of path planning alghoritms and the sensors' fusion to obtain the pose of the vehicle is not part of this thesis, however part of this thesis is to generate a trajectory for the UAV based on the output of a path planning algorithm. In particular, usually exploration algorithms provide only setpoints, not full trajectories [13] [14]. Is then necessary to provide a tool to generate possible trajectories with constraints in the enviroment and in the dynamic of the vehicle. In this project we implemented the solution proposed in the works [15] and [16], where the authors, after providing model and control, generate a trajectory composed by piecewise polynomial functions.

4.1 Trajectoy definition

In this work, a *setpoint* $\boldsymbol{\sigma}_d$is defined as a position in the space, $\mathbf{x}_d$, and a yaw angle, ψ_d, since in the next section we will control four degrees of freedom of the UAV, the position in the space and the yaw angle. We consider the problem of navigate through m setpoints at specific time. A trivial trajectory is one that interpolate the setpoints using straight lines. However, such trajectory is inefficient because it require the quadrotor to come to a stop st each setpoint. This method generates trjectories that smoothly transition throught the setpoints at given times. We write down a trajectory as piecewise polynomial functions of order n over m time intervals

$$\sigma_d(t) = \begin{cases} \sum_{i=0}^{n} \sigma_{d,i,1} t^i & t_0 \leq t < t_1 \\ \sum_{i=0}^{n} \sigma_{d,i,2} t^i & t_1 \leq t < t_2 \\ \vdots & \\ \sum_{i=0}^{n} \sigma_{d,i,m} t^i & t_{m-1} \leq t < t_m \end{cases} \tag{4.1}$$

where $\sigma_{d,i,j}$ is the coefficient of orden i of the trajectory piece j and t_k is the time that the vehicle has to reach setpoint k; $i \in \begin{bmatrix}0 & n\end{bmatrix}$, $j \in \begin{bmatrix}1 & m\end{bmatrix}$, $k \in \begin{bmatrix}1 & m\end{bmatrix}$. The interest is then to minimize a cost function which can be written using these piecewise polynomial.

$$
\begin{aligned}
\min \quad & \int_{t_0}^{t_m} \mu_{\mathbf{x}} \left\| \frac{d^{k_\mathbf{x}} \mathbf{x}_d}{dt^{k_\mathbf{x}}} \right\|^2 + \mu_\psi \left(\frac{d^{k_\psi} \psi_d}{dt^{k_\psi}} \right)^2 dt \\
\text{subject to} \quad & \boldsymbol{\sigma}_d(t_i) = \boldsymbol{\sigma}_{d,i}, & i = 0, \dots, m \\
& \left. \frac{d^p x_d}{dt^p} \right|_{t=t_j} = 0, & j = 0, m; \quad p = 1, \dots, k_r \\
& \left. \frac{d^p y_d}{dt^p} \right|_{t=t_j} = 0, & j = 0, m; \quad p = 1, \dots, k_r \\
& \left. \frac{d^p z_d}{dt^p} \right|_{t=t_j} = 0, & j = 0, m; \quad p = 1, \dots, k_r \\
& \left. \frac{d^p \psi_d}{dt^p} \right|_{t=tj} = 0, & j = 0, m; \quad p = 1, \dots, k_\psi
\end{aligned}
\tag{4.2}
$$

where $\mu_\mathbf{x}$ and μ_ψ are constants that make the integrand nondimensional. Here $\boldsymbol{\sigma}_d = \begin{bmatrix}x_d & y_d & z_d & \psi_d\end{bmatrix}^T$ and $\boldsymbol{\sigma}_{d,i} = \begin{bmatrix}x_{d,i} & y_{d,i} & z_{d,i} & \psi_{d,i}\end{bmatrix}^T$. We also assume that $t_0 = 0$ without loss of generality. The first constraint indicates that the result trajectory has to pass through the desire setpoints, while the rest of the constraints impose that all the derivatives at the initial and final point have to be zero (is also possible to set them to a specific value if necessary). By using the same choice of [17], we decide to minimize the snap for the position ($k_\mathbf{x} = 4$) and the second derivative of the yaw angle ($k_\psi = 2$). Now we want to formulate the trajectory generation problem as an optimization of a functional but in a finite dimensional setting. This will keep the computational effort very small to guarantee a real time application. In order to to this, we first write the constants $\boldsymbol{\sigma}_{d,i,j} = \begin{bmatrix}x_{d,i,j} & y_{d,i,j} & z_{d,i,j} & \psi_{d,i,j}\end{bmatrix}^T$ as a $4 \cdot n \cdot m \times 1$ vector $\mathbf{c}$ with decision variables $\{x_{d,i,j}, y_{d,i,j}, z_{d,i,j}, \psi_{d,i,j}, i \in \begin{bmatrix}0 & n\end{bmatrix}, j \in \begin{bmatrix}0 & m\end{bmatrix}\}$. The trajectory generation preoblem (4.3) can be written in the form of a quadratic program (QP):

$$
\begin{aligned}
\min \quad & \mathbf{c}^T H \mathbf{c} + f^T \mathbf{c} \\
\text{subject to} \quad & A\mathbf{c} \leq \mathbf{b} \\
& A_{eq}\mathbf{c} = \mathbf{b}_{eq}
\end{aligned}
\tag{4.3}
$$

where the objective function will incorporate the minimization of the functional while the constraint can be used to satisfy constraints in the trajectory and it derivatives. A specification of an initial condition, final condition, or intermediate condition on any derivative of the trajectory (e.g. $\frac{d^k x_d}{dt^k}$) can be written as a row of the constraints $A_{eq}\mathbf{c} = \mathbf{b}_{eq}$. If conditions do not need to be specified exactly then they can be represented with the inequality constraint $A\mathbf{c} \leq \mathbf{b}$.
Moreover, to simplify the problem, can be notice that in the cost function of equation (4.3), the four dimensions are indipendent, this means that the problem can be split in four different problems for each dimension. In such a way, the construction of the

quadratic problem vectors and matrices will be considerable more simple. Furthermore, is possible to assume that each setpoint starts from $t_0 = 0$ and ends to $t_j = 1$. This because if we define a new time varible such as

$$\tau = \frac{t - t_{j-1}}{t_j - t_{j-1}} \tag{4.4}$$

the new one dimension position at time τ become

$$x(\tau) = x\Big(\frac{t - t_{j-1}}{t_j - t_{j-1}}\Big) \tag{4.5}$$

and its derivatives

$$\begin{aligned}
\frac{d}{dt}x(t) &= \frac{d}{dt}x(\tau) \\
&= \frac{d}{d\tau}\frac{d\tau}{dt}x(\tau) \\
&= \frac{1}{t_j - t_{j-1}}\frac{d}{d\tau}x(\tau) \\
&... \\
\frac{d^k}{dt^k}x(t) &= \frac{1}{(t_j - t_{j-1})^k}\frac{d^k}{d\tau^k}x(\tau)
\end{aligned}$$

We can thus solve for each piece of any piece-wise trajectory from $\tau = 0$ to 1, then scale to any t_0 to t_j.

4.2 Optimization of a trajectory between two setpoints

To make the derivation simpler, is better to start the optimization problem with only two setpoints and in only one dimension. In particular, the cost function become

$$\begin{aligned}
J &= \int_{t_0}^{t_1} \left|\left|\frac{d^k x(t)}{dt}\right|\right|^2 dt = \mathbf{c}^T H_{(t_0,t_1)}\mathbf{c} \\
\text{subject to} \qquad & A\mathbf{c} = \mathbf{b}
\end{aligned} \tag{4.6}$$

We can instead look for the non-dimensionalized trajectory $x(\tau) = c_n\tau^n + c_{n-1}\tau^{n-1} + \cdots + c_1\tau + c_0$ where $\tau = \frac{t-t_0}{t_1-t_0}$. Note that this makes τ range from 0 to 1. Let $\mathbf{c} = \begin{bmatrix} c_n & c_{n-1} & \dots & c_1 & c_0 \end{bmatrix}^T$.We can wite the cost function J in term of the non-dimensionalized trajectory $x(\tau)$:

$$\begin{aligned}
J &= \int_{t_0}^{t_1} \left|\left| \frac{d^k x(t)}{dt} \right|\right|^2 dt \\
&= \int_{0}^{1} \left|\left| \frac{1}{(t_1-t_0)^k} \frac{d^k x(\tau)}{d\tau} \right|\right|^2 d(\tau(t_1-t_0)+t_0) \\
&= \frac{t_1-t_0}{(t_1-t_0)^{2k}} \int_{0}^{1} \left|\left| \frac{d^k x(\tau)}{d\tau} \right|\right|^2 d\tau \\
&= \frac{1}{(t_1-t_0)^{2k-1}} \mathbf{c}^T H_{(0,1)} \mathbf{c} \\
&= \mathbf{c}^T \left(\frac{1}{(t_1-t_0)^{2k-1}} H_{(0,1)} \right) \mathbf{c}
\end{aligned} \tag{4.7}$$

Thus, we want to minimize the cost function

$$J = \mathbf{c}^T \left(\frac{1}{(t_1-t_0)^{2k-1}} H_{(0,1)} \right) \mathbf{c} \tag{4.8}$$

$$\text{subject to} \qquad A\mathbf{c} = \mathbf{b}$$

To find $H_{(0,1)}$, when $\mathbf{c}' = \begin{bmatrix} c_0 & c_1 & \dots & c_{n-1} & c_n \end{bmatrix}^T$, we can find $H'_{(0,1)}$ with:

$$H'[i,j]_{(t_0,t_1 1)} = \begin{cases} \prod_{z=0}^{k-1} (i-z)(j-z) \frac{t_1^{i+j-2k+1} - t_0^{i+j-2k+1}}{i+j-2k+1} & i \geq k \wedge j \geq k \\ 0 & i < k \vee j < k \end{cases} \tag{4.9}$$

$$i = 0, \dots, n, \qquad j = 0, \dots, n$$

However, $\mathbf{c} = \begin{bmatrix} c_n & c_{n-1} & \dots & c_1 & c_0 \end{bmatrix}^T$. Reflecting H' from equation (4.9) horizontally and vertcally will give the desire H for the form of $\mathbf{c}$ we desire. The function to minimize is then $\left(\frac{1}{(t_1-t_0)^{2k}} H_{(0,1)} \right)$.

To find A

$$A\mathbf{c} = \mathbf{b}$$

$$\begin{bmatrix} A(t_0) \\ A(t_1) \end{bmatrix} \mathbf{c} = \begin{bmatrix} x(t_0) \\ \vdots \\ x^{(k-1)}(t_0) \\ x(t_1) \\ \vdots \\ x^{(k-1)}(t_1) \end{bmatrix}$$

Note that $A\mathbf{c}$ only contains rows where constraints are specified, if a condition is unconstrained just omit a row. Assuming that every condition is constrained, the general form of A is:

$$A[i,j](t) = \begin{cases} \prod_{z=0}^{i-1}(n-z-j)t^{n-j-i} & n-j \geq i \\ 0 & n-j < i \end{cases} \tag{4.10}$$
$$i = 0,\ldots,r-1, \qquad j = 0,\ldots,n$$

where $A[i,j]$ represents the $(n-j)$th coefficient of the ith derivative. In the non-dimensionalized case, we have $\tau_0 = 0$ and $\tau_1 = 1$:

$$\begin{bmatrix} A(\tau_0) \\ A(\tau_1) \end{bmatrix} \mathbf{c} = \begin{bmatrix} x(t_0) \\ \vdots \\ (t_1-t_0)^{k-1}x^{(k-1)}(t_0) \\ x(t_1) \\ \vdots \\ (t_1-t_0)^{k-1}x^{(k-1)}(t_1) \end{bmatrix} \tag{4.11}$$

4.3 Optimization of a trajectory between $m+1$ setpoints

In this section, by recalling what we study in the previous section, is possible to derive the equations to optimize a trajectory for an arbitrary number of setpoints. In particular, we seek the piece-wise trajectory:

$$x(t) = \begin{cases} x_1(t), & t_0 \leq t < t_1 \\ x_2(t), & t_1 \leq t < t_2 \\ \vdots \\ x_m(t), & t_{m-1} \leq t < t_m \end{cases} \tag{4.12}$$

and continue to minimize the cost function

$$J = \int_{t_0}^{t_m} \left\| \frac{d^k x(t)}{dt} \right\|^2 dt = \mathbf{c}^T H_{(t_0,t_m)} \mathbf{c} \tag{4.13}$$
$$\text{subject to} \qquad A\mathbf{c} = \mathbf{b}$$

and again look for the non-dimensionalized trajectory

$$x(\tau) = \begin{cases} x_1(\tau) = c_{1,n}\tau^n + \cdots + c_{1,0}, & t_0 \leq t < t_1, \quad \tau = \frac{t-t_0}{t_1-t_0} \\ x_m(\tau) = c_{m,n}\tau^n + \cdots + c_{m,0}, & t_{m-1} \leq t < t_m, \quad \tau = \frac{t-t_{m-1}}{t_m-t_{m-1}} \end{cases} \tag{4.14}$$
$$0 \leq \tau < 1$$

Let $\mathbf{c}_z = \begin{bmatrix} c_{z,n} & c_{z,n-1} & \ldots & c_{z,1} & c_{z,0} \end{bmatrix}^T$ and $\mathbf{c} = \begin{bmatrix} \mathbf{c}_1^T & \mathbf{c}_2^T & \ldots & \mathbf{c}_m^T \end{bmatrix}^T$. Each piece of the trajectory is optimized individually between $\tau_0 = 0$ and $\tau_1 = 1$. We want to minimize:

$$
\begin{aligned}
J &= \int_{t_0}^{t_m} \left|\left|\frac{d^k x(t)}{dt}\right|\right|^2 dt && (4.15)\\
&= \sum_{z=1}^{m} \int_{t_z-1}^{t_z} \left|\left|\frac{d^k x_z(t)}{dt}\right|\right|^2 dt\\
&= \sum_{z=1}^{m} \int_{0}^{1} \frac{t_z - t_{z-1}}{(t_z - t_{z-1})^{2k}} \left|\left|\frac{d^k x_z(\tau)}{d\tau}\right|\right|^2 d\tau\\
&= \sum_{z=1}^{m} \mathbf{c}_z^T \frac{1}{(t_z - t_{z-1})^{2k-1}} H_{(0,1)} \mathbf{c}_z\\
&= \mathbf{c}^T H \mathbf{c}\\
\text{subject to} \quad & A\mathbf{c} = \mathbf{b}
\end{aligned}
$$

To find H, we recall that for each $\mathbf{c}'_z = \begin{bmatrix} c_{z,0} & c_{z,1} & \dots & c_{z,n-1} & c_{z,n} \end{bmatrix}^T$, where $z = 1, \dots, m$, $H'_{(0,1)}$ is given by equation (4.9). Since $\mathbf{c}_z = \begin{bmatrix} c_{z,n} & c_{z,n-1} & \dots & c_{z,1} & c_{z,0} \end{bmatrix}^T$, refletting H' horizzontally and vertically will give the desire H for the form of $\mathbf{c}_k$. Is then possible to create the block diagonal matrix H

$$
H = \begin{bmatrix}
\frac{1}{(t_1-t_0)^{2k-1}} H_{(0,1)} & \dots & 0 & 0 \\
 & \dots & \dots & \vdots \\
\dots & 0 & \frac{1}{(t_{m-1}-t_{m-2})^{2k-1}} H_{(0,1)} & 0 \\
0 & \dots & 0 & \frac{1}{(t_m-t_{m-1})^{2k-1}} H_{(0,1)}
\end{bmatrix} \tag{4.16}
$$

To find A, first, we need to account for endpoint constraints, in the non-dimensionalized case:

$$
A_{endpoint}\mathbf{c} = \mathbf{b}_{endpoint} \tag{4.17}
$$

$$
\begin{bmatrix}
A(\tau_0) & 0 & \dots & 0 \\
A(\tau_1) & 0 & \dots & 0 \\
0 & A(\tau_0) & \dots & 0 \\
0 & A(\tau_1) & \dots & 0 \\
\vdots & 0 & \dots & \vdots \\
0 & \dots & 0 & A(\tau_0) \\
0 & \dots & 0 & A(\tau_1)
\end{bmatrix} \mathbf{c} =
\begin{bmatrix}
x_1(t_0) \\
\vdots \\
(t_1 - t_0)^{k-1} x_1^{(k-1)}(t_0) \\
x_1(t_1) \\
\vdots \\
(t_1 - t_0)^{k-1} x_1^{(k-1)}(t_1) \\
\vdots \\
x_m(t_{m-1}) \\
\vdots \\
(t_m - t_{m-1})^{k-1} x_m^{(k-1)}(t_{m-1}) \\
x_m(t_m) \\
(t_m - t_{m-1})^{k-1} x_m^{(k-1)}(t_m)
\end{bmatrix}
$$

Like before, we just omit rows where a condition is unconstrained. Also, note that except for constraints at t_0 and t_m, every other constraint must be include twice. The equation for $A[i,j](t)$ is the same of (4.10)
We must also take into account for constraints that ensure that when the trajectory switches from one piece to another at the sepoints, position and all the derivative lower than k remain continuous, for a smooth path. In other words, is require that

$$A_{cont}\mathbf{c} = \mathbf{b}_{cont} \tag{4.18}$$
$$\begin{bmatrix} x_1(t_1) - x_2(t_2) \\ \vdots \\ x_1^{(k-1)}(t_1) - x_2^{(k-1)}(t_1) \\ \vdots \\ x_{m-1}(t_{m-1}) - x_m(t_{m-1}) \\ \vdots \\ x_{m-1}^{(K-1)}(t_{m-1}) - x_m^{(K-1)}(t_{m-1}) \end{bmatrix} = 0$$

Translating to the non-dimeensionalized case, $\tau_0 = 0$, $\tau_1 = 1$, and

$$A_{cont}\mathbf{c} = \mathbf{b}_{cont} \tag{4.19}$$
$$\begin{bmatrix} x_1(\tau_1) - x_2(\tau_2) \\ \vdots \\ \frac{1}{(t_1-t_0)^{k-1}} x_1^{(k-1)}(\tau_1) - \frac{1}{(t_2-t_1)^{k-1}} x_2^{(k-1)}(\tau_1) \\ \vdots \\ x_{m-1}(\tau_1) - x_m(\tau_0) \\ \vdots \\ \frac{1}{(t_{m-2}-t_{m-1})^{k-1}} x_{m-1}^{(K-1)}(\tau_1) - \frac{1}{(t_m-t_{m-1})^{k-1}} x_m^{(K-1)}(\tau_0) \end{bmatrix} = 0$$

and then

$$\begin{bmatrix} A_{cont}(t_1) & 0 & \dots & 0 \\ 0 & A_{cont}(t_2) & \dots & 0 \\ \vdots & 0 & \dots & 0 \\ 0 & \dots & 0 & A_{cont}(t_{m-1}) \end{bmatrix} \mathbf{c} = 0 \tag{4.20}$$

where

$$A_{cont}[i,j](t_z) = \begin{cases} \frac{1}{(t_z-t_{z-1})^i} \prod_{z=0}^{i-1} (n-z-j)\tau_1^{n-j-i}, & n-j \geq i \wedge j \leq n \\ 0, & n-j < i \wedge j \leq n \\ -\frac{1}{(t_{z+1}-t_z)^i} \prod_{z=0}^{i-1} (1-z-j)\tau_0^{1-j-i}, & 1-j \geq i \wedge j > n \\ 0, & 1-j < i \wedge j > n \end{cases} \tag{4.21}$$
$$i = 0, \dots, k-1, \quad j = 0, \dots, 2(n+1)$$

The final constraints $A\mathbf{c} = \mathbf{b}$ take then the final form

$$\begin{aligned} A\mathbf{c} &= \mathbf{b} \\ \begin{bmatrix} A_{endpoint} \\ A_{cont} \end{bmatrix} \mathbf{c} &= \begin{bmatrix} b_{endpoint} \\ 0 \end{bmatrix} \end{aligned} \tag{4.22}$$

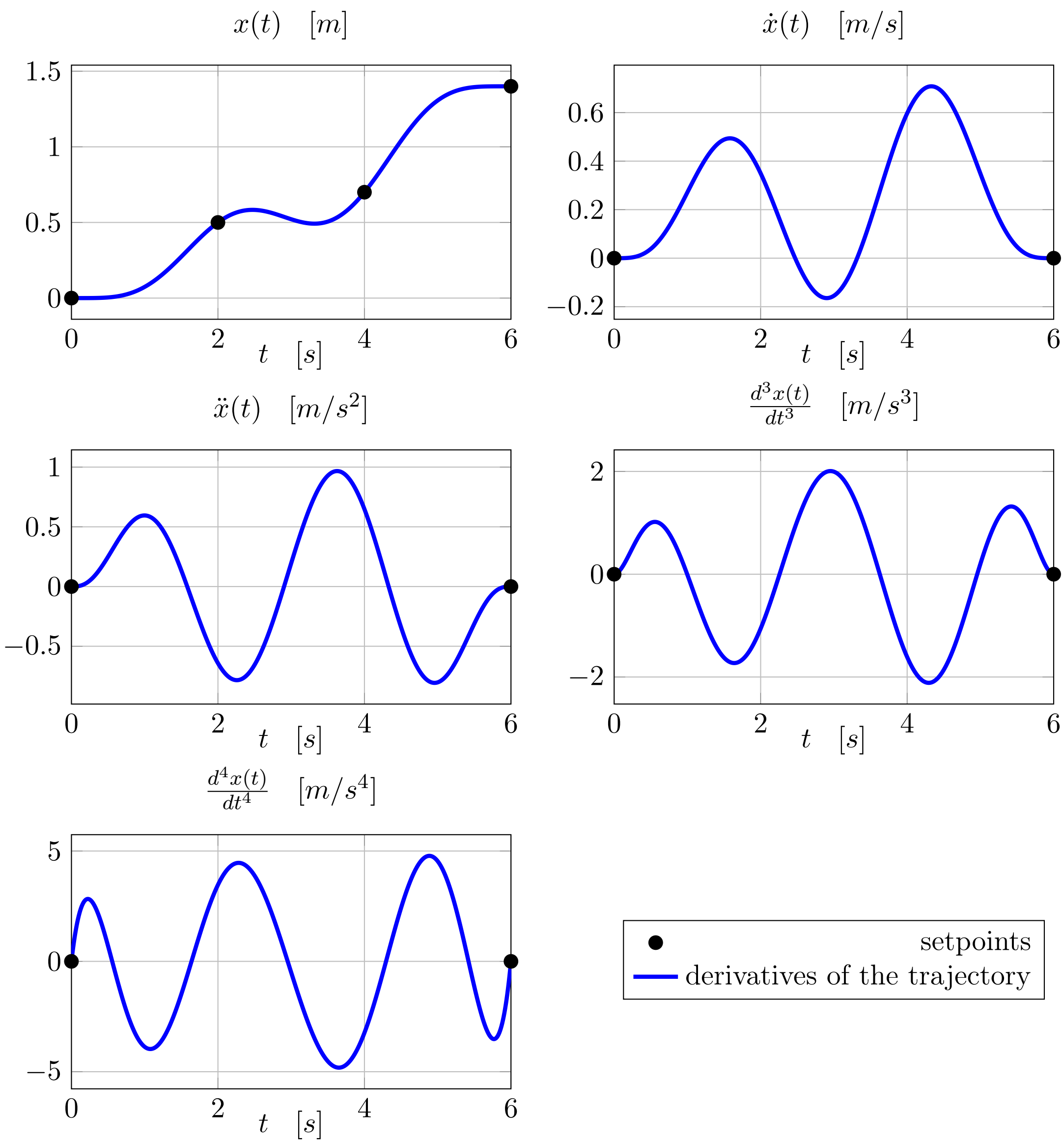

Figure 4.1: Generated trajectory with its derivatives.

In figure 4.1 is reported the first dimension of a generated trajectory evolving over the time. In particular in blue are plotted the trajectory and its first four derivatives, till the snap. Instead in black are plotted the setpoints for the trajectory and the initial and final conditions for each derivative, that are equal to zero. Notice that between setpoints two and three, the trajectory is not as expected, in the sense that first tend to go far from the desired setpoint and than reach it. That's because this is only one dimension of a more complex three dimension trajectory.

4.4 Adding corridor constraints

In this section, corridors constraints will be added in the cost function (4.3). For corridor constraints, we mean that the desire trajectory must be inside a corridor, this why for a safe obstacle avoidance and navigation algorithm, the vehicle must respect distance between walls and obstacles. To do this, we first define $\mathbf{u}_i$ as the unit vector along the segment from setpoint $\mathbf{r}_i$ and setpoint $\mathbf{r}_{i+1}$.

$$\mathbf{u}_i = \frac{\mathbf{r}_{i+1} - \mathbf{r}}{||\mathbf{r}_{i+1} - \mathbf{r}||} \tag{4.23}$$

The perpendicular distance vector, $\mathbf{d}_i(t)$, from segment i is defined as

$$\mathbf{d}_i(t) = (\mathbf{r}_d(t) - \mathbf{r}_i) - ((\mathbf{r}_d(t) - \mathbf{r}_i) \cdot \mathbf{t}_i)\mathbf{u}_i \tag{4.24}$$

where $\mathbf{r}_d(t)$ is the desire trajectory at istant t. A corridor width on the infinity norm, δ_i, is defined for each corridor as follow

$$\left|\left|\mathbf{d}_i(t)\right|\right|_\infty \leq \delta_i \quad \text{while} \quad t_i \leq t \leq t_{i+1} \tag{4.25}$$

The reason to write the constraint like that, is because it can be incorporate into the QP problem by introducing n_c intermediate points as

$$\left| x_a \cdot \mathbf{d}_i\left(t_i + \frac{j}{1+n_c}(t_{i+1} - t_i)\right)\right| \leq \delta_i \quad \text{for} \quad j = 1, \dots, n_c \tag{4.26}$$

where for x_a we mean that this procedure must be compute for x_W, y_W and z_W, with $\mathbf{x}_W = \begin{bmatrix} x_W & y_W & z_W \end{bmatrix}^T$. Of course a corridor constraint in the desire yaw doesn't have sense. To do this, we introduced inequality constraints of the form $A_{ineq}\mathbf{c} \leq \mathbf{b}_{ineq}$.
To find A_{ineq}, we first break down the inequality (4.26) into

$$(x_a \cdot \mathbf{d}_i(t_i + \frac{j}{1+n_c}(t_{i+1} - t_i))) \leq \delta_i \tag{4.27}$$

$$-(x_a \cdot \mathbf{d}_i(t_i + \frac{j}{1+n_c}(t_{i+1} - t_i))) \leq \delta_i \tag{4.28}$$

This result in a total of $2 \cdot 2 \cdot n_c$ constraints for each corridor constraint. Then by performing some math, the matrix A_{ineq} and the vector b_{ineq} can be deduce [1].
In figure 4.2 are reported examples of trajectories, with and without corridor constraints. For better understanding are reposrt 2D trajectories, but the same example could be made also for 3D trajectories. The corridor constraint is present only between two setpoints, setpoint 2 and setpoint 3. As is possible to see, the trajectory remain in beetween the constraint, but become less smooth in compare to the one without constraints. Moreover, notice that the entire trajectory is different and not only the segment in between the corridor, this is another important advantage of using this technique to generate trajectores.

[1]The formulation of such matrix is very eavy and difficoult to understand, that's why is not report. Or maybe put in a appendix

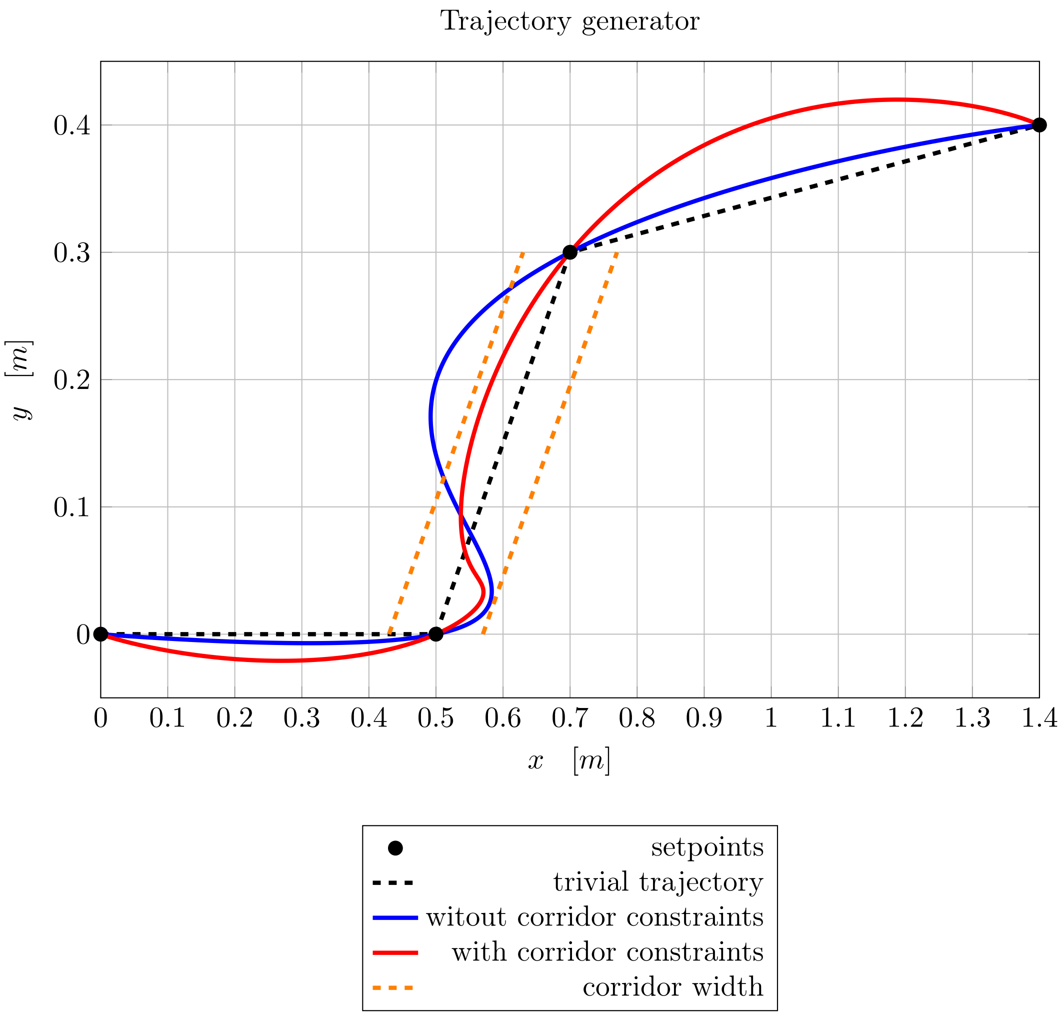

Figure 4.2: Setpoints and desire trajectory, with and without corridor constraints between setpoints 2 and 3.

5 Control

The last important step of this work, is to derive a control law, able to track the desire trajectory and to compensate for the movement of the sensor.

5.1 Force and torque position controller

In this section, will be introduce a Tracking Control for the vehicle, based on the work [18]. We will use this control because it has been show to work well in many different applications, from precision flights, to fast and aggressiv flights. In particular, starting from the model of the UAV, we will derive a position tracking control, based on $SO(3)$ group.
The *general linear group* of order 3, $GL(3)$, is a algebraic group composed by all the non singular matrices $A \in \mathbb{R}^{3\times 3}$ with matrix product.
The *orthogonal group* of order 3, $O(3)$, is deefine as

$$O(3) = \{A \in GL(3) : A^T A = I\} \subseteq GL(3)$$

Is easy to prove that if A belongs to $O(3)$, then $\det[A] = \pm 1$.
The *special orthogonal grouop* of order 3, $SO(3)$, is define as

$$SO(3) = \{A \in O(3) : \det[A] = 1\}$$

Belong to this gruop the rotation matrices, and is because the controller will use errors based iìon matrices, we need to keep all the results in the group $SO(3)$.
To derive the control law, we first need to define the tracking errors. In particular the position and the velocity tracking errors are given by, respectively

$$\mathbf{e}_x = \mathbf{x} - \mathbf{x}_d \tag{5.1}$$
$$\mathbf{e}_v = \dot{\mathbf{x}} - \dot{\mathbf{x}}_d \tag{5.2}$$

where the subscript d stands for desire. The attitude error is insted define as

$$\mathbf{e}_R = \frac{1}{2}(R_c^T R - R^T R_C)^\vee \tag{5.3}$$

where R is the rotation matrix that encode the actual attitude of the UAV, while $R_c(t) \in SO(3)$ is the computed attitude matrix, that must belongs to the special ortoghonal group. In fact we can define it as

$$\mathbf{b}_{1,c} = \begin{bmatrix}\cos(\psi_d) & \sin(\psi_d) & 0\end{bmatrix}^T \tag{5.4}$$

$$\mathbf{b}_{3,c} = -\frac{k_x\mathbf{e}_x + k_v\mathbf{e}_v - g\mathbf{e}_3 - \ddot{\mathbf{x}}_d}{||k_x\mathbf{e}_x + k_v\mathbf{e}_v - g\mathbf{e}_3 - \ddot{\mathbf{x}}_d||} \tag{5.5}$$

$$\mathbf{b}_{2,c} = \frac{\mathbf{b}_{3,c} \times \mathbf{b}_{1,c}}{||\mathbf{b}_{3,c} \times \mathbf{b}_{1,c}||} \tag{5.6}$$

$$R_c = \left[\ \mathbf{b}_{2,c} \times \mathbf{b}_{3,c} \ \middle|\ \mathbf{b}_{2,c} \ \middle|\ \mathbf{b}_{3,c}\ \right] \tag{5.7}$$

Where ψ_d is the desire yaw, $\mathbf{e}_3$ is the third dimension canonical vector, k_x and k_v are positive control constants. The *vee* map $\cdot^\vee$ is the inverse of the *hat* map $\hat{\cdot} : \mathbb{R}^3 \rightarrow \mathrm{SO}(3)$ define as

$$\begin{aligned}\mathbf{v} &= \begin{bmatrix}v_1 & v_2 & v_3\end{bmatrix}^T \\ \hat{\mathbf{v}} &= \begin{bmatrix}0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0\end{bmatrix}\end{aligned} \tag{5.8}$$

The angular velocity error e_ω depends only from the desire yaw, since our trajectory generator compute the desire yaw till the second derivative. However, it can be compute simple by

$$\mathbf{e}_\omega = \omega - R^T R_c \omega_c \tag{5.9}$$

where $\hat{\omega}_c = R_c^T \dot{R}_c$.
The final control law is then divide in two contributions, one for the total force and one for the torque

$$f = -(k_x\mathbf{e}_x + k_v\mathbf{e}_v - g\mathbf{e}_3 - \ddot{\mathbf{x}}_d)^T R\mathbf{e}_3 \tag{5.10}$$

$$\boldsymbol{\tau} = -k_R\mathbf{e}_R - k_\omega e_\omega + \omega \times I_{cm}\omega \tag{5.11}$$

where k_R and k_ω are again positive control gains. As is possible to see, this controller is very simple to implement and can be prove that the control is exponentialy stable, if the initial conditions are sufficently small[1]. Of course the control law is not complete, because what we can control are the inputs to the motors and not the force and torques. So for compute the linear acceleration in the body frame we just need to do

$$\ddot{\mathbf{x}}_B = \frac{1}{m} \cdot \begin{bmatrix}0 & 0 & f\end{bmatrix}^T \tag{5.12}$$

[1] see the paper [18] for more details and the prooof

For the angular acceleration in the body frame, in theory we just need to compute

$$\dot{\boldsymbol{\omega}}_B = I_{cm}^{-1}\boldsymbol{\tau} \tag{5.13}$$

Then, by using the results from section 2.2.2, we can compute the speed of the propeller with

$$\begin{bmatrix} \vdots \\ u_i^2 \\ \vdots \end{bmatrix} = \underbrace{\begin{bmatrix} \dots & \frac{A_{F,i}\Omega_{max,i}^2 \mathbf{n}_i}{m} & \dots \\ \dots & I_{cm}^{-1}\left[\mathbf{l}_i \times A_{F,i}\Omega_{max,i}^2\mathbf{n}_i - \text{sgn}(\Omega_i)B_{F,i}\Omega_{max,i}^2\mathbf{n}_i\right] & \dots \end{bmatrix}^{-1}}_{T^{-1}(\boldsymbol{\beta})} \cdot \begin{bmatrix} f \\ \boldsymbol{\tau} \end{bmatrix} \tag{5.14}$$

where the matrix $T^{-1}(\boldsymbol{\beta})$ is the invers of the estimated parameters. However the problem is in the computation of the inverse of the inertia matrix, since the estimated parameters are coupled with the component of the inertia and is not possible to etimate directly this. A naive approach is simply use the inertia computed with the CAD model. Of course, this approach will add errors, since the CAD doesn't provide a perfect data. HOvewer, this inverse will be multiply with k_R and k_ω and then just a simple retuning of the parameters will be necessary. Instead, the term $I_{cm}^{-1}(\boldsymbol{\omega} \times I_{cm}\boldsymbol{\omega})$ as said in section 3.1 is very small and will not introduce significant errors.

5.1.1 Adding the rotating platform

The previous controller was derive without the roating platform. To introduce the compensation for the movement of the cart, first of all we introduce the compensation for the moving COG. To do this we simply modify the terms $\mathbf{l}_i$ in the matrix $T(\boldsymbol{\beta})$ and then compute the inverse at every iteration. Of course, to do this, we need to know precisely the position γ of the cart, this can be done since the motor that drive the cart is provid of encoder [3]. The changing in the inertia matrix are instead compute with the CAD, a more precisely solution could be to compute the system identification with the moving cart or, if if doesn't work, compute the system identification multiple times with different positions of the sensors and then interpretate the system as a piecewise system.

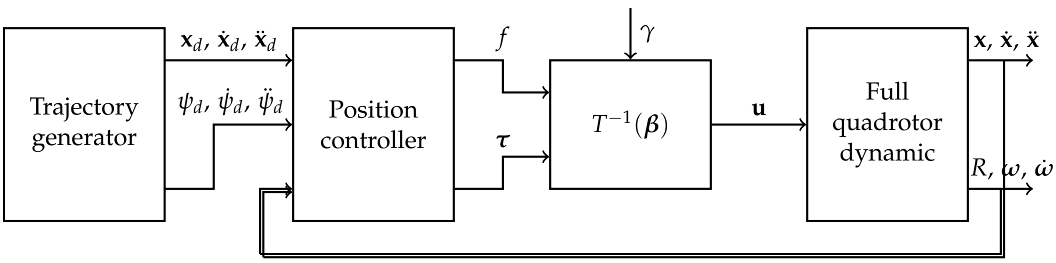

Figure 5.1: Block diagram of the control scheme (the subscripts that indicate the appartenence of the frame are omitted).

5.1.2 Simulation Results

6
Conclusions and future works

write te conclusions

Bibliography

[1] **P. Pounds, R. Mahony, and P.Corke**. Modelling and control of a large quadrotor robot. *Control EngineeringPractice18, 2010*. 1

[2] **C. Mary, L. C. Totu, and S. Konge Koldbæk**. Modelling and Control of Autonomous Quad-Rotor. *Aalborg Universitet, June 2010*. 1

[3] **C. Navarro Leoncio**. Design of the Prometheus UAV. *Master thesis project, Luleå university of technology, 2016*. 2, 11, 31

[4] **J. Solà**. Quaternion kinematics for the error-state KF. *February 2, 2016*. 5

[5] **E. Fresk, and G. Nikolakopoulos**. Full Quaternion Based Attitude Control for a Quadrotor. *2013 European Control Conference (ECC), July*. 5

[6] **J. Diebel**. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. *Stanford University, 2006*. 6

[7] **T. Bresciani**. Modelling, Identification and Control of a Quadrotor Helicopter. *Department of Automatic Control, Lund University, October 2008*. 8

[8] **E. Fresk, and G. Nikolakopoulos**. Experimental Model Derivation and Control of a Variable Pitch Propeller Quadrotor. *IEEE Multi-Conference on System and Control, 2014*. 9

[9] **E. Fresk, R. J. Cotton, and G. Nikolakopoulos**. Generalized Model Identification for Multirotors: Towards applications in Auto Tunning. *2016*. 9, 13, 14

[10] **F. P. Beer, E. R. Johnston, D. F. Mazurek, P. J. Cornwell, and B. P. Self**. Vector Mechanics for Engineers, Statics and Dynamics. *Tenth edition*. 11

[11] **N. Abas, A. Legowo, and R. Akmeliawati**. Parameter Identification of an Autonomous Quadrotor. *2011 4th International Conference on Mechatronics, 17-19 May 2011, Kuala Lumpur, Malaysia*. 15

[12] **G. Welch, and G. Bishop**. An Introduction to the Kalman Filter. *University of North Carolina at Chapel Hill, Department of Computer Science*. 15, 16

[13] **J. Barraquand, B. Langlois, and J.-C. Latombe**. Numerical Potential Field Techniques for Robot Path Planning. *Stanford University, California, 1989*. 19

[14] **M. Bl¨osch, S. Weiss, D. Scaramuzza, and R. Siegwart**. Vision Based MAV Navigation in Unknown and Unstructured Environments. *Autonomous Systems Lab, ETH Zurich*. 19

[15] **D. Mellinger, and V. Kumar**. Minimum Snap Trajectory Geeration and Control for Quadrotors. *IEEE International Conference on Robotics and Automation, Shanghai, China, 2011*. 19

[16] **C. Richter, A. Bry, and N, Roy**. Polynomial Trajectory Planning for Quadrotor Flight. *Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge*. 19

[17] **D. W. Mellinger**. Trajectory Generation and Control for Quadrotors. *University of Pennsylvania, 2012*. 20

[18] **T. Lee, M. Leok, and N. H. McClamroch**. Nonlinear Robust Tracking Control of a Quadrotor UAV on $SE(3)$. *2012 American Control Conference, Fairmont Queen Elizabeth, Montréal, Canada, June 27 - June 29, 2012*. 29, 30