Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea in Ingegneria dell'Automazione

Tesi di Laurea Magistrale

Modeling, identification and navigation of autonomous air vehicles

Candidato
Matteo Vanin

Relatore
Ch.mo Prof. Giorgio Picci

Anno Accademico 2012-2013

Dedicato ad Alba e Luciano

Abstract

During the last few years Unmanned Air Vehicles have seen a widespread utilization, both in civilian and military scenarios, because of the benefits of replacing the human presence in unsuitable or hostile environments and dangerous or dull tasks. Examples of their use are surveillance, firefighting, rescuing, extreme photography and environmental monitoring.
The main interest of this work is autonomous navigation of such air vehicles, specifically quadrotor helicopters (quadrocopters), and the focus is on convergence to a target destination with collision avoidance. In this work, a general model for a quadrocopter UAV is obtained, making use of a first principles modeling approach, and system identification is exploited in order to relate in a suitable manner the control signals to the effective behavior of the vehicle. The main contribution is the design of a controller for convergence and avoidance of static obstacles, based both on considerations on the dynamics of the agent and knowledge of the testbed for the experiments.
The controller is composed of a layered structure. The external layer consists in the computation of a collision-free path leading to the target position and is based on a navigation function approach. The inner layer is meant to make the vehicle follow the waypoints imposed by the outer layer and thus consists in a position controller. Experiments have been conducted in different scenarios in order to analyze the behavior of the controlled system.
The final part of the work regards the design of a controller for 3D navigation and collision avoidance for an air vehicle with more constrained dynamics in respect to the quadrocopter. This controller exploits both dipolar navigation functions and model predictive control in order to obtain the control inputs that safely lead the vehicle to its destination with the desired orientation.

Acknowledgements

I express my gratitude to Dr. Dimos Dimarogonas for supervising me in this project and being a constant source for assistance, suggestions and useful critics. I would like to thank Prof. Giorgio Picci and Prof. Bo Wahlberg for giving me the opportunity to do my thesis in the Automatic Control Lab of KTH.
I also thank Michele Colledanchise and Mani Amoozadeh for their help on theoretical aspects, code and hardware and Martina Zambelli for her excellent opposition.

I wish to extend my gratitude to the people who shared with me these years at university. Thanks to Christian Piccolo, Nicola Belli, Federico Bolner, Paolo Anzelini and Giacomo Borz for the laughter, wise words, food and playtime we shared. You made our house such a dangerous place to live in.
Thanks to Anna Vanzan, for her beautiful mind and for everything we shared in the past years: you will always remain a part of me.
I am grateful to all of my friends. Thanks to the ones I grew up with: you are an everlasting and fundamental part of my life. Thanks to the ones I met in Stockholm: you made this experience unforgettable and opened my mind as never before.

Last, but most important, I wish to thank my family for always supporting me and being the protection behind my back and the spring to walk forth. Thank you mamma, papà, Gianmarco, Silvana, Franco.

Contents

Chapter 1

Introduction

The interest of mankind for aerial vehicles in general has always been flourishing, and in the twentieth century the aircraft industry has seen its birth and boost due to military employment first and civilian transport then.

The concept of Unmanned Air Vehicles (UAVs), or drones, was born quite early in the twentieth century: as far back as in 1915 Nikola Tesla described an armed, pilotless aircraft designed to defend the United States [1]. The development of UAV technology has regarded largely military purposes, but lately a lot of civilian applications have become established as well. The main applications are intelligence gathering, surveillance, platooning, as well as climate and pollution monitoring, rescuing operations, pipeline inspection [2]. The success of this technology relies in replacing the human presence when navigation and maneuver in adverse or uneasily reachable environments is needed.

The control of these vehicles can be either manual, via remote transmitters, or automatic, thanks to computing units, that in turn can be onboard or remotely connected to the agent. UAV units are usually equipped with various kinds of sensors that aim to provide information on the environment or are exploited for the control of the vehicle itself. We can number among these: GPS antennas, vision, infrared (IR), thermal, proximity sensors, inertia measurement units (IMU), accelerometers, magnetometers.

Our focus regards vertical take off and landing (VTOL) UAVs: their clear advantages are the reduced maneuvering space required for takeoff and landing operations and the possibility to hover. Multicopters, or multirotors, are VTOL UAVs propelled by more than two rotors, and depending on the number of propellers (four, six or eight) they are named *quadrocopters*, *hexacopters* or

octocopters respectively. These kinds of vehicles are increasingly being exploited for visual documentation of sites and buildings because of their low-budget availability.

In our work we will analyze and exploit a quadrocopter (or quadrotor). Therefore, let us review the literature on control methodologies and goals for such type of vehicles.

1.1 Literature survey

Literature on quadrotor control is ample, because of the above-mentioned easy availability of multicopters, the possibility for both indoor and outdoor flight and the simpler structure in respect to the other multicopters.

Some of the most common applications for the control of quadrotors are

- Stabilization
- Path following
- Obstacle avoidance
- Cooperative control (see [3], [4])
- Acrobatic and aggressive maneuver (see [5], [6])

Various control methodologies are associated to these control goals, let us briefly describe them in association with their applications.

Stabilization. PID controllers are often used in onboard controllers to stabilize the angles of the quadrocopter. The generic (theoretical) structure of a PID controller in Laplace's domain is, being K_P, K_I and K_D nonnegative constants,

$$C(s) = K_P + \frac{K_I}{s} + K_D s.$$

This transfer function expresses the relation between the control input and the process error.

Path following. Techniques as backstepping control [7], [8], [9], sliding mode control [10], [11] associated with input-output feedback linearization [12] are exploited for the task of path following; these methods all mean to control

nonlinear dynamics. Backstepping is used to stabilize nested dynamical systems: when a subsystem (e.g. position control and yaw control) is assumed to be stabilized using some other method, backstepping provides a procedure to progressively stabilize the whole system. Generally speaking, sliding mode method consists in driving the trajectory of the nonlinear state of the system into a pre-designed attractive surface of the state space (*switching surface*) and to maintain the state on this surface by a control that switches its gain according on the state trajectory being above or below the switching surface.

Obstacle avoidance The problem of obstacle avoidance presents basically two approaches: the first one relies on vision and distance sensors onboard the vehicle, the second one is based on a *a priori* knowledge of the environment the agent moves in. The former case presents methodologies based on visual recognition of the obstacles (see e.g. [13]), while the latter exploits techniques as

- C-obstacles: in [3] the region at disposal of the vehicle is restricted by the subtraction of the subset that causes at least one collision from the total area and then dynamic programming is exploited for the motion;
- LQG obstacles [14]: LQG-Obstacle is defined as the set of control objectives that result in a collision with high probability. Selecting a control objective outside the LQG-Obstacle then produces collision-free motion;
- Potential functions [15]: potential and navigation functions are designed to be attractive towards the destination of the vehicle and repulsive from the obstacles; these functions are integrated in the control to provide a collision-free path to the agent.

1.2 Approach and motivations

The aim of the controller we design is providing collision-free navigation towards a preset target. Fist we will present a heuristic controller: its structure and design come from considerations about the actual testbed in which we operate, in particular:

- The quadrocopter comes with an onboard out-of-the-box stabilization system;

- The structure of the quadrocopter is such that lateral movement is easily achieved in every direction;
- We can rely on a very accurate motion capture system providing 6 degrees of freedom (6DoF) data for the agent;
- The workspace is indoor so we need to take care of safety issues coming from abrupt or too fast maneuvers

The control task will be pursued splitting it into collision-free path generation and path following. In particular, we suppose to know the position of the obstacles[1] and we will obtain a collision-free path for a kinematic agent, meaning by this that we do not consider any dynamical constraints on the motion of the agent itself. Because of this procedure, we cannot grant *a priori* that the path we generate can be tracked exactly by the actual quadrotor, that presents a highly nonlinear dynamic model. Nevertheless, the fact that our measurement system is extremely precise and the possibility to exploit the stabilization system of the quadrotor suggest that a point-to-point controller could be satisfactory implemented and if the desired path is sampled in a suitable manner and safety margins are considered, the control can track a collision-free path to the destination. The experiments performed with the above-described heuristic controller aim to test the collision avoidance system by reproducing in reduced scale possible scenarios in which an unmanned air vehicle can be exploited, such as inspection and patrolling of environment or reaching of an adequate position to perform tasks of surveillance, rescue or visual documentation.

In order to extend the possibility of navigation in our framework it is suitable to make the theory for a navigation controller as general as possible, so that differently constrained air vehicles can be used in the testbed. For this purpose, we will introduce a controller for a single integrator air vehicle that has a more constrained dynamic than the one of the quadrotor. In this case the controller is also based on a navigation function approach, but the analysis will not just provide a suitable path, but also the inputs to feed the model in order to obtain convergence and collision avoidance: these properties will be demonstrated exploiting Lyapunov's theory. Moreover, the controller will also take into account the orientation of the vehicle, since for the sake of generality we do not consider it

[1] or be able to retrieve this information thanks to the motion capture system

symmetrical, and a generic performance measure, that can be adapted according to the cases.

1.3 Outline

We will now present shortly the content of each of the next chapters.

Chapter 2. The hardware and software of the testbed is presented. After a qualitative introduction on the principles of working of a quadrocopter, the quadrocopter platform, the motion capture system and the communication link are described in detail.

Chapter 3. First, a model of the dynamics of the quadrocopter is obtained exploiting a first principles approach. Then, system identification is performed in order to obtain suitable transfer functions that relate the user inputs to the effective behavior of the agent.

Chapter 4. The heuristic controller for obstacle avoidance is presented, as well as the numerical and visual results of the experiments performed to test its behavior.

Chapter 5. The controller for a more constrained vehicle is presented and after an introduction to dipolar navigation functions and model predictive control, its properties of convergence and collision avoidance are demonstrated.

Chapter 6. The results of this work are summed up and perspectives for future development are presented.

Chapter 2

Hardware and software

2.1 Overview on quadrotors and test-bed

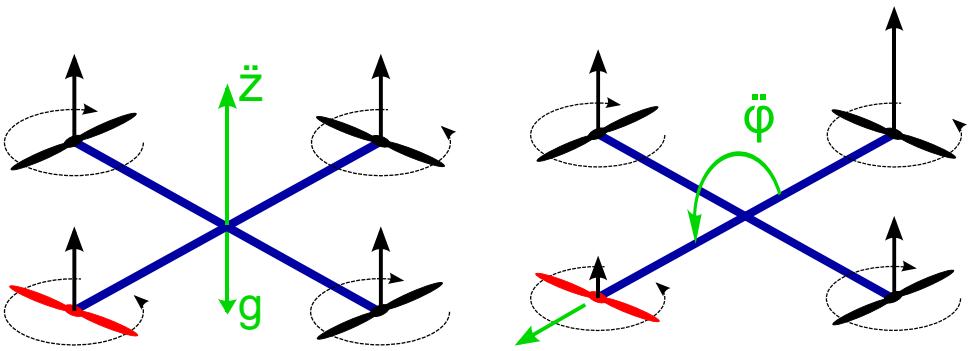
In this work we will exploit Jdrones ArduCopter quadrotors (see figure 2.1). A detailed description of the hardware can be found in section 2.3.

Figure 2.1 – ArduCopter quadrotor (`http://www.diydrones.com/profiles/blogs/updates-on-arducopter-3d-model`)

The experiments have been performed in the Smart Mobility Lab of KTH - Royal Institute of Technology, Stockholm. Inside the laboratory a system of cameras is used to track the movement of the quadrocopter and can be though as an indoor, small-scale reproduction of a GPS system for outdoor tracking. This is a fair assumption dealing with position tracking, but the motion capture system provides also information about the orientation of the agent, and in an outdoor testbed this feature could be replaced by the use of other navigational instruments e.g. magnetometers.

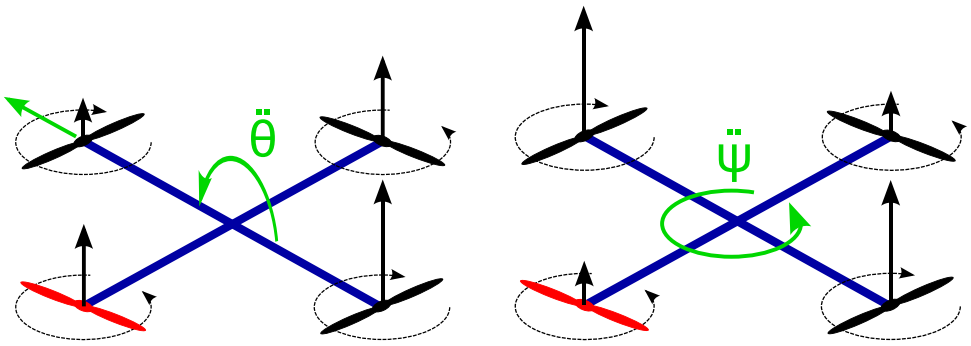
2.2 Principles of working

Here we present in an intuitive manner the basics that rule the flight of a quadrotor; these are basic concepts that do not require any further knowledge on the hardware or on the physical model, but give an appropriate understanding on how the vehicle works. We refer to figure 2.2 for a visual representation.



(a) Upward movement

(b) Pitch and lateral movement



(c) Roll and lateral movement

(d) Yaw and rotational movement

Figure 2.2 – Basic movements of a quadrotor. The front propeller is marked in red and the symbols z, ϕ, θ, ψ refer respectively to the vertical position and the pitch, roll and yaw angles.

The four onboard motors can be controlled separately; each motor produces a torque that makes the rotors spin producing an upward thrust. Just like in single-rotor helicopters, the torque created by the motors would make the body of the quadrotor spin in the opposite direction of the rotors: to avoid this phenomenon two opposite rotors spin clockwise and the other two counterclockwise, so that the effect is balanced.

The upward movement is achieved when all the rotors spin at the same rate: in this case the total thrust vector is perpendicular to the ground and if its

absolute value equals the gravity force acting on the quadrotor, the vehicle will hover at a fixed height.

The lateral movement of the quadrotor is performed by decreasing the rate of one rotor and increasing by the same amount the rate of the opposite one. In this scenario the quadrotor will tilt towards the direction of the slow propeller and as long as the total thrust do not variate, a lateral movement is achieved. This tilting movement is referred as *pitch* if the rotor that slows down is the front or the rear one, or *roll* if the rotor that slows down is a lateral one. However, since the vehicle is symmetrical with respect to its center of gravity, the choice of the orientation is non-influential to the dynamics.

Finally, the rotational movement[1], or yaw movement, can be obtained by slowing down the spinning rate of two opposite propellers and increasing the spinning rate of the others by the same amount (so that the total thrust vector is not influenced). In this way, the quadrotor will rotate in the same direction as the slow propellers, in fact as we have already stated, the torque of the motors produces a rotation of the body in the opposite direction according to the third law of motion.

[1] around the vertical axis passing through the center of gravity of the craft.

2.3 Introduction to the resources

In this section we present the test-bed and the hardware and software used for this project. The main components of the test-bed are the quadrotor (agent), the Qualysis camera system, a PC running the motion capture software and a PC running NI Labview and Matlab to control the quadrotor.

As a preliminary overview, we summarize in the next lines and in figure 2.3 the representation of the testbed and how its various parts are put in communication and form a closed loop.

In order to get the current position and orientation of the quadrotor, we exploit the Qualisys motion capture system. It consists of a set of twelve Oqus infrared cameras: after a calibration process that fixes the reference frame, they are used to capture the position of small retro-reflective markers that are placed on the quadrotor. The information coming from the cameras is sent via ethernet to a desktop computer running a proprietary software that computes the 3D position of the markers and provides the user 6 degrees of freedom data. A library (*QLC.lvlib*) is provided to communicate this data to LabView.

The control of the system is performed using LabView and exploiting Matlab scripts into LabView code. The reference signals coming from LabView (i.e. one byte for each of desired throttle, pitch, roll, yaw and flight mode) are sent via wifi to an actuator TMote Sky module onboard the quadrotor. Then these signals are transferred via serial to a serial adapter board, that in turn forwards the logic-level serial signals to the Pololu servo controller[2], that converts them in PWM signals. They, in turn, feed the Arduino CPU board containing two cascade PID loops that stabilize the quadrotor with the desired references. The outputs of the controller are finally delivered to the power distribution that forwards those signals to the four speed controllers that set the input voltages of each brushless AC motor individually.

In the following we will describe more accurately the hardware of the quadrocopter, the motion capture system, the wireless communication link and the operations that must be done to get the quadrocopter ready to fly.

[2] Detailed information about the serial board and the Pololu board can be found in the Smart Mobility Lab manual [16]

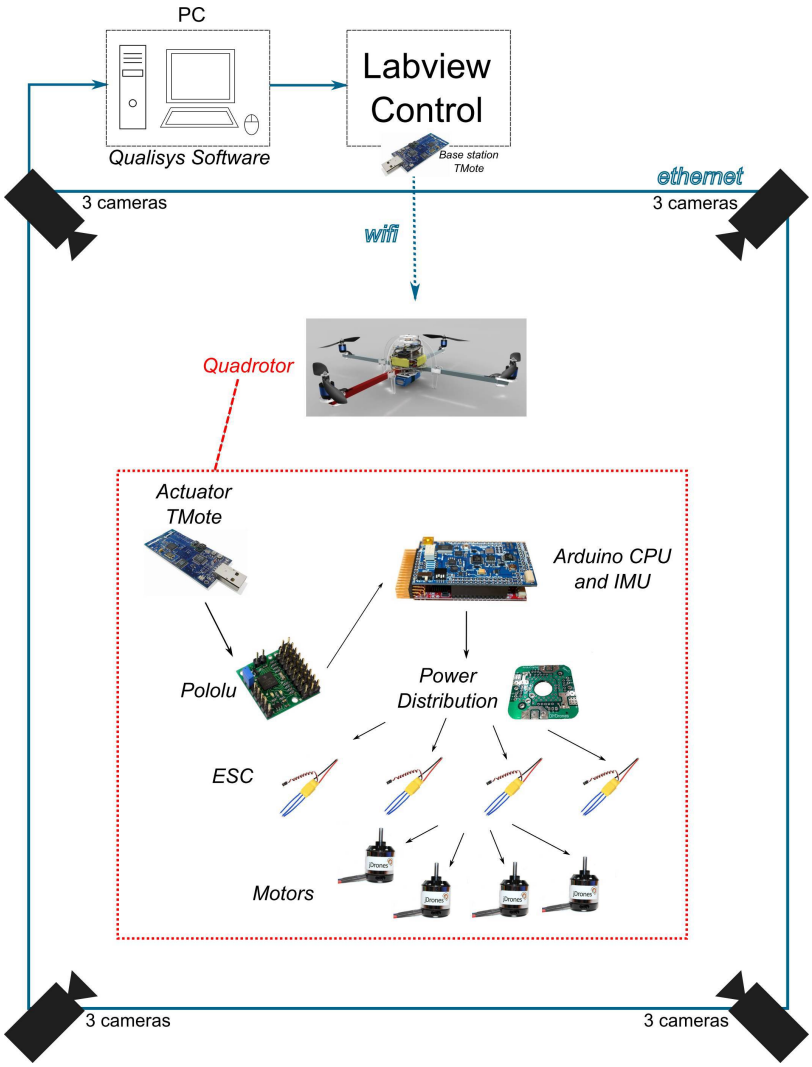


Figure 2.3 – Testbed

2.4 Jdrones ArduCopter

Arducopter is a platform for multicopters and helicopters that provides both manual remote control and an autonomous flight system equipped with mission planning, waypoints navigation and telemetry, managed via software from a ground station.

The quadrocopter has been assembled from the ArduPilot Mega kit, containing the Arduino controller board and the IMU board, and the JDrones ArduCopter kit, containing the frame, the motors, the speed controllers, the power distribution board and the propellers. Detailed instruction on how to solder and assemble the parts of the kits are provided in the Wiki section of [17], we here present briefly how these components are connected and their role in the functioning of the quadrocopter.

2.4.1 ArduPilot Mega

The core processing unit of the quadcopter consists in the ArduPilot Mega 1 board (figure 2.5). The firmware of this board is uploaded via usb using the *APM planner* software, downloadable from the Download section of [17].

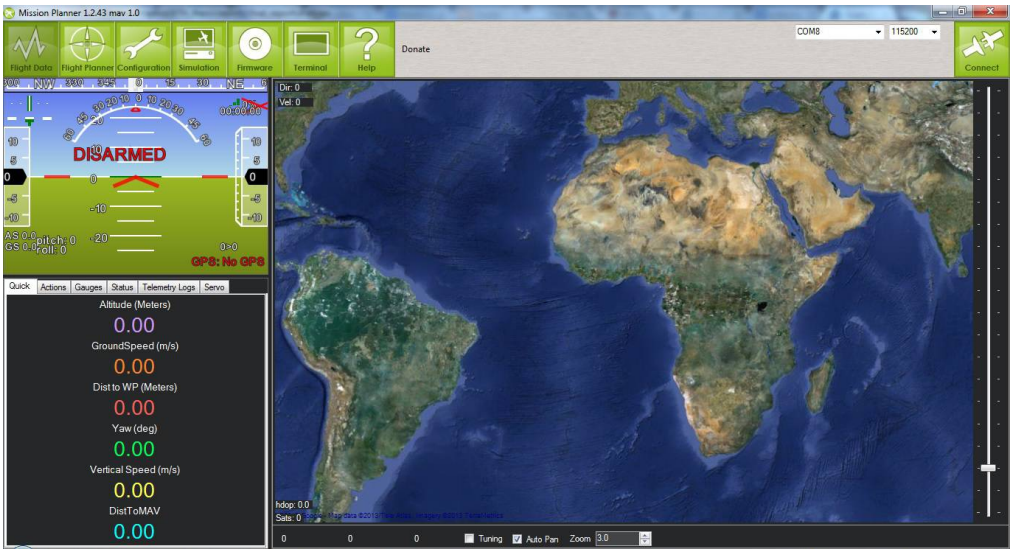
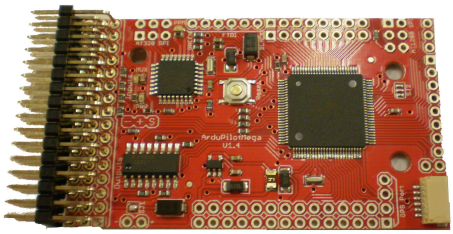


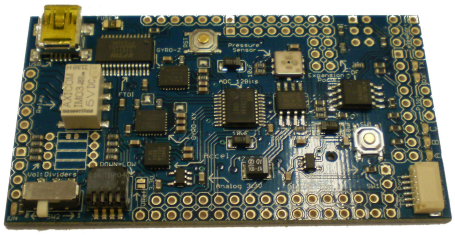
Figure 2.4 – Screenshot of APM planner software.

Two boards compose the ArduPilot Mega: the control board (red) and the IMU board (blue). In our testbed the control board is responsible for taking as

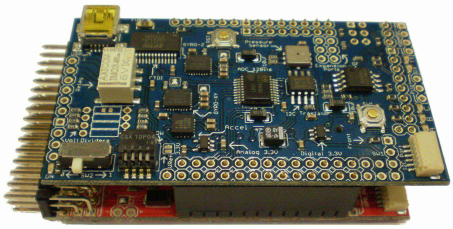
input five remote signals in form of PWM waves (these signals correspond to throttle, pitch, roll, yaw and flight mode[3]) provided by the user and use them as references of the double cascade PID onboard controller. The control board is connected to the IMU board, that is equipped with a large number of sensors providing readings that are exploited by the controller board or can be used for other applications; in our setup the exploited sensors are an accelerometer and a gyroscope, other sensors are a GPS module (not connected), a magnetometer (soldered but not used), pressure and temperature sensors.



(a) *Control board.*



(b) *IMU board.*



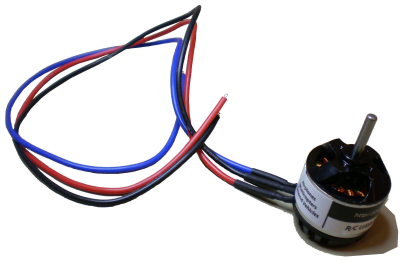
(c) *ArduPilot Mega soldered.*

Figure 2.5 – ArduPilot Mega

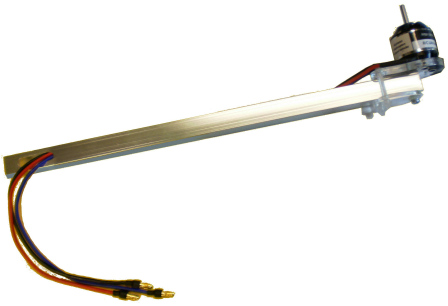
[3] In our implementation we do not actually exploit the *flight mode* signal, so the inputs of interest are actually four.

2.4.2 Arducopter frame, power distribution and motors

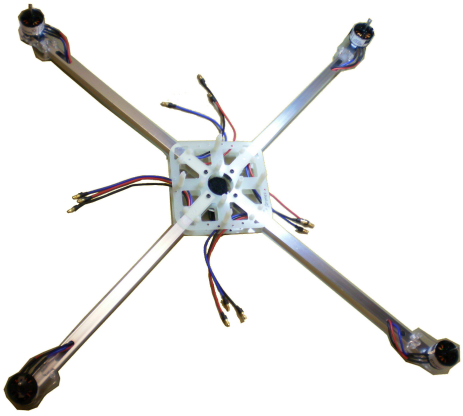
The frame of the quadrocopter is basically made of plastic pieces composing the structure and the protection for the electronics, and four aluminium bars (i.e. the arms of the quadrocopter) that support the four brushless AC motors and contain the wires from the motors to the speed controllers (see figure 2.6).



(a) *One motor.*



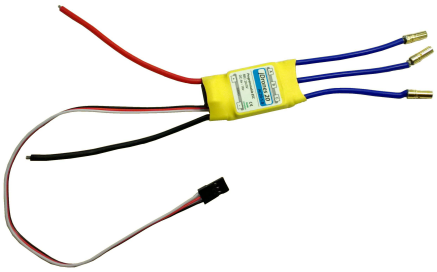
(b) *One motor and the arm it is attached to.*



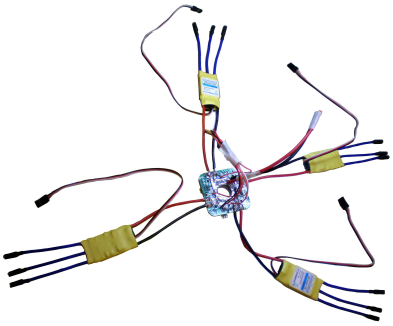
(c) *The four arms with the plastic structure that keeps them together.*

Figure 2.6 – Motors and arms.

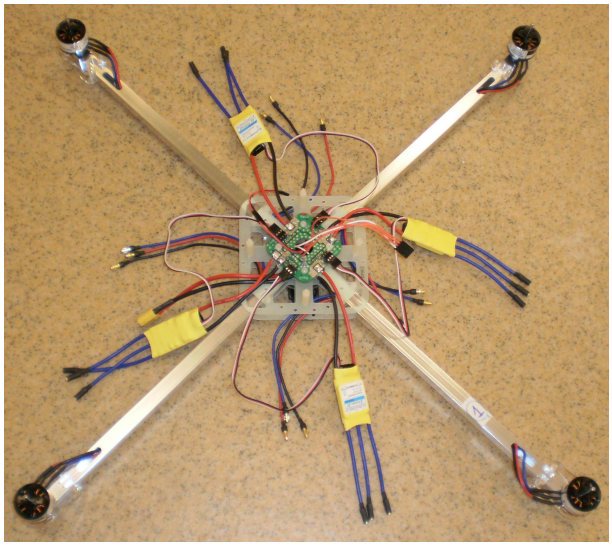
The electronic speed controllers (ESC) are soldered to the power distribution board and they get from it both the power and the signal that is generated by the ArduPilot board (figure 2.7).



(a) *One ESC.*



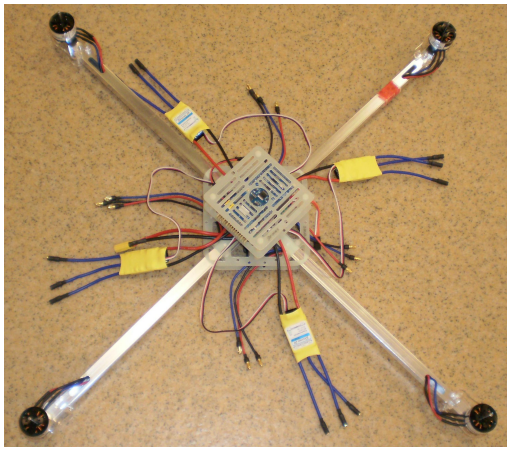
(b) *Power distribution board attached to the ESC's.*



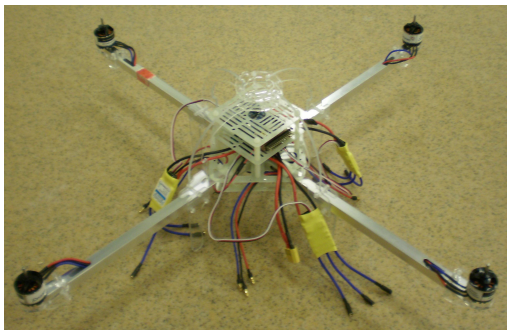
(c) *Quadrotor structure with motors, ESC's and power distribution board.*

Figure 2.7 – Esc and power distribution.

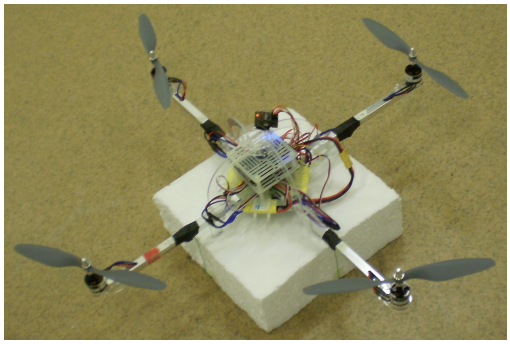
The structure of the quadrocopter is completed placing the ArduPilot on the top with its own structure and building the external frame that protects the vehicle and make it stand still. The final configuration includes the propellers, that are fixed to the motors, and the receiver of the remote signal. A polystyrene structure has been added as base of the quadrocopter because the plastic legs are indeed very fragile and do not stand aggressive landings (see figure 2.8).



(a) *Positioning of the ArduPilot board in the frame.*



(b) *Complete frame with protections and legs.*



(c) *Final structure, with propellers, RC receiver and polystyrene base.*

Figure 2.8 – Frame and final configuration.

The propellers are plastic blades that are sold in pairs, the model of the ones we used is 10x4.5. The front/rear propellers are different from the left/right

ones, since as we remarked in section 2.2 the front and back ones must rotate counter-clockwise (*puller propellers*) and the others clockwise (*pusher propellers*, marked with the letter *R* after the size label) (see figure 2.9).

Figure 2.9 – Propellers of the quadrocopter, puller on the top and pusher on the bottom.

The battery that powers the quadrocopter is a 3 cells Li-Po battery (see figure 2.10) that provides 11.1V as output voltage and 2200 mAh as rated capacity. The average time of flight before the battery discharges is from 5 to 10 minutes, depending on the throttle that is applied by the motors.

Figure 2.10 – Li-Po battery of the quadrocopter.

2.5 Qualysis motion capture system

(a) Qualisys logo. **(b)** QTM logo.

Figure 2.11 – Qualysis and Qualisys Track Manager logos

The motion capture (MoCap) system provided by Qualisys consists in a certain number of IR cameras, each of them equipped with an infrared flash. The light of the flashes is reflected by small reflective balls (*markers*) that are placed in advance on the objects we are interested in tracking, and the cameras capture these reflected beams and compute relative position and size of the markers. After the information collected by every camera is transmitted via ethernet to a computer running the Qualisys Tracking Software (QTM) and the position of the cameras being known, the 3D position of the markers is computed by the software. If a group of markers have been previously gathered by the user forming a *body*, then the software is able to track not just the position of the markers, but also the 3D position and orientation of the body when it moves in the workspace.

In order for the cameras to accurately compute the position of the markers, a calibration process has to be carried out before data collection. This is performed by placing an L-shaped tool representing the reference system on the ground, and, after starting the procedure in the QTM software, carrying around a calibration wand and moving it in all directions for a preset time (see the tools in figure 2.12).

Figure 2.12 – Wand and reference system (`www.qualysis.com`).

In our testbed the cameras are twelve, hung in groups of three to the four corners of the ceiling (see figure 2.13). Ten cameras are Oqus 4 and the remaining two are Oqus 3+. These last have improved functions, such as light sensitivity, image and high-speed video mode; for the full specifications see table 2.1.

Figure 2.13 – Group of three Oqus cameras.

Camera	**Normal mode (full FOV)**		**High Speed mode (full FOV)**		**Max fps (reduced FOV)**
Oqus 4	3 MP	480 fps	n/a		10000 fps
Oqus 3+	1.3 MP	500 fps	0.3 MP	1750 fps	10000 fps

Table 2.1 – Specifications of the Oqus cameras (`www.qualisys.com`). `FOV` stands for Field Of View and `fps` for frames per second.

In figures 2.14 and 2.15 we show two screenshoots from the QTM software, both are images from the point of view of one of the Oqus 3+ cameras framing the quadrocopter: in the first one we can see the five markers that have been placed on the quadrocopter, in the second one we overlap the video recorded by the camera and the 3D visualization of the body and the reference system.

The 3D data[4] obtained by the Qualisys software can be requested from other computers connected to the PC that runs the QTM proprietary software via a TCP/IP connection and exploiting the libraries that Qualisys provides for NI Labview and Matlab. For all our experiment in the testbed, the frequency of the camera has been set to 100[Hz]; this is enough considering that is ten times the speed of the control loop for the quadrocopter (see section 4).

The performances of the Qualisys system are really good as far as the precision is concerned: it has been observed that for a recorded *body* in a fixed position the variance of the recorded position is less than 1[mm], that is definitely reliable for our purposes. Therefore there is absolutely no need to filter the measurements to get more reliable information on position.

[4] 3D position, pitch, roll and yaw of the body

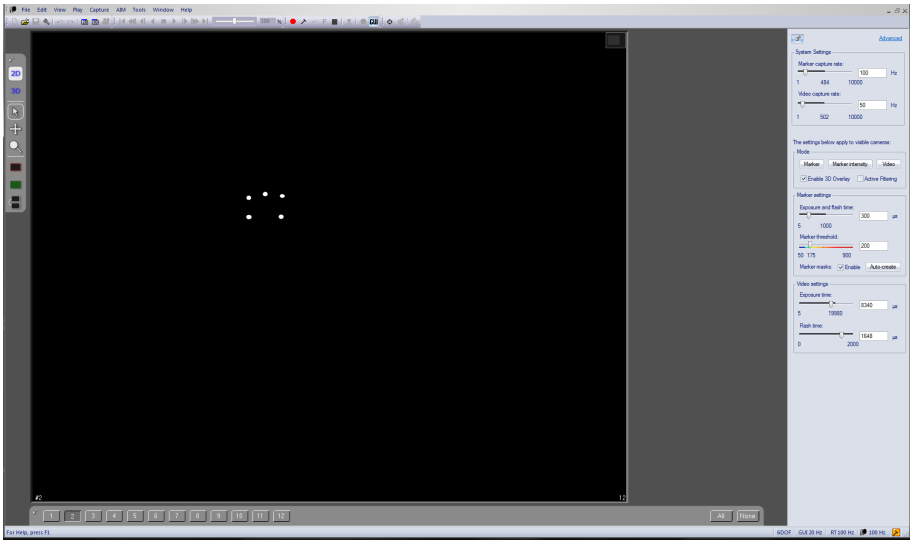


Figure 2.14 – Snapshot of the markers viewed by an Oqus 3+ camera.

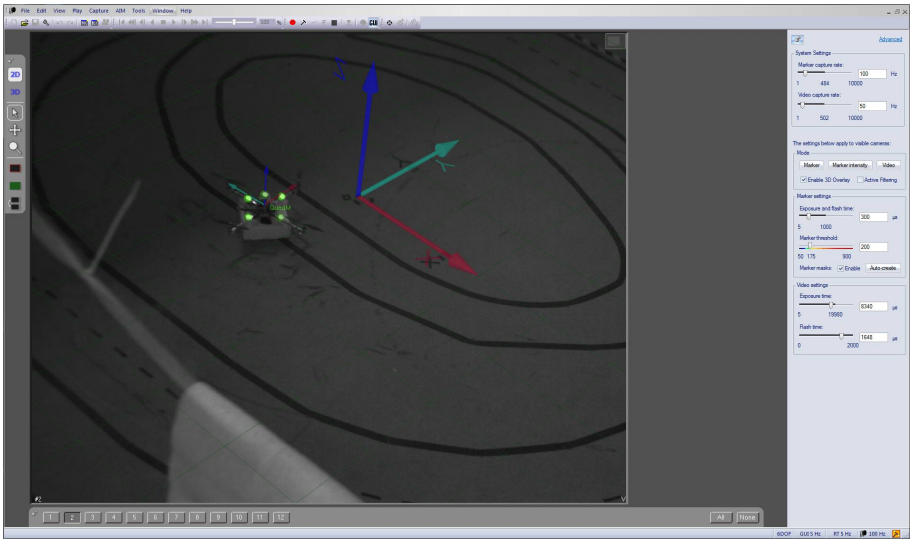


Figure 2.15 – Snapshot of the video capturing of an Oqus 3+ camera, overlayed with the quadrocopter body and the reference system.

2.6 Communication link

The wireless communication between the controller PC and the quadrocopter is made using a pair of Tmote Sky devices. Tmote Sky is an ultra low power wireless module for use in sensor networks, monitoring applications, and rapid application prototyping. Figure 2.16 shows the components of a mote: for more information about them and Tmote Sky in general, refer to the datasheet [18].

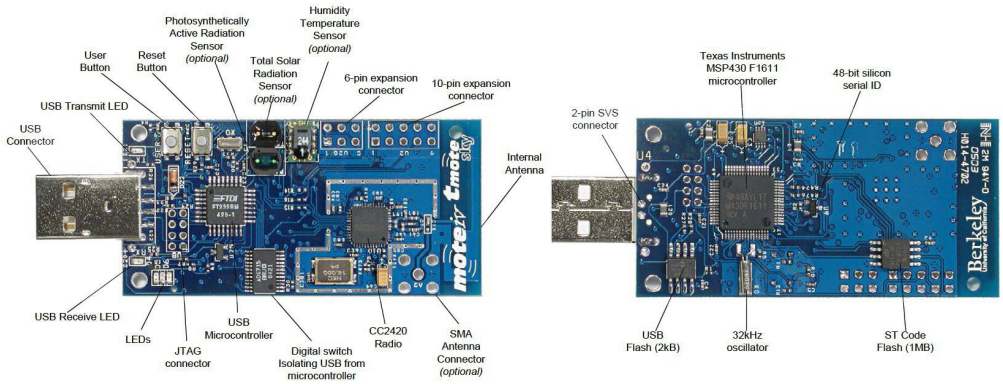


Figure 2.16 – Tmote Sky components.

One mote is plugged into an USB port of the controller computer and the other one is attached to the quadrocopter in the manner we described in section 2.3. They have been programmed using TinyOS, exploiting the code available in the Smart Mobility Lab repository[5]; refer to the manuals in the same website for the procedure for programming the motes.

The scheme of the communication link is depicted in figure 2.17, we here summarize it and refer once again to the laboratory manual [16] for a more detailed description:

- A C-based serial forwarder is used to send data from the computer to the PC mote: the serial forwarder creates a TCP server process in the computer that listens to the local port we specify and forwards the received data to the PC mote.
- The Labview program creates a TCP client process that connects to the TCP server, therefore creating a TCP connection. The data we need to

[5] `https://code.google.com/p/kth-smart-mobility-lab/source/browse/#svn%2Ftrunk%2FTinyOS_Code%2FQuad_TinyOS/`

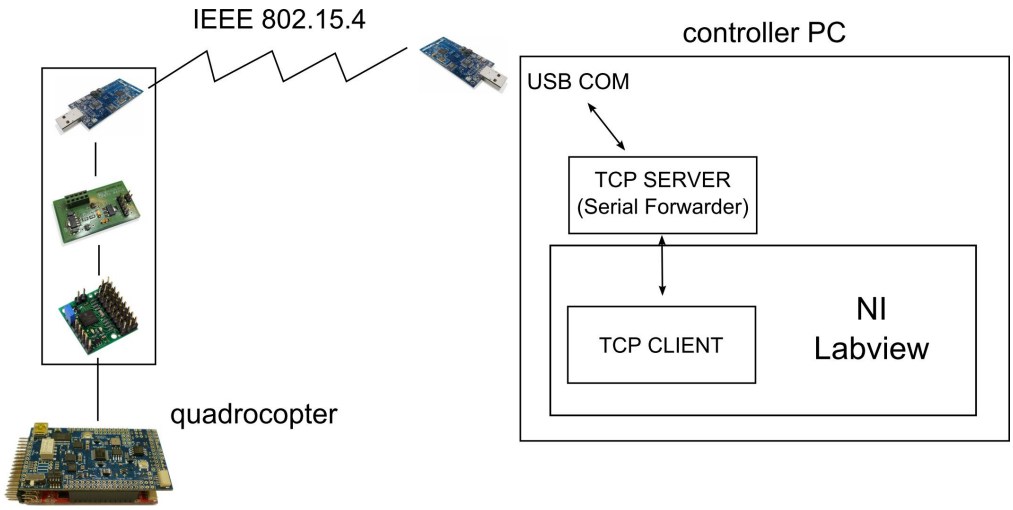


Figure 2.17 – Communication link.

send are thus forwarded to the serial forwarder via this TCP connection and in turn they are received by the PC mote;

- The motes use the IEEE 802.15.4 protocol to send data wirelessly from the PC one (base) to the quadrocopter one (actuator). Please note that the pair of motes need to be programmed on the same channel and they should be the only ones using this channel in the working environment, in order to avoid packet collisions;

- The data received by the actuator mote are forwarded via serial to the serial board and the Pololu board and finally reach the Arduino board of the quadrocopter.

2.7 Preliminary operations

Before flight, some procedures must be carried out in order to set the quadrocopter in a proper way, exploiting the APM software. We will list them briefly, referring once again to [17] for more details.

Radio range calibration. This operation has to be done before the first flight and whenever the quadrocopter seems not to respond accordingly to the input commands. It consists in setting the end points of each of the radio signals that

feed the APM board and is done referring to the *Radio calibration* section of the *Configuration* tab of the APM software.

Quadrocopter level. This operation has also to be done before the first flight and whenever the quadrocopter seems not to respond accordingly to the input commands. It consists in calibrating the accelerometer of the APM board and this is performed automatically by the software (*Level* section of the *Configuration* tab): the user only needs to orientate the APM accordingly to the onscreen instructions and keep it still while the software records the data it needs.

Esc calibration. This operation has to be done right after building the quadrocopter and whenever we suspect that the speed of the motors are not correctly balanced. It aims to make the speed controllers react in the same fashion to the APM and to the RC commands. Automatic calibration is the easiest way to perform it, since it is just necessary to hear the tones emitted by the APM and move the remote sticks[6] as a consequence.

Onboard PIDs. This operation is not strictly necessary, but we observed that it provides better results in the height control. Since the motors the quadrocopter carries are quite powerful in respect to the weight of the vehicle itself, it is recommended to modify the proportional gain parameter of the stabilizing onboard controller.[7] The default parameter is the one for medium-size motors (0.110), we decided to switch it to 0.100, that is the mean value between the ones recommended for medium-size and large-size motors.

[6] or changing the input values if we are using remote Labview control

[7] This can be done via the APM software, in the section Configuration → Standard parameters → Arducopter PIDs → Stabilization, angular rate control

Chapter 3

Quadrotor modeling and identification

3.1 Quadrotor mathematical model

We want to provide here a mathematical model of the quadrotor, exploiting Newton and Euler equations for the 3D motion of a rigid body, i.e. the so-called *First principles modeling*) (see for instance [19], [20], [21]). The goal of this section is obtaining a deeper understanding of the dynamics of the quadrotor and provide a model that is enough reliable for simulating its behavior.

We start by defining two reference frames: the body reference frame (B), that is the local reference system of the quadrotor, and the Earth inertial reference frame. In the continuation, all the quantities referring to the body and Earth frames will be written with B or E superscripts respectively.

Let us denominate as

$$q^E \triangleq [\ x \quad y \quad z \quad \phi \quad \theta \quad \psi\ ]^T$$

the vector containing the linear and angular position of the quadrotor in the earth frame and

$$\dot{q}^B \triangleq [\ u \quad v \quad w \quad p \quad q \quad r\ ]^T$$

the vector containing the linear and angular velocities in the body frame. From 3D body dynamics, it follows that the two reference frames are linked by the

following relations (see [19], [22]):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = {}^{E}R_{B} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \triangleq \begin{bmatrix} c_\psi c_\theta & -s_\psi c_\phi + c_\phi s_\theta s_\phi & s_\psi s_\phi + c_\psi s_\theta c_\phi \\ s_\psi c_\theta & c_\psi c_\phi + s_\psi s_\theta s_\phi & -c_\psi s_\phi + s_\psi s_\theta c_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$
$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = {}^{E}T_{B} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \triangleq \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi / c_\theta & c_\phi / c_\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \tag{3.1}$$

so we get

$$\dot{q}^{E} = \left[\begin{array}{c|c} {}^{E}R_{B} & 0_{3\times 3} \\ \hline 0_{3\times 3} & {}^{E}T_{B} \end{array}\right] \dot{q}^{B} \tag{3.2}$$

We indicated for the sake of brevity $sin(\alpha)$, $cos(\alpha)$ and $tan(\alpha)$ as s_α, c_α and t_α, for a generic angle α.

Newton's law states the following matrix relation for the total force acting on the quadrotor in the body frame:

$$F^{B} = m\dot{V}^{B} + \Omega^{B} \times mV^{B}, \tag{3.3}$$

where $F^{B} \in \mathbb{R}^{3\times 1}$ is the total force, m is the mass of the quadrotor, $V^{B} \in \mathbb{R}^{3\times 1}$ is the linear velocity of the quadrotor and $\Omega^{B} \in \mathbb{R}^{3\times 1}$ is the angular velocity of the quadrotor. Euler's equation gives the total torque applied to the quadrotor:

$$\tau^{B} = J\dot{\Omega}^{B} + \Omega^{B} \times J\Omega^{B}, \tag{3.4}$$

where $\tau^{B} \in \mathbb{R}^{3\times 1}$ is the total torque and $J \in R^{3\times 3}$ is the diagonal inertia matrix.

The kinematic equations (3.3) and (3.4) stand as long as we assume that the origin and the axes of the body frame coincide with the center of mass of the quadrotor and the axes of inertia, respectively.

Let us now consider the dynamics of the quadrotor, i.e. how the total force and the total torque vectors are composed.

Gravity. Gravity has effect on the total force and its direction is along the z axis of the earth frame[1]:

$$F_{gr}^{B} = {}^{B}R_{E}\, F_{gr}^{E} = ({}^{E}R_{B})^{T} \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} = \begin{bmatrix} mgs_{\theta} \\ -mgc_{\theta}s_{\phi} \\ -mgc_{\theta}c_{\phi} \end{bmatrix} \tag{3.5}$$

Gyroscopic effects. Gyroscopic contributions are present if the total sum of the rotational speeds of the rotors $\Omega_{Rtot} = -\Omega_{R1} + \Omega_{R2} - \Omega_{R3} + \Omega_{R4}$ is not zero and they influence the total torque adding the following term (see [19], [23]):

$$\tau_{gyro}^{B} = J_{Rot} \begin{bmatrix} -q \\ p \\ 0 \end{bmatrix} \Omega_{Rtot}, \tag{3.6}$$

where J_{Rot} is the moment of inertia of any rotor.

Control inputs. We here consider the inputs that can be applied to the system in order to control the behavior of the quadrotor. The rotors are four and the degrees of freedom we control are as many: commonly, the control inputs that are considered are one for the vertical thrust and one for each of the angular movements. Let us consider the values of the input forces and torques proportional to the squared speeds of the rotors (see [19] for aerodynamic motivations); their values are the following:

$$\begin{aligned} U_1 &= b(\Omega_{R1}^2 + \Omega_{R2}^2 + \Omega_{R3}^2 + \Omega_{R4}^2) \\ U_2 &= bl(\Omega_{R3}^2 - \Omega_{R1}^2) \\ U_3 &= bl(\Omega_{R4}^2 - \Omega_{R2}^2) \\ U_4 &= d(-\Omega_{R1}^2 + \Omega_{R2}^2 - \Omega_{R3}^2 + \Omega_{R4}^2), \end{aligned} \tag{3.7}$$

wherer l is the distance between any rotor and the center of the drone, Ω_{Ri} is the angular speed of the i-th rotor, b is the thrust factor, d is the drag factor. The inputs U_i can be directly related, at least on first approximation, respectively the vertical, pitch, roll and yaw accelerations (see also the oncoming equation (3.9)).

As we summarize the above-obtained relations in one matrix equation we get

[1] Since ${}^{B}R_{E}$ is a rotational matrix, it is orthogonal, i.e. its inverse equals its transposed.

[2]

$$\left[\begin{array}{c|c} mI_{3\times3} & 0_{3\times3} \\ \hline 0_{3\times3} & J \end{array}\right] \ddot{q}^B + \left[\begin{array}{c|c} 0_{3\times3} & -S_k(mV^B) \\ \hline 0_{3\times3} & -S_k(J\Omega^B) \end{array}\right] \dot{q}^B = \left[\begin{array}{c} F^B \\ \hline \tau^B \end{array}\right], \tag{3.8}$$

where the second term can be written explicitly as

$$\left[\begin{array}{c} F^B \\ \hline \tau^B \end{array}\right] = \left[\begin{array}{c} mgs_\theta \\ -mgc_\theta s_\phi \\ -mgc_\theta c_\phi + U_1 \\ \hline J_{rot}(-q)\Omega_{Rtot} + U_2 \\ J_{rot}(p)\Omega_{Rtot} + U_3 \\ U_4 \end{array}\right].$$

Now, we are interested in obtaining the equations of motion in the Earth reference frame, because this is the frame the motion capture system refers to: we can use relation (3.2) to switch to the Earth reference frame. We notice that for small variations of the pitch and roll angles, the transfer matrix ${}^E T_B$ approaches the identity matrix: we can exploit this approximation assuming that the quadrotor flies close to hovering state.

Applying relation (3.2) to (3.8) and rewriting the matrix equation in form of system, we obtain the following:

$$\begin{cases} \ddot{x} &= -(c_\phi s_\theta c_\psi + s_\phi s_\psi)\frac{U_1}{m} \\ \ddot{y} &= -(c_\phi s_\theta s_\psi - s_\phi c_\psi)\frac{U_1}{m} \\ \ddot{z} &= g - (c_\phi c_\theta)\frac{U_1}{m} \\ \ddot{\phi} &= \dot{\phi}\dot{\psi}\left(\frac{J_Z - J_X}{J_Y}\right) - \frac{J_R}{J_Y}\dot{\phi}\Omega_{Rtot} + \frac{U_2}{J_Y} \\ \ddot{\theta} &= \dot{\theta}\dot{\psi}\left(\frac{J_Y - J_Z}{J_X}\right) - \frac{J_R}{J_X}\dot{\theta}\Omega_{Rtot} + \frac{U_3}{J_X} \\ \ddot{\psi} &= \dot{\theta}\dot{\phi}\left(\frac{J_X - J_Y}{J_Z}\right) + \frac{U_4}{J_Z} \end{cases} \tag{3.9}$$

Several models in literature are obtained in the way we described, either exploiting first principles modeling (e.g. [19], [24], [25]) or Euler-Langrange equations (e.g. in [23]); since gyroscopic effects are not very effective for relatively small values of the pitch and roll angles, they are often excluded from the model

---

[2] We remind that the general cross product $a \times b \triangleq \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$ can be written as $S_k(a)\, b = -S_k(b)\, a$, where $S_k(a)$ is the skew-symmetric matrix $\begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$

[26], [4], [27], [12]. Other contributions apart the ones we dealt with are considered e.g. in [23], [20] and consist for instance in aerodynamic damping, blade flapping and ground effects.

The approach for modeling that we described takes into account the main contributions for the flight dynamics. However, the control inputs we defined are explicitly linked to the speed of the rotors by relations (3.7) and, as seen in the previous chapter, in our test-bed we do not directly control the motor speeds, nor we need to consider the inner dynamics of the motors (i.e. the transfer function from the input voltage to the output torque) thanks to the onboard controller.

Basing on these considerations, let us consider a model for the quadrotor that fits the control input we have at our disposal in the testbed. As a first approximation we can state that the values for throttle, pitch, roll and yaw movements we send to the quadrotor are the control inputs U_i we described in the previous section. Since in equation 3.9 the values of inertia are unknown[3] and willing to understand how the behavior of the actual system relates to those equations, we now aim to retrieve the relations between our inputs and effective vertical thrust, pitch, roll and yaw in the form of transfer functions (t.f.).

These transfer functions can be estimated thanks to the knowledge of the inputs (expressed as byte values) and outputs of the system (expressed as height wrt the throttle input and as angular value in degrees wrt the pitch, roll and yaw), and this process will be content of the following section.

[3] the value of the mass, instead, has been measured and is 1.1[Kg], polystyrene base included.

3.2 Transfer functions identification

In order to perform the aforementioned identification, we will follow the scheme depicted in figure 3.1, that is typical of this kind of procedure.

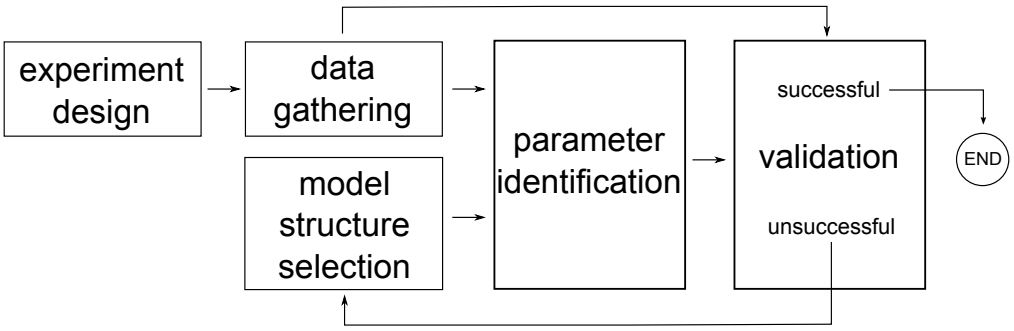


Figure 3.1 – Generical identification procedure

The identification process aims to give a clearer view on the system, in particular how the agent reacts to the input signals sent by the user or the control program.

Each of the four transfer functions we want to identify is related to one single input. So the experiments in which the data for the identification and validation have been collected consist in test flight focused on the variation of one single input: for instance, when collecting the data for identifying the *roll* transfer function, after a take-off phase the test consisted in moving the agent with manual control only along the lateral axis (roll axis) and collecting the pair of sequences of input bytes and output roll, the latter collected by the motion capture system.

We decided to treat the system as discrete-time and identify the transfer functions via Matlab's System Identification Toolbox. The idea is trying to identify polynomial models of the form Output-Error (OE) i.e. models with the generic structure

$$y(t) = \frac{B(z)}{F(z)} u(t - n_d) + e(t),$$

where $B(z)$ and $F(z)$ are the polynomials to identify, n_d is the input-output delay expressed as number of samples and $e(t)$ is white noise. We chose this kind of model since we are not interested in the structure of the noise, that we indeed assume to be white, and since under hypothesis of small angular values, equations (3.9) let us relate the input and output at our disposal in the following

way:

$$\begin{cases} \ddot{z} \approx g - \frac{U_1}{m} \\ \ddot{\phi} \approx \frac{U_2}{J_Y} \\ \ddot{\theta} \approx \frac{U_3}{J_X} \\ \ddot{\psi} \approx \frac{U_4}{J_Z} \end{cases} \tag{3.10}$$

so a linear input-output transfer function can be considered a fair approximation as long as the previous ones stand.

Let us now describe the procedure for identification; for each unknown transfer function:

- The collected data (both inputs and outputs) have been divided into identification and validation data;
- Two `iddata` objects are created, one for the identification and one for the validation;
- The number of poles, zeroes and delay samples is set and Matlab function `oe` is run on the data. This function is based on the prediction error minimization method (PEM): starting from a chosen structure of the model, a predictor of the present output is built, based on past inputs and outputs. The PEM method estimates the parameters of the model minimizing the prediction errors, i.e. minimizing a cost function that depends on these errors, that in turn depend on the unknown parameters and the measured outputs. For a detailed description and analysis of the PEM method and system identification in general, see [28];
- The obtained transfer function is tested on the validation data using the `compare` function of the toolbox, that gives both a numerical fit and a visual depiction of the suitability of the estimated model;
- If the results are not satisfactory we change the order of the transfer function and repeat the procedure.

We aim to find a decent trade-off between the order of the transfer function and the fit we get with the validation data. The approximations (3.10) suggest that second order models should already give proper results, since inputs and outputs (z axis position, pitch, roll and yaw angles) are linked by second order integrators and constant values. Of course the higher the order of the identified

transfer function, the better the fit with the dataset used to obtain it, but if we test the same transfer function on a different dataset the results will be extremely inaccurate, since the large variance of the estimation would make the model useless (this is referred in literature as the *bias vs. variance dilemma*).

In the following we report the results we got for the identification of the four transfer functions. Please note that the roll and pitch dynamics are, at least in theory, equivalent; in order to check this we will also cross-test the results obtained with the pitch data with the ones obtained with the roll data.

3.2.1 Throttle transfer function

The dynamic of the throttle has been the most difficult to identify in a satisfactory way, since the relation between throttle and vertical thrust is strictly dependent on the battery charge and if this is quite low we are basically not able to get an adequate response of the vehicle to the input. So in order to overcome this problem the datasets were collected with full charged battery. A second order linear model was not accurate enough to describe the dynamics of the throttle; a third order transfer function with 8 samples delay was obtained, but still its validation gives just a qualitative decent behavior:

$$B(z) = 16.74z^{-8} - 31.25z^{-9} + 14.51z^{-10}$$

$$F(z) = 1 - 2.829z^{-1} + 2.659z^{-2} - 0.83z^{-3}$$

Fits of 76.95% and -15.41% were obtained with identification and validation data respectively (see figure 3.2). Trials with higher order OE and ARMAX models were conducted as well, but the improvements were not satisfactory.

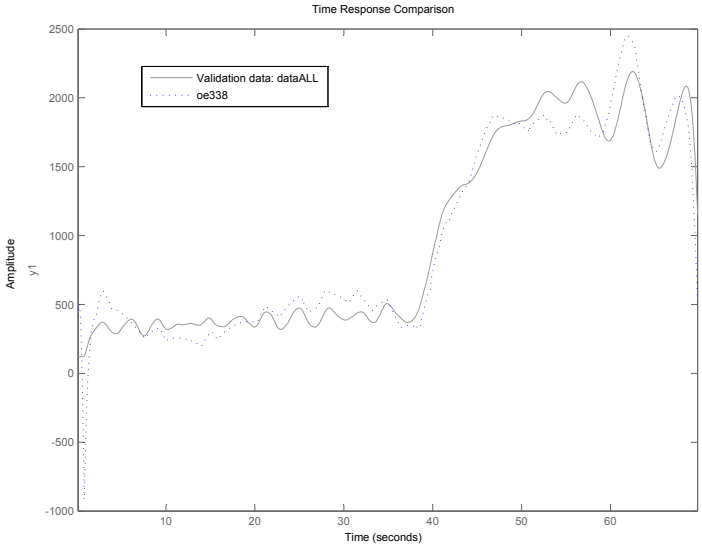
3.2.2 Yaw transfer function

The yaw transfer function has been estimated with excellent results as a second order transfer function with 3 samples delay:

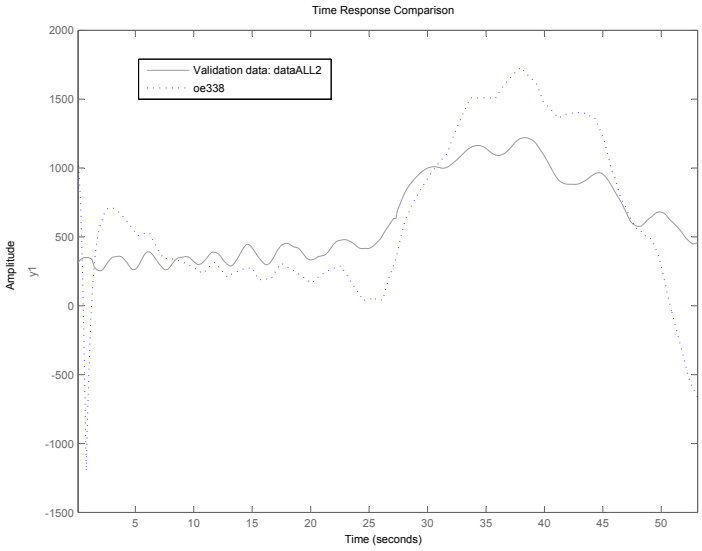
$$B(z) = -0.2236z^{-3} + 0.2236z^{-4}$$

$$F(z) = 1 - 1.998z^{-1} + 0.9983z^{-2}$$

Fits of 91.69% and 81.82% were obtained with identification and validation data respectively (see figure 3.3).

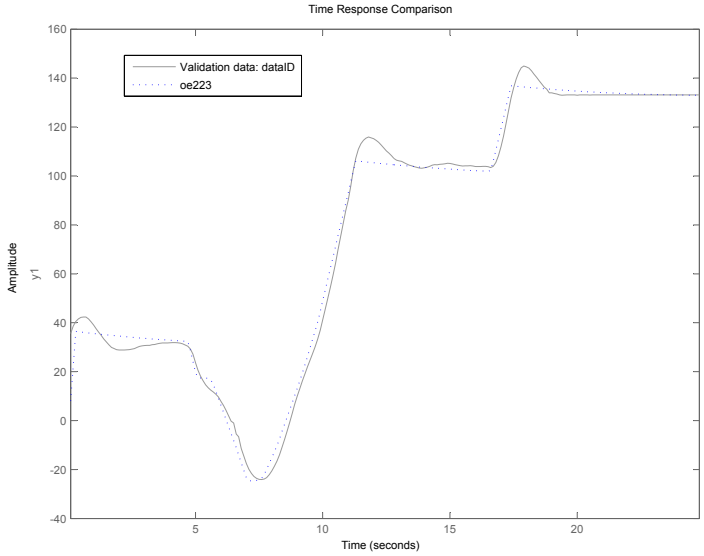


(a) Fit between identified t.f. output (dashed) and identification data (solid)

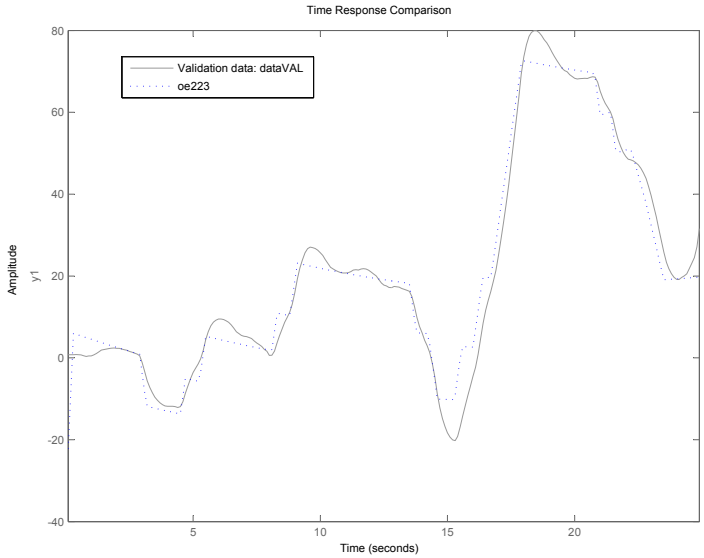


(b) Fit between identified t.f. output (dashed) and validation data (solid)

Figure 3.2 – Throttle identification results



(a) Fit between identified t.f. output (dashed) and identification data (solid)



(b) Fit between identified t.f. output (dashed) and validation data (solid)

Figure 3.3 – Yaw identification results

3.2.3 Pitch and Roll transfer functions

Both for the pitch and roll transfer functions we get satisfactory results. Even though second order transfer functions provided a visible concordance between inputs and outputs, we decided that it was worthy to keep as valid third order t.f.'s (with one delay sample), since the fits with the validation data were more than double the ones with second order t.f.'s.

For the pitch we obtain:

$$B(z) = 0.09248z^{-1} - 0.3035z^{-2} + 0.2111z^{-3}$$

$$F(z) = 1 - 2.295z^{-1} + 1.841z^{-2} - 0.5439z^{-3}$$

Fits of 64.78% and 48.31% were obtained with identification and validation data respectively (see figure 3.4).

For the roll we obtain:

$$B(z) = 0.09248z^{-1} - 0.3035z^{-2} + 0.2111z^{-3}$$

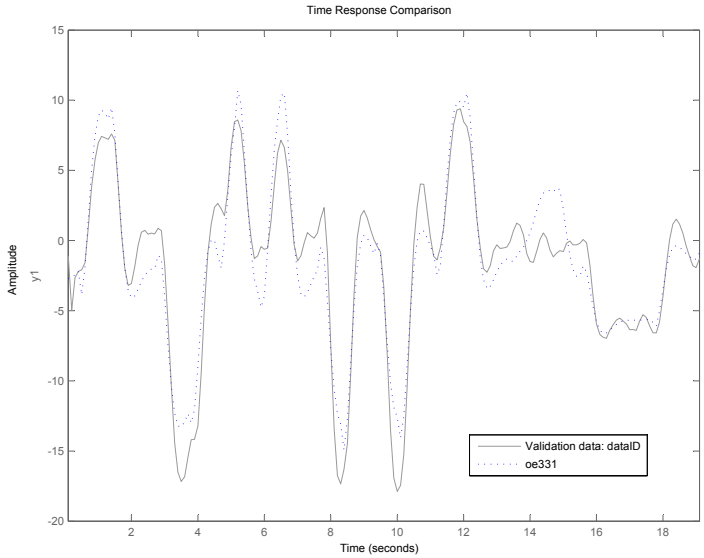
$$F(z) = 1 - 2.295z^{-1} + 1.841z^{-2} - 0.5439z^{-3}$$

Fits of 67.15% and 54.5% were obtained with identification and validation data respectively (see figure 3.5).

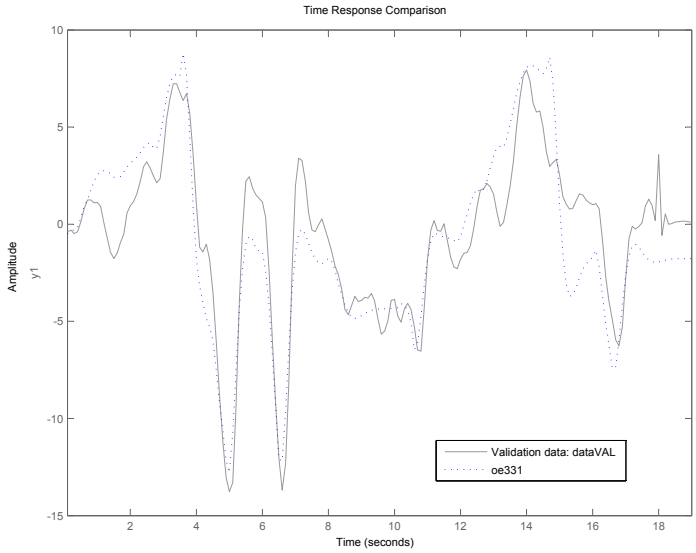
In order to compare the t.f.'s for the pitch and roll dynamics we applied the model obtained for the roll[4] to the validation data of the pitch. We obtained a fit of 42.86% (see figure 3.6), that is very close to the one we had with the pitch model itself.

From the zeroes-poles diagrams (3.7) of the two transfer functions we can see that their positions are really close so the dynamics for the pitch and roll can indeed be considered equivalent to a good approximation.

[4] The sign of the transfer function had to be inverted in order to get concordance

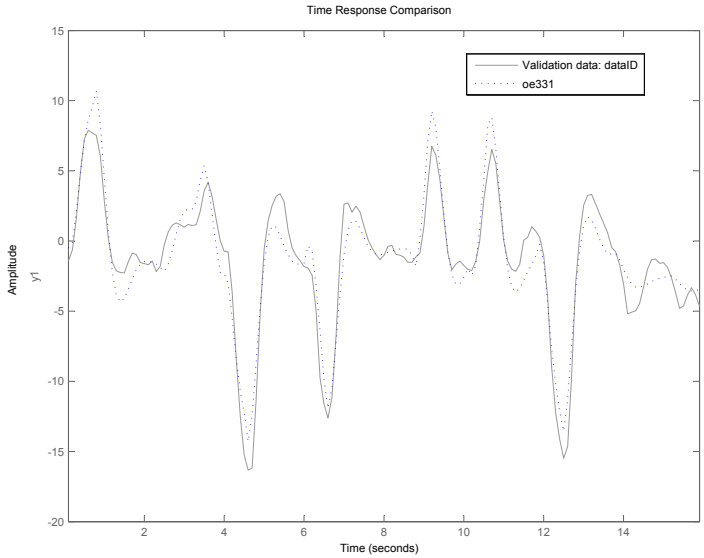


(a) Fit between identified t.f. output (dashed) and identification data (solid)

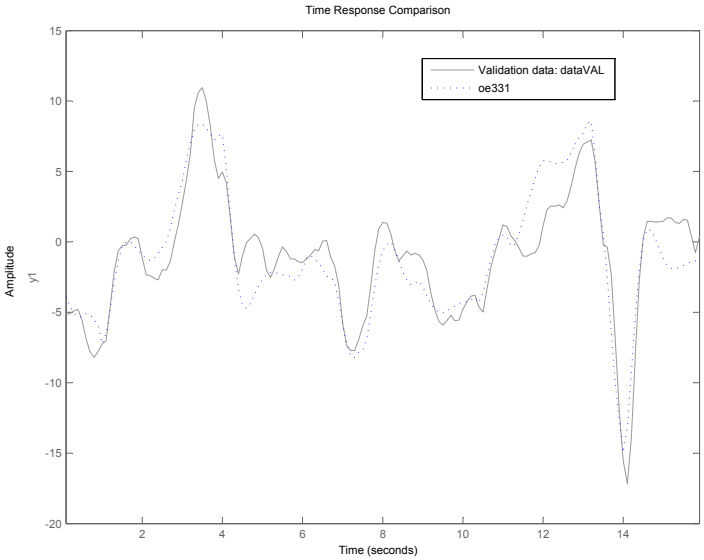


(b) Fit between identified t.f. output (dashed) and validation data (solid)

Figure 3.4 – Pitch identification results



(a) Fit between identified t.f. output (dashed) and identification data (solid)



(b) Fit between identified t.f. output (dashed) and validation data (solid)

Figure 3.5 – Roll identification results

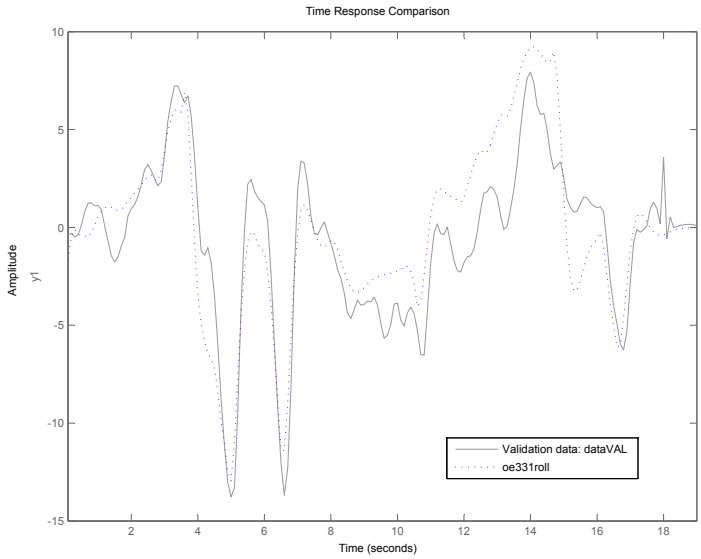


Figure 3.6 – Roll model applied on pitch dataset.

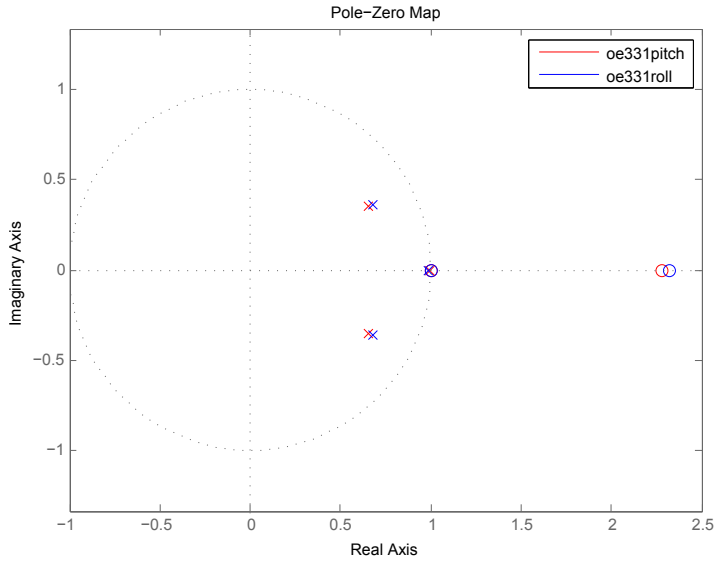


Figure 3.7 – Zeroes-poles diagram of estimated transfer functions for pitch (red) and roll (blue).

The identification results show that the less noisy dynamic is the one of the yaw angle, whereas the relation between the throttle input and the vertical position/acceleration is not straightforward and is affected by significant disturbances. These considerations will match with the control presented in next chapter, since if on the one hand the yaw is easily controllable with a proportional controller and the pitch and roll controls are satisfactory, on the other hand the height control is the most challenging, even if it does not affect critically the success of the experiments.

Chapter 4

Quadrotor control

In this section we will present the controller that has been designed and implemented in the testbed. Since the setup allows online tuning of the controllers with safety precautions that are not too onerous, we decided to bypass the tuning on a simulated model, that would request to be refined online anyway, once implemented in the testbed.

4.1 Controller description

The aim of the controller is point-to-point navigation with obstacle avoidance. The controller is structured in a layered fashion, as we can see in figure 4.1:

- The most external control is the path planning, i.e. the computation of a path to the destination point that provides obstacle avoidance. This is done exploiting the implementation of a navigation-function based control in `Matlab`. The path planning provides the waypoints that have to be followed by the agent and deals with the switching from one waypoint to the following when either a temporal or a positional condition is triggered.
- The references provided by the path planning feed four PID controllers[1], one for each of the throttle, pitch, roll and yaw movements of the quadrotor and the output of the controllers is sent via wireless to the agent.
- As we already mentioned, the onboard controllers of the quadrotor stabilize its position exploiting the references for throttle, pitch, roll and yaw.

[1] We will not focus on the theory of the PID control, referring to any basic textbook of automatic control for clarifications.

- Additional control is also provided, in the form of a safety control that makes the quadrotor land if it approaches non-fly zones and a manual control, that can be either partial (control of one of more of the inputs, while the others are generated by the PID's) or full (the quadrotors only responds to the joypad commands). We can say that the quadrocopter has an onboard safety feature as well, since before it can fly it must be *armed*, i.e. it has to receive a fixed sequence of inputs (0 64 64 127) for at least 4 seconds before the motors can be turned on by a throttle input; moreover, when the throttle is cut and remains null for some time, or the user sends continuously the *disarm* sequence (0 64 64 0), the quadrocopter needs to be armed again in order to restart to fly.

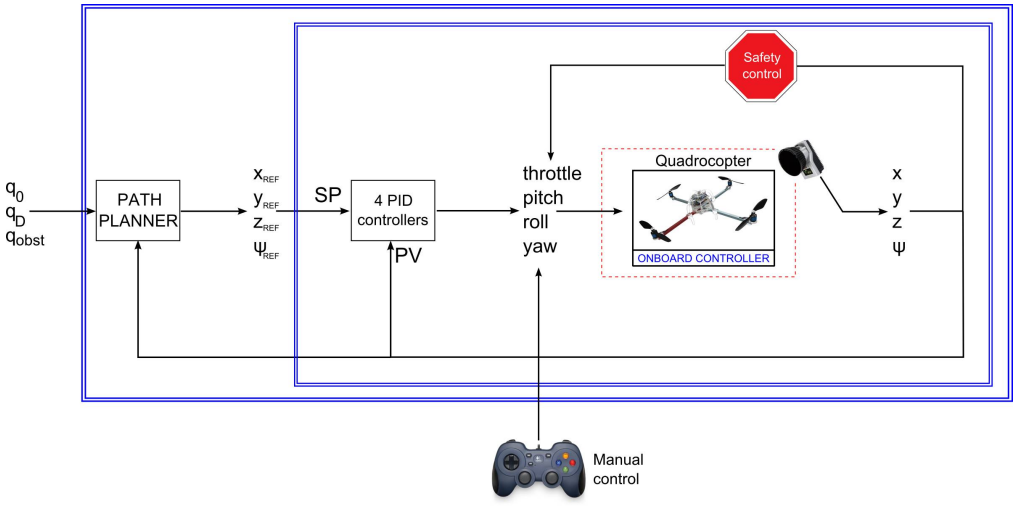


Figure 4.1 – Control structure

In order to control separately the movements along the x, y and z axis and the yaw rotation, we exploited both the model in 3.1 and heuristic considerations we infer from the observation of the behavior of the vehicle in the workspace.

First of all, let us state that the roll and pitch angles cannot assume large values during the flight. This is said both for safety reasons, since we do not want the quadrocopter to perform off-hand maneuver, and in order to be able to rewrite the equations of the model with the usual approximations $sin(\alpha) \approx \alpha$ and $cos(\alpha) \approx 1$ that stand for small values of α[2]. Thus, we are able to rewrite

[2] Let us say the approximation is pretty good if we bound the angles to 40°

the equations for the movements along the earth-fixed axes as follows:

$$\begin{cases} \ddot{x} \approx -(\theta cos(\psi) + \phi sin(\psi))\frac{T}{m} \\ \ddot{y} \approx -(\theta sin(\psi) - \phi cos(\psi))\frac{T}{m} \\ \ddot{z} \approx g - \frac{T}{m} \end{cases} \tag{4.1}$$

Let us suppose for now that the yaw angle is zero. We can assume this without loss of generality, since in our application (navigation with obstacle avoidance) we are just interested in the position of the quadrotor: being the agent symmetrical wrt its vertical axis, a rotation around this same axis does not affect the distance from any obstacle. Standing this assumption, we can rewrite (4.1):

$$\begin{cases} \ddot{x} \approx -\theta\frac{T}{m} \\ \ddot{y} \approx \phi\frac{T}{m} \\ \ddot{z} \approx g - \frac{T}{m} \end{cases} \tag{4.2}$$

From the previous system we see clearly that, as long as the approximations we made stand and for a fixed value of the thrust T, the movement along the earth-fixed x and y axes can be controlled independently acting on the pitch and roll inputs. Values of the acceleration $\frac{T}{m}$ that are close to the gravity acceleration g are the ones that keep the quadrocopter in stable flight, that is the agent neither falls to the ground nor *escapes* towards the ceiling. Unfortunately, this range of values of the thrust do not correspond to a fixed range of the throttle input, since the relation between the two depends widely on the battery level, and also for this reason the controller on the vertical motion is the one that gives the greatest issues, as we will show in the following.

Before proceeding with the controller description, let us adapt equations 4.2 to the reference system of the motion capture we deal with in the lab, that is rotated wrt the one we exploited to obtain the models of section 3.1. The system we obtain and will use for the controller is

$$\begin{cases} \ddot{x} \approx -\phi\frac{T}{m} \\ \ddot{y} \approx -\theta\frac{T}{m} \\ \ddot{z} \approx \frac{T}{m} - g \end{cases} \tag{4.3}$$

Please note that in the testbed the z axis is always positive, in contrast to the previous situation.

The four PIDs we implement are:

1. One full PID controller for the throttle, that controls the position along the earth-fixed z axis. We remark here that the control of the throttle influences the magnitude of the accelerations along the x and y axes as well, so higher values for the throttle produce more aggressive movements along these axes, for the same pitch/roll angle;

2. One PD controller for the pitch, that controls the position along the earth-fixed x axis;

3. One PD controller for the roll, that controls the position along the earth-fixed y axis;

4. One P controller for the yaw, that controls the yaw angle to zero in order for system (4.3) to be valid. Please note that requiring the yaw angle to be zero is not strictly necessary, but the equations we get in this situation are really simple and allow us to control directly the x and y movements using only one input each. Moreover, since our agent does not carry any camera or end effector, there is no need to prefer one rotation in the xy plane to another.

We will describe specifically the aforementioned controllers and the way they were tuned in the next section. The PID controllers were implemented in NI Labview modifying the script *PID.vi* of the PID and Fuzzy Logic Toolkit in order to get three separate outputs, one for each contribution (proportional, integral, derivative): seeing how each of them acts on the output helps to understand which one should be modified in order to get the desired results, and in which amount. The PID VI implements a PID controller with

- Derivative action operating on the process variable instead than on the error, in order to avoid bumps (*derivate kicks*) due to the modification of the set point:

$$u_D(k) = -K_c \frac{Td}{\Delta T}(PV(k) - PV(k-1))$$

- Trapezoidal integration to avoid sharp changes in the integral action when there is a sudden change in the process variable or the set point

- Anti windup algorithm: it avoids that when the input saturates the integrator keeps on integrating a considerable error, therefore making it easy for the input to get back to the admissible interval. With this action the presence of high and long lasting overshoots on the output is limited. The pseudocode for this implementation of the anti windup is

$$\text{if } |u_P(k) + u_I(k)| > |\text{limit}|, \text{ then } u_I(k) = \text{limit} - u_P(k)$$

4.2 Controller tuning

We will deal with the controllers according to the order we followed to tune them. An idea of the order of magnitude of the control parameters was given by a previous work with the same quadrocopter hardware[3]. In the Labview controller program we left the possibility to change the PID parameters online, so if minor adjustment were to be tried to improve the performances, it was not necessary to stop the experiment to change them and run the program again.

We here remark that the sampling time we set for the loop that contains the reference reading, the control and the forwarding of the inputs to the quadrotor has been set to 100[ms]. This value has been set considering that the bottleneck for the loop speed is the wireless connection between the sender and receiver motes.

4.2.1 Yaw control

We decided to tune the yaw controller first, because its proper functioning is paramount for equations (4.3) to stand and thus for the x and y positions to be controlled independently. The tuning process consisted in placing the quadrocopter on the ground with a non-zero angle, give a throttle step for the quadrocopter to become airborne and observe the reactivity of the controller to set the desired orientation. No extra safety measures[4] were adopted for this tuning procedure, since the inplace rotation of the agent is not a movement that can be dangerous for people or provoke a crash. Nevertheless, we noticed that a consistent yaw input produces a short upward acceleration of the agent.

[3] EL2421 project course, KTH, 2012

[4] With the expression "no extra safety measures" we mean that no other actions were taken to improve safety apart from the safety net that protects the people in the room, the emergency stop if the quadrocopter enters a non-flight zone and the possibility to act immediately on the inputs with the joypad (manual control).

A proportional controller was considered enough for the yaw dynamics, since it is already very stable and also large initial errors of the angle are mitigated in short time. The introduction of an integral part would compensate for disturbances, but we noticed that for large initial errors even a small integral part will introduce an overshoot that we want to avoid since the yaw controller must be the most reactive, the other ones depending on its good performances. Moreover, even if a small static error in the yaw angle exists, we observed that this does not influence considerably the performance of the pitch and roll controllers. The value of the gain of the controller has been set to 0.04.

Please note that if we assume that the variations of the yaw angle are not very fast, we can deal with a yaw reference different from 0, simply adding a coordinate rotation (of magnitude equal to the value of the yaw) between the earth-fixed frame and the controlled frame, that is the body-fixed one.

4.2.2 XY plane position control

We gather the controller of the movements along the x and y axes since the dynamics they regulate are equal; of course this is a theoretical consideration, that is not completely true if the hardware of the quadrocopter presents asymmetries along the two longitudinal axes. We noticed anyway that even rather obvious bending of the arms or imperfections in the propellers do not influence really much the behavior of the controlled system. Moreover, the identification process we carried out revealed that the two dynamics can be considered equal with a good approximation.

The setting for the tuning of these controllers was the following: a wire was hung on the ceiling of the testbed and in order to tune a controller, say the pitch controller, the arms controlled by the roll movement were fixed with the wire and therefore kept steady, allowing just the pitch movement (see figure 4.2). This action is meant to observe just the dynamics we are interested to and avoid unsafe abrupt movements. The reference was set to the projection to the ground of the vertical position of quadrotor when hung to the wires and not moving.

The tuning was first done on the proportional gain starting from the an unitary value and varying it online according to how the quadrotor reacted to regulate the tilting speed, keeping in mind that being tracking precision an important feature for our controller, the movements should not be too aggressive. Indeed, in order to make the response smoother and restrict the overshoot a

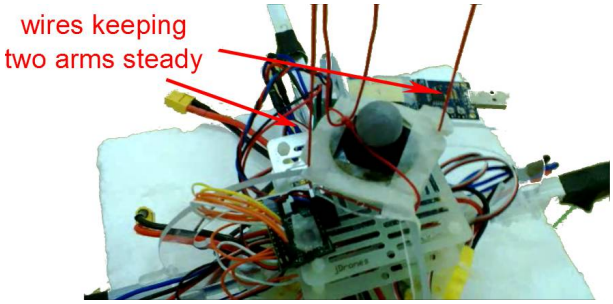


Figure 4.2 – Setting for the xy plane position control.

derivative component has been added.

Once converted to proper input byte to send to the quadrocopter, the output of the pitch and roll controllers have been limited to bound the output angles as mentioned in section 4.1. The limitation on the input bytes have been set to ± 20 wrt 64, that is the encoded value that corresponds to a value of zero for the desired angle; this bounds the pitch and roll angles to about $\pm 20°$, a satisfactory range for our maneuver.

In order to compensate any asymmetric behavior, for instance due to inaccurate calibration of the accelerometers of the quadrocopter, the user is given the possibility to shift (online) the input value corresponding to 0 in the output angles.

We set the same parameters for the pitch and roll controllers, namely 0.7 for the proportional and 0.7 for the derivative gain. Since different scenarios can require different behaviors by the position controllers, if necessary an integral contribution ($0.05 \div 0.1$) could be added when a higher overshoot could be allowed for the sake of speed and steady precision.

4.2.3 Height controller

The controller of the vertical position is the one that took longer to tune, and also the least accurate; its performances are deeply influenced by the battery status. In order to tune this controller we placed the quadrocopter on the ground and activate all the other (already tuned) controllers. A step was given to the throttle in order for the vehicle to become airborne and the proportional and derivative gains were tuned to keep it as steady as possible in the air, paying attention to immediately decrease and eventually cut the throttle with the manual control if the quadrocopter seemed to raise too fast in the air and going

out of control. After obtaining reasonably reduced oscillations, an integral term was added to eliminate the steady state error. A higher value of the integral part produces an overshoot but speeds up the response.

For the height controller the zero output of the PID is not transformed to a zero input for the throttle, indeed the output of the PID controller is added to a *base throttle* that is suitable to keep the quadrotor in the air. This value oscillates between 48 and 55, depending on the battery status (we considered here situations in which the battery is still utilizable for flying, of course if it is about to discharge completely the thrust decreases critically and it is impossible to keep the vehicle airborne). Moreover, for safety reasons the input byte we send to the quadrotor is bounded as well, in order to limit the maximum height it can achieve (the upper bound with full battery is set to 63). The gains we use for the height controller are 8 for the proportional gain, 1.24 for the integral gain and 0.31 for the derivative gain.

We summarize in table 4.1 the controller gains we will use in the experiments.

Controller	K_P	K_I	K_D
Heigth	8	1.24	0.31
XY position	0.7	-	0.7
Yaw	0.04	-	-

Table 4.1 – PID parameters.

4.3 Experimental results

In this section we report the results of the test-bed implementation of different kinds of scenarios.

4.3.1 Hover

This scenario consists in hovering the quadrotor, i.e. maintaining the agent in a fixed position in the 3D space. The experiment is meant to evaluate the performances on the controls of the different inputs, with particular regard to the throttle input, that revealed to be the most challenging to control. The controllers for pitch, roll and yaw begin to work as soon as the scenario is started and they aim to keep the agent flying above the reference xy position. A phase of take off makes the quadrotor lift by rapidly increasing the throttle input and when the agent reaches a height that is close enough to the hovering height, the PID controller for the throttle switches on. We show in figure 4.3 the 3D trajectory traced by the agent and in figures 4.4, 4.5 and 4.6 the position and the reference in the three earth-fixed axis. The hovering reference point is $q_d = (0, 0, 0.8)$ [m]. Moreover, we plot in figure 4.7 the yaw angle of the quadrotor during the hover experiment; its reference value is set to 0.

We summarize in table 4.2 the results of this experiment, in the form of root mean square and maximum errors after the take off phase.

Controlled value	RMS error	Max error
x axis	$65[mm]$	$168[mm]$
y axis	$123[mm]$	$-255[mm]$
z axis	$133[m]$	$-194[mm]$
yaw axis	$6.77[^\circ]$	$-9.2[^\circ]$

Table 4.2 – Hover experiment: error

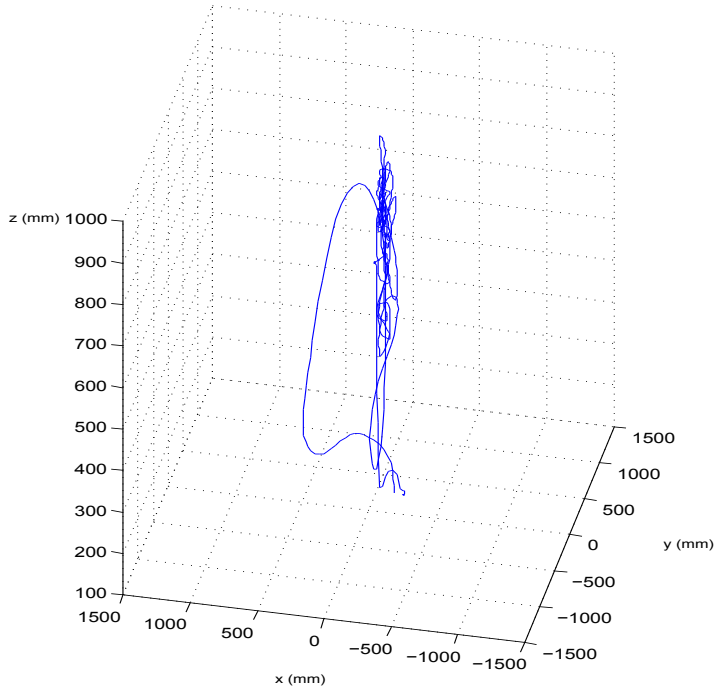


Figure 4.3 – Hover experiment: 3D trajectory of the quadrotor.

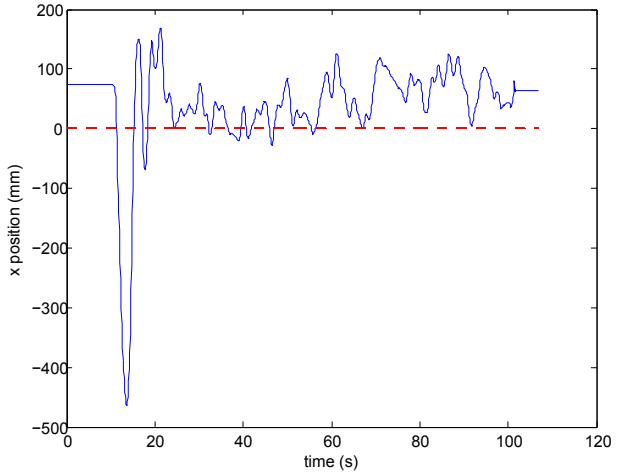


Figure 4.4 – Hover experiment: x position and reference

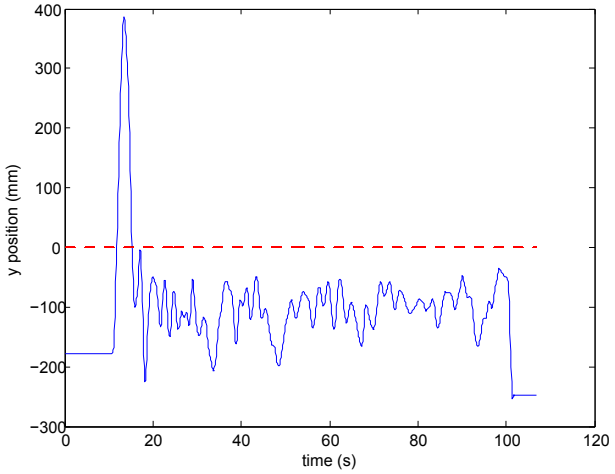


Figure 4.5 – Hover experiment: y position and reference

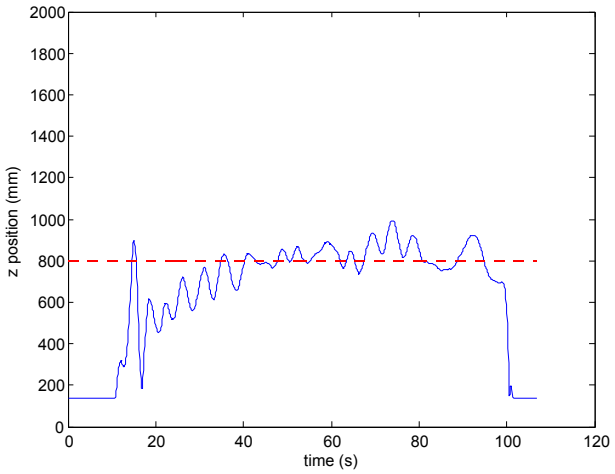


Figure 4.6 – Hover experiment: z position and reference

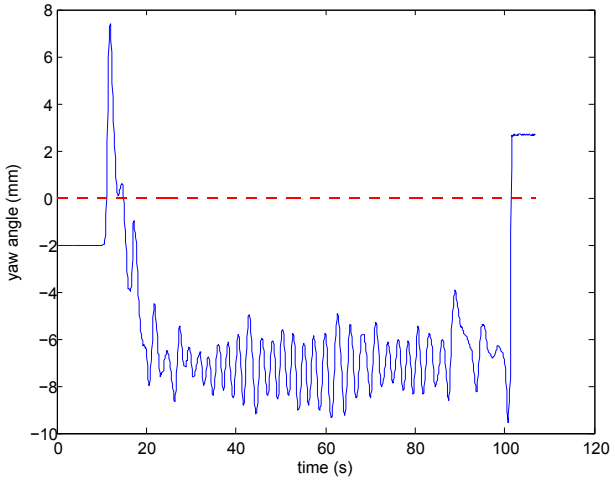


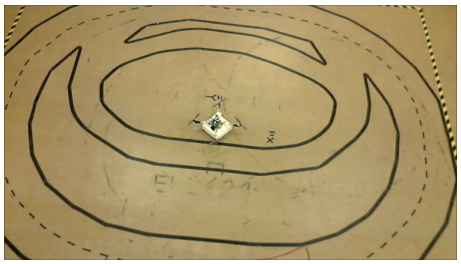
Figure 4.7 – Hover experiment: yaw angle and reference

We can state that the x-axis and y-axis controllers work well and the static error that is present in the x, y and yaw controllers are due to the lack of an integral part in the corresponding controllers. Since this static error has a value that does not exceed the $10[cm]$ for the x and y controllers and is about $7[°]$ for the yaw controller, it is satisfactory: adding an integral part slows down quite a lot the performances, in particular wrt the x and y controllers. The z position is the most difficult to control, since the input range of the throttle does not permit very smooth variations of the speed of the motors. This can be fixed in a future implementation by increasing the quantization of the throttle range. Anyway, after a settling time of $10 \div 15[s]$, the oscillations of the height around its reference does not exceed $\pm 20[cm]$ and for our purposes this value is acceptable, since for the obstacle avoidance we will consider larger safety radii. The transition between the takeoff phase and the controlled phase can be made smoother if we provide that the gap between the last value of the throttle during the takeoff and the base throttle of the height controller is null or sufficiently small.

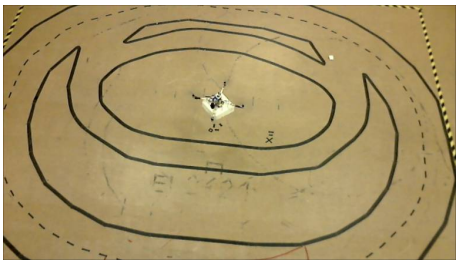
For this experiment and for each of the next ones, two videos have been recorded:

- The fist one is a top view from a point near the ceiling of the laboratory, recorded by a Logitech HD Webcam C310.
- The second one is a lateral view recorded by the camera of a Nokia Lumia 820 smartphone.

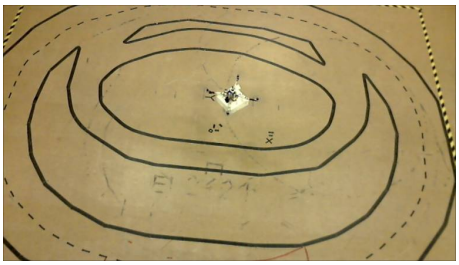
Snapshots of the videos captured for this experiment are reported in figures 4.8 and 4.9.



(a) Before takeoff.



(b) Takeoff.



(c) Hovering.

Figure 4.8 – Hover experiment, top view.

(a) Before takeoff.

(b) Takeoff.

(c) Hovering.

Figure 4.9 – Hover experiment, lateral view.

4.3.2 Circular path

The first experiment using waypoints is a circular path, centered in the origin of the earth reference system and with radius r = 1[m]. For this scenario the height is not controlled, i.e. a fixed throttle is applied to the quadrocopter and the aim of the experiment is to verify the correctness of the controllers for the movement along x and y when the reference changes.

The circular trajectory has been created and sampled in Matlab, obtaining 20 waypoints. The switch from one waypoint to the following can be performed manually by the user or by a time-based switch.

In the experiment first the vehicle is made hover above the starting point (1,0)[m], and once it starts the tracking the time switch activates with a period of 1000[ms] from one waypoint to the following. This is quite a reasonable time since our aim is not to have speed maneuver but rather precision. In the following pictures we show first the 2D path (reference vs. quadrotor position) and then the x and y positions individually (figures 4.10, 4.11, 4.12) of a three-laps long experiment.

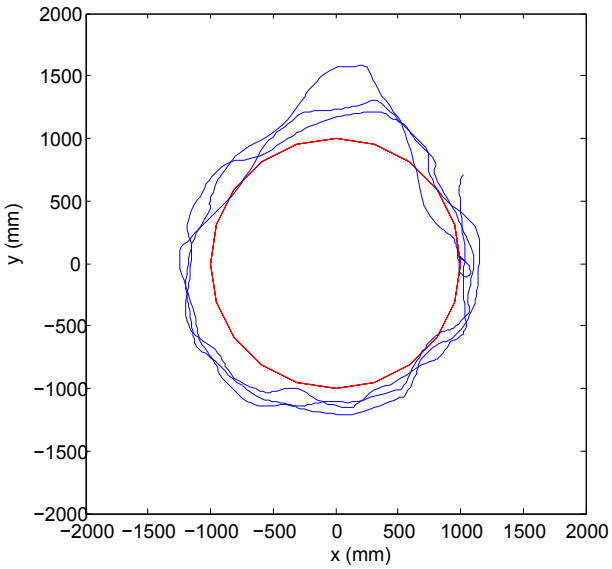


Figure 4.10 – Circular path experiment: xy position (blue) and reference (red)

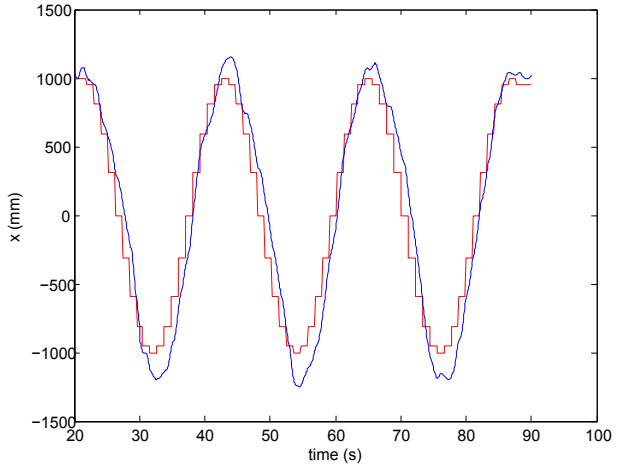


Figure 4.11 – Circular path experiment: x position (blue) and reference (red)

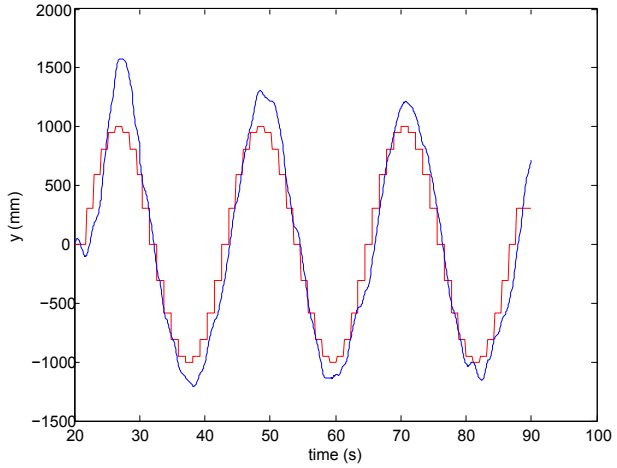


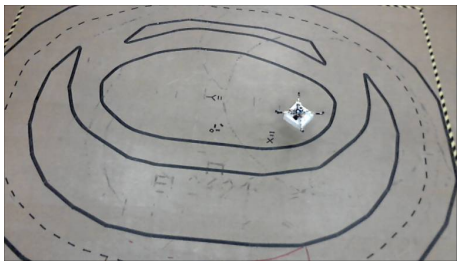
Figure 4.12 – Circular path experiment: y position (blue) and reference (red)

As we can observe, the path is tracked in a satisfactory manner since the error is almost always less that 20[cm], that is a value inside safety constraints if we consider that is about one fourth of the diameter of the quadrocopter itself. The major drift is along the roll axis and is about 50[cm]; this large error regards the first lap of the path and can be due to the fact that the quadrocopter has barely started its flight from the hovering position and has to accelerate from zero speed in the positive direction of the y axis. In the other points of the path and in the following laps the required acceleration is not as large, since the quadrotor is already flying in the desired direction.

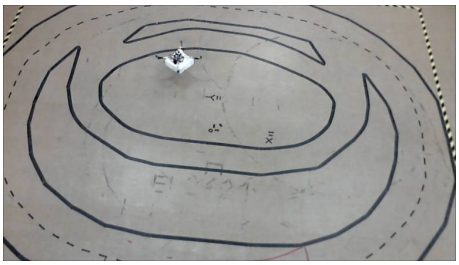
This experiment shows that even though the controller is able to make the

vehicle follow sinusoidal references of the x and y positions, attention must be paid to avoid collisions due to drifts of the quadrotor along the direction of the movement. This can be done applying a safety margin on the radii of the obstacles (see next subsection) or the quadrotor itself.

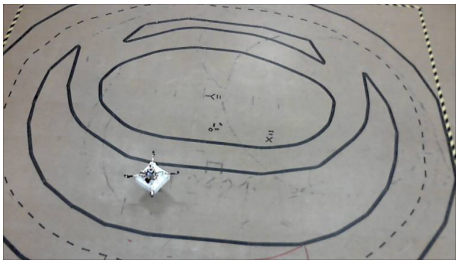
The snapshots of the cameras in three different positions along the circular path are depicted in figure 4.13 and 4.14.



(a) First snapshot.

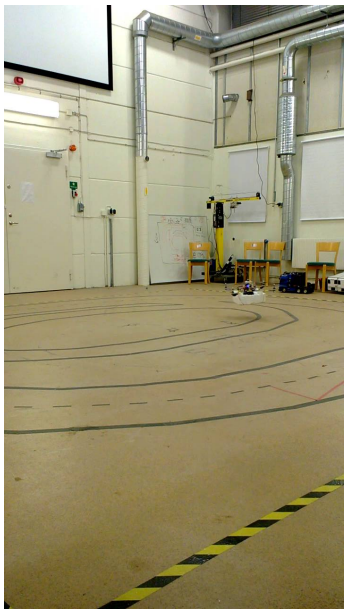


(b) Second snapshot.

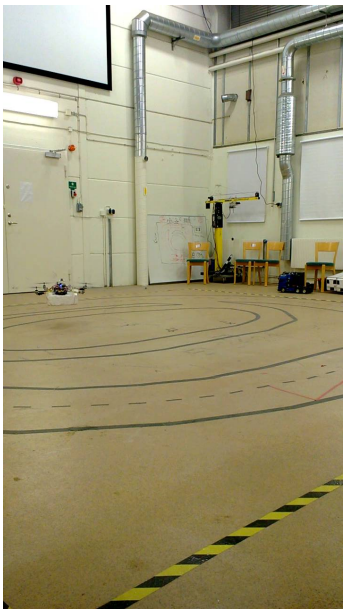


(c) Third snapshot.

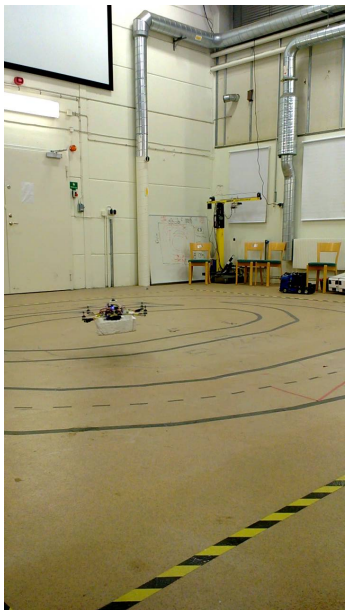
Figure 4.13 – Circular path experiment, top view.



(a) First snapshot.



(b) Second snapshot.



(c) Third snapshot.

Figure 4.14 – Circular path experiment, lateral view.

4.3.3 Obstacle avoidance path tracking

The main object of our control is collision-free navigation. The procedure we used in order to get a collision-free path exploits navigation functions to get the sequential positions that should be tracked. We shall summarize in the following lines the notion of navigation function and its utilization in autonomous navigation.

4.3.3.1 Navigation functions and their application to autonomous navigation

Once called E_n an n-dimensional Euclidean space, q the center of the position of the agent and q_d its destination, let us define a navigation function as follows:

Definition (**Koditscheck and Rimon [29]**) Let $\mathscr{F} \subset E_n$ be a compact connected analytic manifold with boundary. A map $\Phi : \mathscr{F} \to [0, 1]$, is a *navigation function* if it is:

1. Analytic on $\mathscr{F}$;
2. Polar on $\mathscr{F}$, with minimum at $q_d \in \mathscr{F}$;
3. Morse on $\mathscr{F}$, i.e. all its critical points are non-degenerate
4. Admissible on $\mathscr{F}$, i.e. $\lim_{q \to \partial\mathscr{F}} \Phi(q) = 1$.

Given the radius of the agent r, the destination q_d, the positions and radii of the M obstacles q_j and ρ_j, the center and radius of the circular workspace q_0 and ρ_0, a suitable navigation function is the one expressed as follows.

Let us define

$$\gamma_d(q) \triangleq ||q - q_d||^2$$

and

$$\gamma(q) \triangleq \gamma_d(q)^k,$$

with $k \in \mathbb{N}$ tuning parameter. $\gamma(q)$ is a measure of how close the agent is to its destination.

Let

$$\beta_j \triangleq ||q - q_j||^2 - (r + \rho_j)^2,$$

$$\beta_0 \triangleq (\rho_0 - r)^2 - ||q - q_0||^2$$

and

$$\beta = \prod_{j=0}^{M} \beta_j.$$

β_j and β_0 are indicators of how far the agent is from the obstacles and the workspace boundary.

A function with the form

$$\Phi = \left(\frac{\gamma}{\gamma + \beta} \right)^{\frac{1}{k}}$$

has the properties required by the definition of navigation function. In short, this function assumes high values in proximity of obstacles and workspace boundary and low values in proximity of the destination. In figure 4.15 we depict an example of navigation function for a bi-dimensional workspace of radius 15, an agent of radius 0.05 with destination $q_d = [7, 0]$, an obstacle of unitary radius in position $[0, -2]$ and $k = 2$.

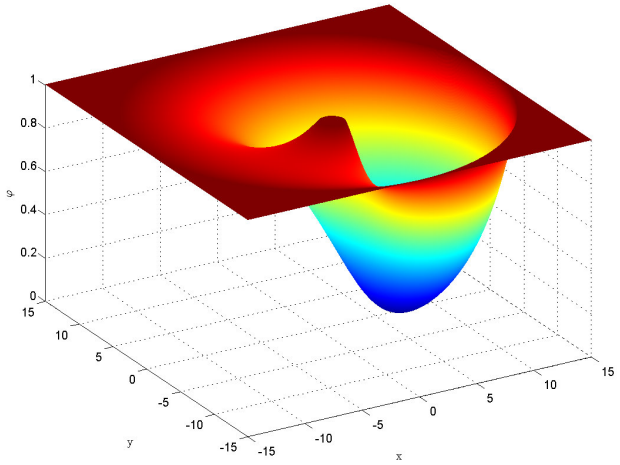


Figure 4.15 – Example of navigation function.

It is a known result that an agent with the kinematics of the form $\dot{q} = u$, i.e. an agent of which we can directly control the speed in any given direction, can be controlled with a law of the form $u = -K_u \nabla \Phi$ in order to obtain convergence and obstacle avoidance (see [29]). We will exploit this result to obtain a collision-free 3D path for our agent to follow, aware that the nonlinear second order dynamics of the quadrocopter do not allow a navigation function-based approach that

acts directly on its inputs and we cannot prove *a priori* that the quadrocopter can actually track in a proper way a path that is generated considering such a different dynamical model. Nevertheless, the results we will get are encouraging.

Seen these last considerations and in order to effectively obtain obstacle avoidance we payed attention to tune the parameters of the scenario and the navigation function in such a way that we could rely on safety margins wrt the distance from the points of the path to the obstacles. This has been done acting

- on the radius of the agent and the obstacles: the longer the radii, the more reliable the distance constraint that provides collision-free navigation;
- on the gain k of the navigation function: a high value of k implies a path closer to the obstacles, and vice versa.

The procedure we follow for obstacle avoidance navigation in our testbed is the following:

- Get from the motion capture system or manually set the initial position, destination and obstacles position in the workspace.
- Translate this coordinates to a simulated environment in Matlab and get the collision-free path of a single-integrator omnidirectional agent.
- Take back the obtained path to the workspace coordinate system
- Sample the path to get a suitable number of waypoints to track. In the following experiments the path has been sampled in such a way that the euclidean distance from one waypoint to the following is 20[cm]. This value has been chosen after some trials as a trade-off between an easily manageable number of waypoints and a good matching between the continuous[5] path computed by Matlab and the path we get connecting the sampled waypoints.
- Make the quadrocopter hover on the starting point of the path and, when a switch is commuted, start the tracking of the waypoints. As we said, the switch from one waypoint to another can be done manually, basing on a fixed-step timer but also using a rule on the time the agent stays in a

[5] Of course the path computed by Matlab is not literally continuous, but is composed of a number of points more than one order of magnitude greater than the sampled ones. For instance, we set the Matlab solver to integrate along a 500 time samples interval, so the output path is composed by 500 points. The number of waypoints we get for the experiments, on the other hand, is always below 30.

neighborhood of the current waypoint. Let us report the pseudocode for the last case:

```
time_on_position = 0
for each iteration of the control algorithm do
  c_d ← distance between agent's position and current waypoint
  if c_d ≤ d_neighborhood AND time_on_position > max_time_on_position
  then
    switch to next waypoint
    time_on_position = 0
  else if c_d ≤ d_neighborhood then
    time_on_position = time_on_position + Δt
  end if
end for
```

- Once the quadrocopter reaches its destination and if the space below is clear from obstacles, land.

The implementation of the waypoint tracking has been done using a simple state machine. The switching between its states can be either automatic or manual; anyway even if the automatic switching is used we should always be ready to intervene with the manual control in order to avoid major crashes or to stop the experiment for any reason. In figure 4.16 we visualize the states of the quadrocopter and the possible transitions.

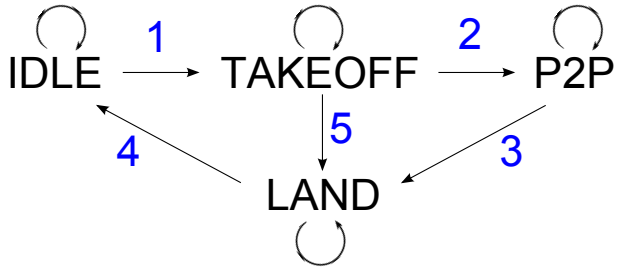


Figure 4.16 – State machine of the quadrotor.

The names of the states are self-explaining: Idle, Takeoff, Point to point navigation (P2P), Land. Each number indicates a condition for the corresponding transition to happen:

1. Takeoff button switched ON *AND* Land button OFF;

2. P2P button switched ON *OR* distance from hovering position reached;

3. Land button switched ON *OR* last waypoint visited;

4. Throttle input = 0 *AND* Takeoff button OFF;

5. Land button switched ON.

Please note that the pitch, roll and yaw controllers are always active during the flights, whereas the PID throttle controller is active only in the P2P state: during the takeoff the throttle is quickly increased from the byte value of 45 in order to get the agent in air and in the landing state the throttle is quickly decreased and is cut when either it reaches the value 40 or the agent reaches the height of 400[mm].

4.3.3.2 Experiment A: towers

We now present the experiments that have been conducted for obstacle avoidance scenarios; we remark that in the collision-free path calculation we assume that the obstacles are spherical and can float in the 3D space. On the other hand, the obstacles we placed in the testbed have their base on the ground and their top is about at the position where the spherical obstacles would be. So before proceeding with the scenarios we needed to pay attention that the path computed for the spherical obstacles is still collision-free and safely distant from the obstacle bodies. In these test our aim is not to present an aggressive controller nor to track a speed reference, but rather check the performances in path following and the success in obstacle avoidance. For this reason we decided to keep using a time-based switch for the waypoints, with period 1000[ms].

The first scenario consists in three obstacles of different height: two high ones on the left of the agent facing the destination at 1.67[m] and 1.83[m] and a small one on the right at 0.5[m]. The path provided by the four dimensional navigation function passes between the left obstacles and the right one and first raises up over the height of the two taller obstacles and the settles down to the destination point, situated behind the tallest obstacles at 1.825[m] of height.

In figures 4.17 and 4.18 we see the trajectory of the quadrocopter in blue and the collision-free path computed by Matlab (before the sampling) in red. We can state that the collision avoidance goal is reached, and the XY position is tracked pretty well: the maximum error we get is about 20[cm]. On the other hand, the height controller is really precise in the first part of the trajectory and gets a bump towards the end, before the quadrocopter starts to hover around the destination point, of about 30[cm]. These values for the error are definitely satisfactory, if we consider the fact that the vertical extension of the agent is about 25[cm] so we can still rely on the safety margins given by the spherical approximation of the agent itself.

Snapshots of the flight for experiment A are depicted in figures 4.19 and 4.20.

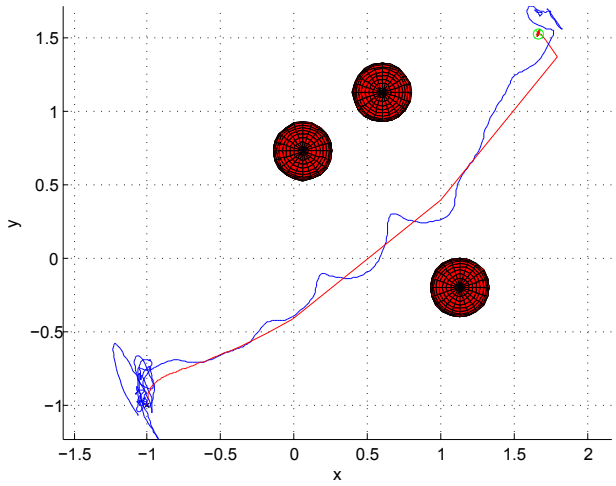


Figure 4.17 – Experiment A, top view: xy position (blue), xy reference (red)

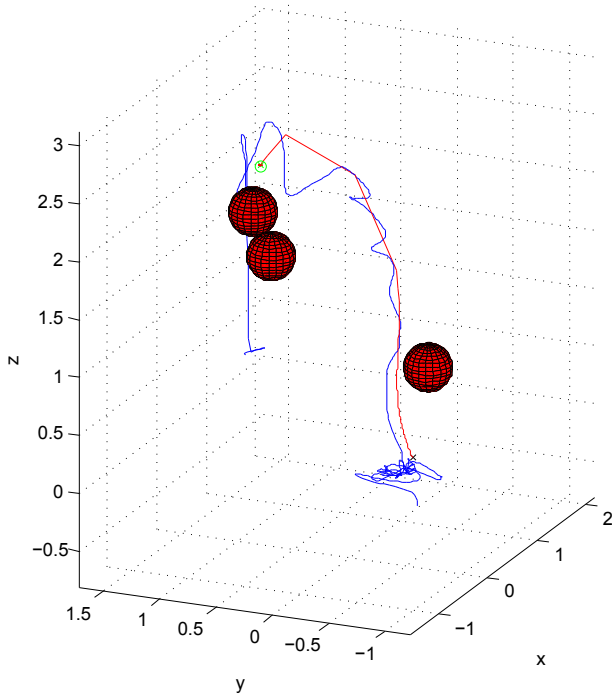
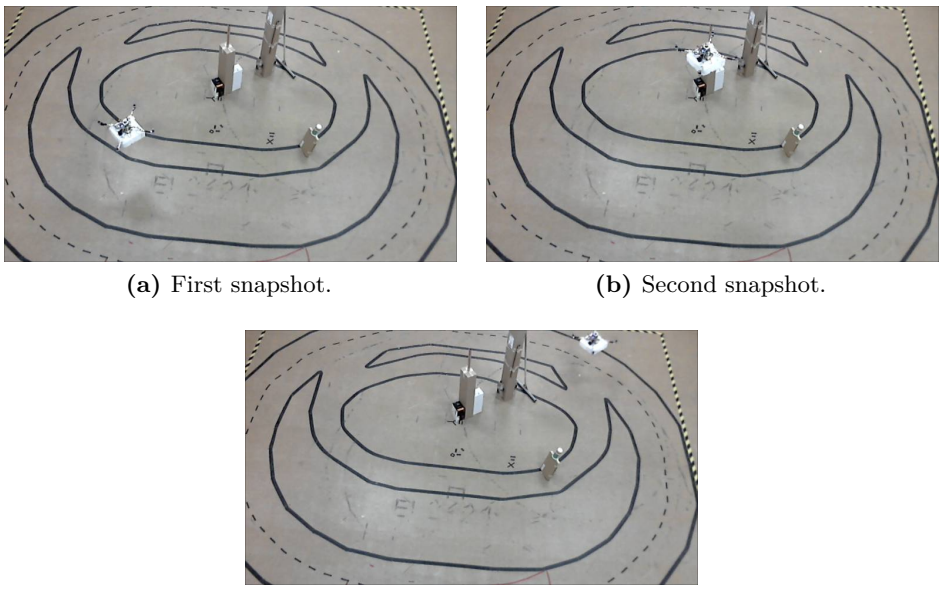


Figure 4.18 – Experiment A, 3D view: xy position (blue), xy reference (red)

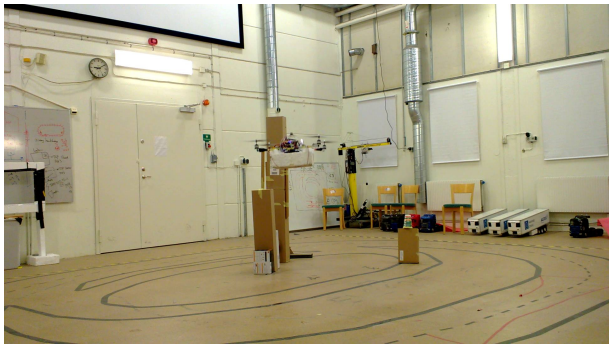


(a) First snapshot.

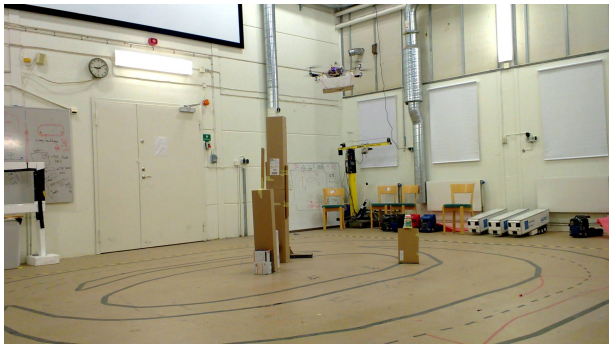
(b) Second snapshot.

(c) Third snapshot.

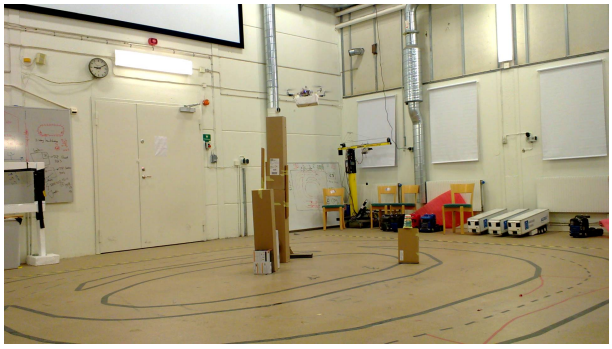
Figure 4.19 – Towers experiment (A), top view.



(a) First snapshot.



(b) Second snapshot.



(c) Third snapshot.

Figure 4.20 – Towers experiment (A), lateral view.

4.3.3.3 Experiment B: pillar

The second scenario also consists in three obstacles of different height: two low ones (0.6[m]) in front of the quadrocopter facing the destination and a taller one (1.1[m]) behind the first two. This setting has been designed *ad hoc* in order to see the performance of the agent avoiding the first two obstacles from above and the third one from aside.

In figures 4.21 and 4.22 we see the trajectory of the quadrocopter in blue and the collision-free path computed by Matlab (before the sampling) in red. Similarly to the previous case, here the position along the x and y axes is tracked in a very satisfactory manner, the maximum error being approximately 25[cm], due to a pretty abrupt change of direction towards the end of the path, when the agent flies towards its destination behind the tall obstacle. Here the height controller works decently as well, presenting the most significant errors at the beginning (20[cm]) right when it has to avoid the first obstacles from above (once again a proper margin on the distance between the path and the obstacles in the design phase has avoided collision danger) and towards the end (25[cm]), in conjunction with the XY error. The last part of the trajectory is thus the most critical, and we must be aware of the error we observe when the direction of the motion is changed quite abruptly.

Snapshots of the flight for experiment B are depicted in figures 4.23 and 4.24.

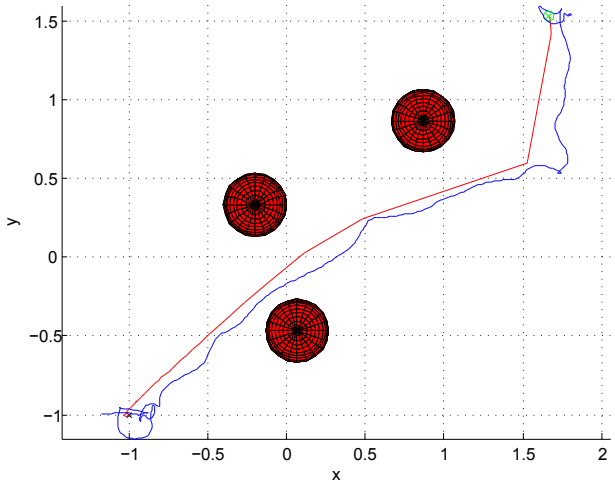


Figure 4.21 – Experiment B, top view: xy position (blue), xy reference (red)

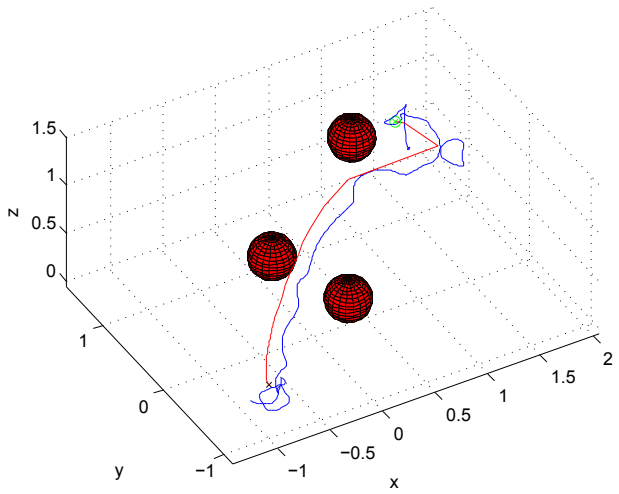
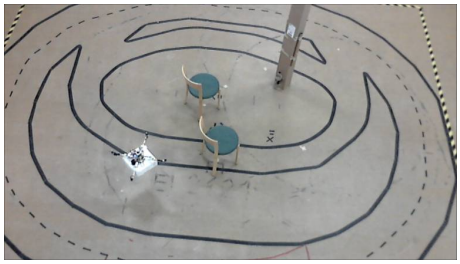
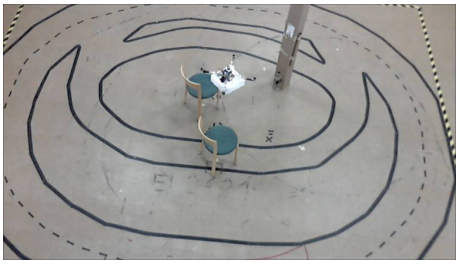


Figure 4.22 – Experiment B, 3D view: xy position (blue), xy reference (red)



(a) First snapshot.

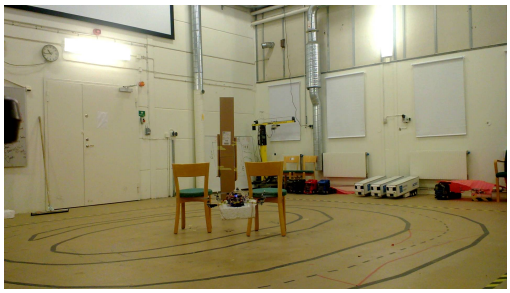


(b) Second snapshot.

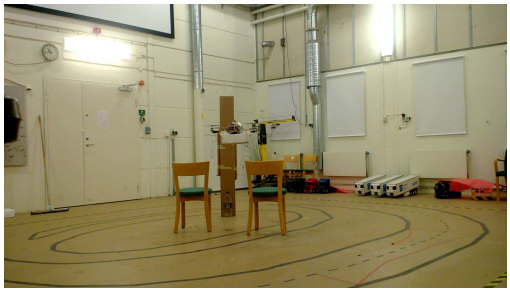
(c) Third snapshot.

(d) Fourth snapshot.

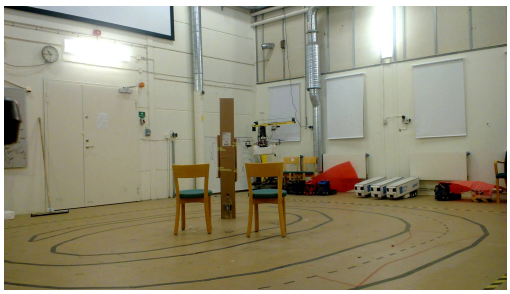
Figure 4.23 – Pillar experiment (B), top view.



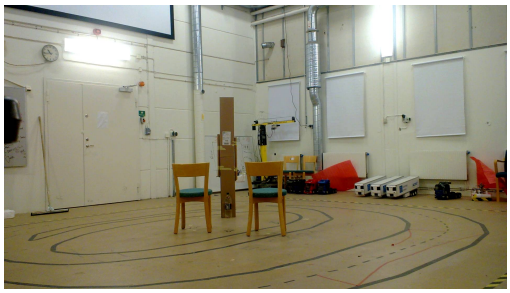
(a) First snapshot.



(b) Second snapshot.



(c) Third snapshot.



(d) Fourth snapshot.

Figure 4.24 – Pillar experiment (B), lateral view.

4.3.3.4 Experiment C: slalom

The third experiment for collision avoidance differs from the other two since in this case we decided to put one more obstacle in the workspace and keep all the obstacles at the same height (0.8[m]); the path that results is bi-dimensional (i.e. we can suppose to exploit a three dimensional navigation function to obtain it), so the challenge is here having good performances in response only to abrupt variations of the direction of the motion in the XY plane, while the vertical reference is kept constant at 0.8[m].

We plot in figure 4.25 the the trajectory of the quadrocopter in blue and the collision-free path computed by Matlab (before the sampling) in red, viewed from above. In this last scenario, the tracking of the XY position is good, as a matter of fact even thought the gaps between the obstacles in which the quadrotor must pass are quite narrow wrt the agent dimension, the collision avoidance is still achieved and the maximum tracking error in this case is around 10[cm], if we neglect the very final section in which the agent stops quite abruptly since it has reached its destination, provoking an error that is anyway little more than 20[cm]. It is interesting to note also the performances of the height controller (figure 4.26): its performance is decent, even though the oscillating behavior is clear but never exceeds 20[cm] after the initial take off.

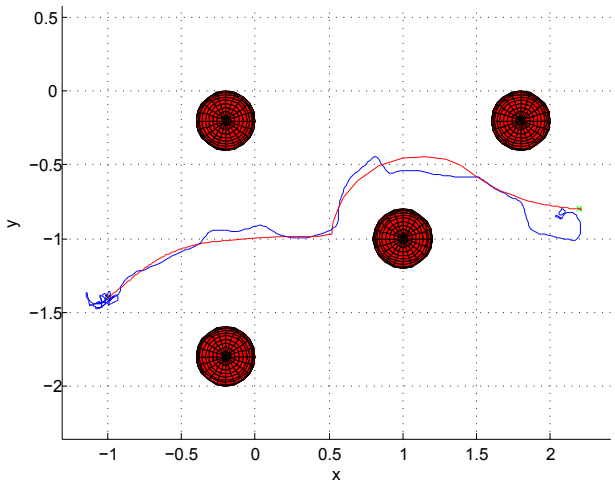


Figure 4.25 – Experiment C, top view: xy position (blue), xy reference (red)

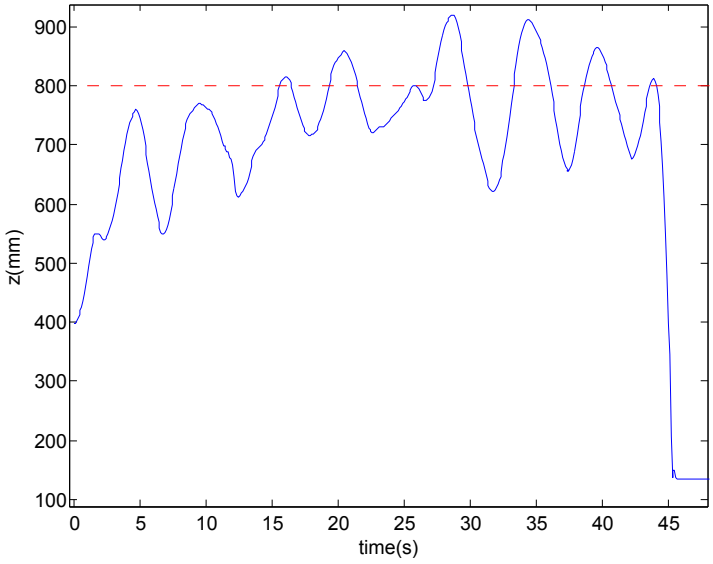
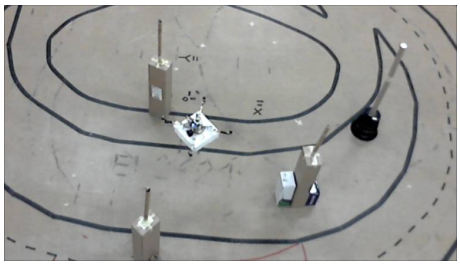


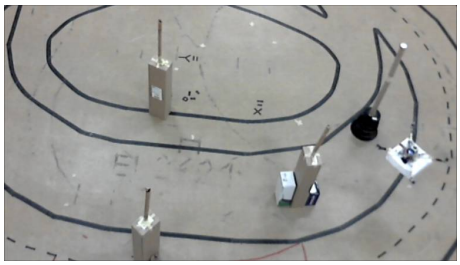
Figure 4.26 – Experiment C: height position (blue) and reference(red).

Here we report snapshots of the flight for experiment C (figures 4.27 and 4.28).



(a) First snapshot.

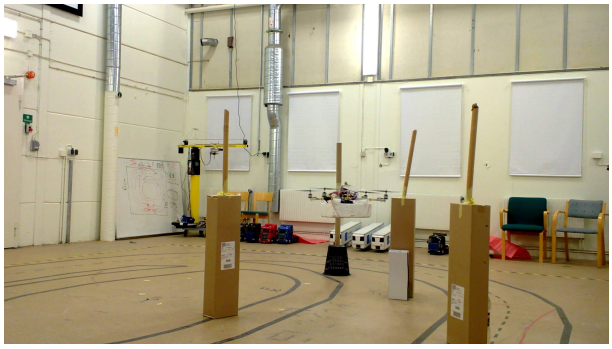
(b) Second snapshot.



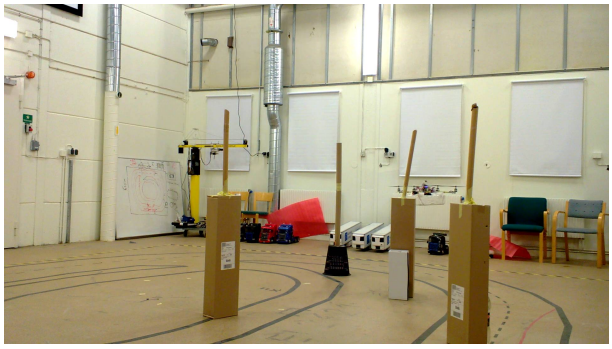
(c) Third snapshot.

Figure 4.27 – Slalom experiment (C), top view.

(a) First snapshot.



(b) Second snapshot.



(c) Third snapshot.

Figure 4.28 – Slalom experiment (C), lateral view.

Chapter 5

Autonomous navigation of an air vehicle

In the previous chapter we stated that the heuristic controller we presented allows path tracking with pretty satisfactory results. However, flying vehicles have usually more complicated dynamics than the one of the quadrocopter, in the sense that their structure allows more constrained movements. It is interesting to analyze this kind of air vehicles and how an algorithm for stabilization and obstacle avoidance can be implemented on them. As we already stated in the Introduction, the controlled that has been presented in chapter 4 relies on considerations on the features of the hardware that we have at our disposal; on the other hand, the aim of the current chapter is to depict a more general vehicle in order to make theory as general as possible and be able to handle a more constrained dynamic agent in future applications in the framework.

For this purpose, let us consider the following model, that consists in a flying vehicle governed by four inputs, one for the longitudinal velocity and one for each of the angular velocities around the axis of the vehicle (see also [30]). The control that will be presented is an extension to the three-dimensional case of the controller designed in [31] and in the proof we will exploit some useful results that are shown in that work.

5.1 Air vehicle model

Here follows the kinematic model for an air vehicle moving in a three dimensional space.

Let q be the state of the agent:

$$q = \left[\begin{array}{c} q_1 \\ \hline q_2 \end{array}\right] = \left[\begin{array}{c} x \\ y \\ z \\ \hline \phi \\ \theta \\ \psi \end{array}\right] \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \text{position} \\ \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \text{orientation} \end{array} \tag{5.1}$$

where the orientation is expressed via the Euler angles $[\ \phi \quad \theta \quad \psi\ ]^T$, that we assume restricted to the limits $\phi \in (-\pi, \pi]$, $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2}]$, $\psi \in (-\pi, \pi]$. Both linear and angular positions are referred to an *Earth-fixed* coordinate system.

Let the motion of the agent be described by

$$\dot{q} = \left[\begin{array}{c} \dot{q}_1 \\ \dot{q}_2 \end{array}\right] \triangleq \tau = \left[\begin{array}{c} \tau_1 \\ \tau_2 \end{array}\right] \tag{5.2}$$

where τ_1 and τ_2 define the linear and angular velocities, respectively.

Now we want to express the motion of the vehicle in *Body-fixed* coordinates. Let

$$r = \left[\begin{array}{c} l \\ \hline a \end{array}\right] = \left[\begin{array}{c} l_1 \\ l_2 \\ l_3 \\ \hline a_1 \\ a_2 \\ a_3 \end{array}\right]$$

be the vector containing linear and angular positions of the agent in the *Body-fixed* system and let us state that the directions of l_1, l_2 and l_3 are as in figure 5.1.

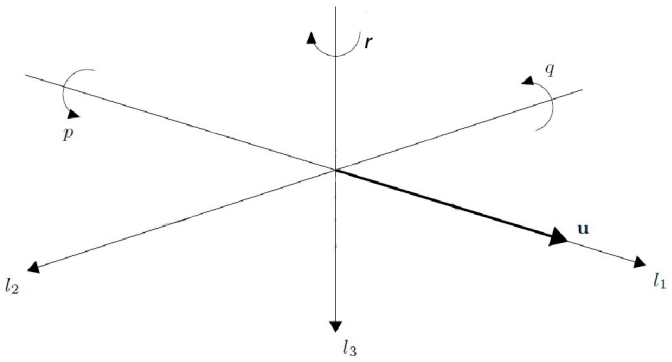


Figure 5.1 – *Body-fixed* coordinates

The corresponding velocities are

$$v = \left[\begin{array}{c} v_1 \\ \hline v_2 \end{array}\right] = \left[\begin{array}{c} u \\ v \\ w \\ \hline p \\ q \\ r \end{array}\right].$$

The transformation between *body-fixed* and *earth-fixed* coordinates is the following:

$$\begin{aligned} \dot{q}_1 &= \tau_1 = J_1(q_2) \cdot v_1 \\ \dot{q}_2 &= \tau_2 = J_2(q_2) \cdot v_2 \end{aligned} \tag{5.3}$$

where

$$\begin{aligned} J_1 &= \begin{bmatrix} c_\psi c_\theta & -s_\psi c_\phi + c_\psi s_\theta s_\phi & s_\psi s_\phi + c_\psi c_\phi s_\theta \\ s_\psi c_\theta & c_\psi c_\phi + s_\phi s_\theta s_\psi & -c_\psi s_\phi + s_\theta s_\psi c_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \in SO(3) \\ J_2 &= \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & \frac{s_\phi}{c_\theta} & \frac{c_\phi}{c_\theta} \end{bmatrix} \end{aligned} \tag{5.4}$$

where we shortened the trigonometric functions $\sin(\cdot)$, $\cos(\cdot)$ and $\tan(\cdot)$ with s, c and t.

The inputs of the system are the longitudinal linear velocity u in the *body-fixed*

reference system and the angular velocities in the *earth-fixed* reference system. Let us gather them in the input vector

$$v_k = [\ u \quad \omega_1 \quad \omega_2 \quad \omega_3\ ]^T.$$

According to equations (5.3) we can derive the complete model that we are going to consider:

$$\dot{q} = \tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \underbrace{\begin{bmatrix} \bar{J} & 0_{3\times 3} \\ 0_{3\times 1} & I_3 \end{bmatrix}}_{\triangleq R \in \mathbb{R}^{6\times 4}} v_k \triangleq R(n_2) \cdot v_k \tag{5.5}$$

where $\bar{J} \triangleq [\ c_\psi c_\theta \quad s_\psi c_\theta \quad -s_\theta\ ]^T$ is the first column of J_1.

Let us rewrite the previous matrix expression in form of system for the sake of clarity and depict it in figure 5.2

$$\begin{cases} \dot{x} = \cos\psi \cos\theta\, u \\ \dot{y} = \sin\psi \cos\theta\, u \\ \dot{z} = -\sin\theta\, u \\ \dot{\phi} = \omega_1 \\ \dot{\theta} = \omega_2 \\ \dot{\psi} = \omega_3 \end{cases} \tag{5.6}$$

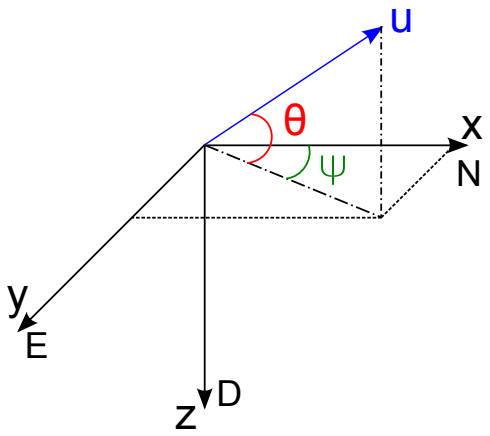


Figure 5.2 – Visualization of the axes, velocity vector and angles of interest.

5.2 Control approach

Let us consider a spherical workspace $\mathcal{W} \in \mathbb{R}^3$ of radius ρ_0, in which N_o spherical obstacles of radii $\rho_1, \ldots, \rho_{N_o}$ and in positions $q_{o_1}, \ldots, q_{o_{N_o}}$ stand. The radius of the agent is taken large enough to guarantee the mentioned safety requirement. The problem we consider is designing a control law to steer the agent described in the previous section from an initial configuration q_0 to a desired configuration q_d, avoiding collisions with the obstacles.

In order to reach our goal, we will exploit again a navigation function-based approach and we will add a model predictive integration to satisfy desired performance requirements. Regarding the navigation functions, we will now exploit *dipolar navigation functions*, in order for the agent to reach the destination with the desired orientation, avoiding in-place rotations.

5.3 Dipolar navigation functions

Dipolar navigation functions (DNF) are exploited in order to drive the agent to destination with the desired orientation (see [32] and [30]). This is done by creating a navigation function such that the integral lines of its potential field are all tangent to the desired orientation at the destination. This is achieved by considering the plane of which the normal vector is parallel to the desired orientation, and includes the destination, as an additional artificial obstacle. Namely, the *obstacle function* β described in section 4.3.3.1 is pre-multiplied by the contribution

$$H_{nh} = \epsilon_{nh} + n_{nh}$$

with ϵ_{nh} small positive constant and

$$n_{nh} = ||\bar{J}(q_{2d}) \cdot (q_1 - q_{1d})||^2$$

where q_{1d} and q_{2d} are the desired position and orientation at destination.

In figure 5.3 we report an example of a three-dimensional DNF (i.e. in a two-dimensional workspace) as presented in [30].

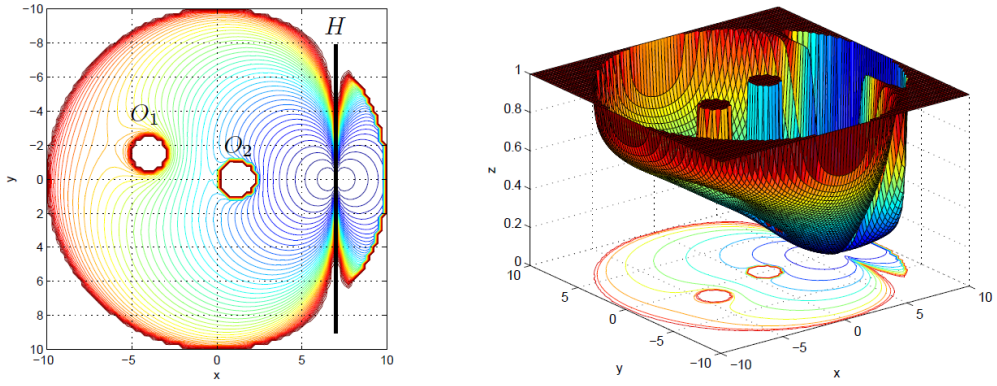


Figure 5.3 – Dipolar navigation function with two obstacles (image from [30]); the nonholonomic obstacle is the line $x = 7$.

The control law presented in [30] exploits DNF for 3D navigation and has the following structure:

$$\begin{aligned} u &= -\text{sgn}(\bar{J}\frac{\partial \Phi}{\partial q_1})F(q_1) \\ \omega_1 &= -k_\phi(\phi - \phi_{\mathbf{nh}}) \\ \omega_2 &= -k_\theta(\theta - \theta_{\mathbf{nh}}) \\ \omega_3 &= -k_\psi(\psi - \psi_{\mathbf{nh}}) \end{aligned} \tag{5.7}$$

where Φ is the dipolar navigation function, F is a function that regulates the magnitude of the linear velocity i.e. $F = k_u||\frac{\partial \Phi}{\partial q_1}||^2 + k_z||q_1||^2$, $k_\phi, k_\theta, k_\psi, k_u, k_z$ are positive gains and $\phi_{\mathbf{nh}}$, $\theta_{\mathbf{nh}}$, $\psi_{\mathbf{nh}}$ are the nonholonomic angles, described in the following.

Let us define the function

$$\text{atan2}(y, x) \triangleq arg(x, y), \quad (x, y) \in \mathbb{C}$$

and the partial derivatives

$$\Phi_x \triangleq \frac{\partial \Phi}{\partial x} \quad \Phi_y \triangleq \frac{\partial \Phi}{\partial y} \quad \Phi_z \triangleq \frac{\partial \Phi}{\partial z}$$

The nonholonomic angles, i.e. the angles defined by the gradient of the

dipolar navigation function, are:

$$\begin{aligned}\psi_{\mathbf{nh}} &= \operatorname{atan2}\left(\operatorname{sign}(d)\Phi_y, \operatorname{sign}(d)\Phi_x\right)\\ \theta_{\mathbf{nh}} &= \operatorname{atan2}\left(-\operatorname{sign}(d)\Phi_z, \sqrt{\Phi_x^2+\Phi_y^2}\right)\\ \phi_{\mathbf{nh}} &= \operatorname{atan2}\left(\operatorname{sign}(d)\,\cos\theta\,\omega_\psi, \operatorname{sign}(d)\,\omega_\theta\right)\end{aligned} \tag{5.8}$$

where the vector $\operatorname{sign}(d)\nabla\Phi$ is the direction that the longitudinal axis l_1 steers to align with and the angles $\psi_{\mathbf{nh}}$ and $\theta_{\mathbf{nh}}$ represent respectively azimuth and elevation of the vector. So according to sign of d the aircraft must approach the target moving forward (i.e. it steers towards the direction of $-\nabla\Phi$), or backwards (i.e. it steers toward the direction of $\nabla\Phi$).

Since the above-defined angles are not continuous at the destination position (where $\nabla\Phi$ is null), the approximation scheme in [30] is employed, that guarantees to have $\phi_{\mathbf{nh}} = \phi_d$, $\theta_{\mathbf{nh}} = \theta_d$ and $\psi_{\mathbf{nh}} = \psi_d$ at the destination.

The bank angle ϕ does not influence directly the motion of the aircraft and so the goal of the nonholonomic angle $\phi_{\mathbf{nh}}$ is to track the reference bank angle so that the agent tends to eliminate the yaw rate and achieve the required alignment only through pitch rotation.

5.4 Model predictive control

Generally speaking, Model Predictive Control (MPC) consists in an iterative finite horizon optimization, conducted with regards to a performance measure. We would like our agent to reach the destination point and avoid obstacles and at the same time minimize a functional, that can be related to the control effort, to input constraints or other parameters we are interested to keep monitored. Let us call T_p the prediction time horizon; we want to minimize a cost functional in the form:

$$J(t,x,u,T_p) = \underbrace{\int_t^{(t+T_p)} \Lambda(q(\tau),u(\tau))d\tau}_{\text{Running cost}} + \underbrace{M(q(t+T_p))}_{\text{Terminal cost}} \tag{5.9}$$

At each calculation time t, the state of the system is sampled and a cost minimizing control law is computed for a time horizon $[t, t+T_p)$. The control law obtained with this minimization is applied during a shorter period of time $T_c < T_p$ and the minimization problem is solved again for the new state $q(t+T_c)$.

The terminal cost M represents an approximation of the cost-to-go from $t + T_p \to \infty$: such a function can be well represented by a navigation function Φ (see [31]), so the cost functional becomes

$$J(t, x, u, T_p) = \int_t^{(t+T_p)} \Lambda(q(\tau), u(\tau))d\tau + \Phi(p(t + T_p)) \tag{5.10}$$

where we indicate with p the linear position vector of the agent.

Now, we want to focus on model predictive navigation, i.e. how MPC can be applied to the model we are dealing with. In order to do so, let us consider that a control law based only on a navigation function approach does not allow us to choose one trajectory over another, so we need to introduce some *deviations* from the direction of the DNF gradient in order to have some values we can act on to minimize the index. We define those deviations in terms of variations of the angles θ and ψ in respect to the value set by the DNF control (5.7): let us call them $\bar{\delta\theta}$ and $\bar{\delta\psi}$ and let us impose that for each prediction interval the deviations are first order polynomials (i.e. straight lines) with the previous value of the deviation as starting point and the candidate value for the current deviation as ending point[1]. For each prediction interval $[t_k, t_k + T_p)$ the minimization problem to be solved is

$$J^*(t_k, x, T_p) = \min_{\substack{\bar{\delta\theta}([t_k, t_k + T_p)), \\ \bar{\delta\psi}([t_k, t_k + T_p))}} J(t_k, x, u, T_p)$$

and the optimal values of the deviations for each prediction interval are the couple of deviations that minimize the index when applied together

$$\begin{bmatrix} \delta\theta \\ \delta\psi \end{bmatrix} \triangleq \operatorname*{argmin}_{\substack{\bar{\delta\theta}([t_k, t_k + T_p)), \\ \bar{\delta\psi}([t_k, t_k + T_p))}} J^*(t_k, x, T_p)$$

[1] We make this assumption in order to keep the value of the deviation bounded during the prediction interval, a requirement that will be exploited in the upcoming demonstration of stability (see also [31])

5.5 Proposed control

The control law that we propose comes from the combination of the input given by the navigation-function approach and the deviations given by the model predictive control and is the following:

$$\begin{aligned} u &= -\operatorname{sign}\left(\bar{J}\frac{\partial\Phi}{\partial q_1}\right) k_u \left|\left|\frac{\partial\Phi}{\partial q_1}\right|\right|^2 \\ \omega_\phi &= -k_\phi(\phi - \phi_{\mathbf{nh}}) + \dot{\phi}_{\mathbf{nh}} \\ \omega_\theta &= -k_\theta(\theta - \theta_{\mathbf{nh}} - \delta\theta) + \dot{\theta}_{\mathbf{nh}} + \dot{\delta\theta} \\ \omega_\psi &= -k_\psi(\psi - \psi_{\mathbf{nh}} - \delta\psi) + \dot{\psi}_{\mathbf{nh}} + \dot{\delta\psi} \end{aligned} \tag{5.11}$$

where:

- Φ is the dipolar navigation function (DNF);
- k_u, k_ϕ, k_θ and k_ψ are positive constant control parameters;
- $\phi_{\mathbf{nh}}$, $\theta_{\mathbf{nh}}$ and $\psi_{\mathbf{nh}}$ are the nonholonomic angles;
- $\delta\theta$ and $\delta\psi$ are the optimal deviations provided by the model predictive algorithm.

We recall from section 5.4 that the deviations provided by the MP algorithm are calculated at every prediction interval $[t, t+T_p)$ and applied during a shorter time slot called control interval $[t, t+T_c)$. At every control interval the deviations are first order polynomials

$$\begin{aligned} \delta\theta(\tau) &= (1 - \frac{\tau}{T_c})\delta\theta(t) + (\frac{\tau}{T_c})\delta\theta(t+T_c) \qquad \tau \in [t, t+T_c) \\ \delta\psi(\tau) &= (1 - \frac{\tau}{T_c})\delta\psi(t) + (\frac{\tau}{T_c})\delta\psi(t+T_c) \qquad \tau \in [t, t+T_c) \end{aligned}$$

and their concatenation is a continuous function by construction.

Moreover, the model predictive control is turned off as we reach a neighborhood of radius r_{nb} of the destination q_{1d}, and we set:

$$\begin{aligned} \delta\theta(t \geq t_f) &= \frac{||q_1 - q_{1d}||^2}{r_{nb}}\delta\theta(t_f) \\ \delta\psi(t \geq t_f) &= \frac{||q_1 - q_{1d}||^2}{r_{nb}}\delta\psi(t_f), \end{aligned} \tag{5.12}$$

with $t_f = \inf\{t : ||q_1 - q_{1d}|| \leq r_{nb}\}$. These terms are continuous in $||q_1 - q_{1d}|| = r_{nb}$ and are null with null derivative at the destination point:

$$\begin{aligned} &\delta\theta(||q_1 - q_{1d}|| = r_{nb}) = \delta\theta(t_f) \\ &\delta\theta(q_{1d}) = 0 \\ &\dot{\delta\theta}(q_{1d}) = 0 \end{aligned} \tag{5.13}$$

Similar expressions can be found for $\delta\psi(t \geq t_f)$.

5.5.1 Useful lemmas

We report here two lemmas we are going to use in the demonstration. They have been used for demonstrating the two-dimensional case of the problem in [31] and their demonstration can be found in the same work.

Lemma 5.5.1 *The deviations $\delta\theta$ and $\delta\psi$ are continuous functions and*

$$|\delta\theta| < \frac{\pi}{2}, \quad |\delta\psi| < \frac{\pi}{2} \quad \forall t$$

Lemma 5.5.2 *At every recalculation time t_k, let us modify the sampling set for the deviations, that originally were $\Theta = \Psi = (-\frac{\pi}{2}, \frac{\pi}{2})$, as follows:*

$$\Theta_k = \left(-\frac{\pi}{2} + |\theta(t_k) - \theta_{\mathbf{nh}}(t_k) - \delta\theta(t_k)|, \frac{\pi}{2} - |\theta(t_k) - \theta_{\mathbf{nh}}(t_k) - \delta\theta(t_k)|\right)$$

$$\Psi_k = \left(-\frac{\pi}{2} + |\psi(t_k) - \psi_{\mathbf{nh}}(t_k) - \delta\psi(t_k)|, \frac{\pi}{2} - |\psi(t_k) - \psi_{\mathbf{nh}}(t_k) - \delta\psi(t_k)|\right)$$

By doing so, if at the beginning $|\theta(0) - \theta_{\mathbf{nh}}(0)| < \frac{\pi}{2}$ and $|\psi(0) - \psi_{\mathbf{nh}}(0)| < \frac{\pi}{2}$, then

$$|\theta(t) - \theta_{\mathbf{nh}}(t)| < \frac{\pi}{2}, \quad |\psi(t) - \psi_{\mathbf{nh}}(t)| < \frac{\pi}{2} \quad \forall t$$

We remark that the hypothesis $|\theta(0) - \theta_{\mathbf{nh}}(0)| < \frac{\pi}{2}$ and $|\psi(0) - \psi_{\mathbf{nh}}(0)| < \frac{\pi}{2}$ are reasonable, in particular if we assume that the agent can perform maneuvering before take off.

5.5.2 Proof of convergence and collision avoidance

We can finally state the following

Theorem 5.5.3 (Convergence and collision avoidance) *An agent described by the model (5.5) navigating under the control law (5.11) converges to its target destination with the desired azimuth and elevation angles, avoiding collisions with obstacles.*

Proof Let us consider the ordinate sequence of recalculation times in which the model predictive algorithm is applied, t_f being the last element: $t_0, t_1, \ldots, t_k, t_{k+1}, \ldots, t_f$, with $t_0 = 0$ and $t_{k+1} = t_k + T_c$.

Now, for each interval $[t_k, t_{k+1})$, $t_k \neq t_f$ we consider the Lyapunov function candidate

$$V_k = \Phi + \frac{1}{2}\left(\theta - \theta_{\mathbf{nh}} - \delta\theta[t_k, t_{k+1})\right)^2 + \frac{1}{2}\left(\psi - \psi_{\mathbf{nh}} - \delta\psi[t_k, t_{k+1})\right)^2 \qquad (5.14)$$

and the extended system

$$x_{ext} = \begin{bmatrix} q_1 \\ \theta \\ \theta_{\mathbf{nh}} \\ \delta\theta \\ \psi \\ \psi_{\mathbf{nh}} \\ \delta\psi \end{bmatrix}, \qquad \dot{x}_{ext} = f(x_{ext}) = \begin{bmatrix} \bar{J}u \\ \omega_\theta \\ \dot{\theta}_{\mathbf{nh}} \\ \dot{\delta\theta} \\ \omega_\psi \\ \dot{\psi}_{\mathbf{nh}} \\ \dot{\delta\psi} \end{bmatrix}$$

We can define the Filipov set $K[f](x_{ext})$ and the generalized derivative of V_k, ∂V_k, as follows:

$$K[f] = \begin{bmatrix} \bar{J}K[u] \\ \omega_\theta \\ \dot{\theta}_{\mathbf{nh}} \\ \dot{\delta\theta} \\ \omega_\psi \\ \dot{\psi}_{\mathbf{nh}} \\ \dot{\delta\psi} \end{bmatrix}, \qquad \partial V_k = \begin{bmatrix} \nabla\Phi \\ \theta - \theta_{\mathbf{nh}} - \delta\theta \\ -(\theta - \theta_{\mathbf{nh}} - \delta\theta) \\ -(\theta - \theta_{\mathbf{nh}} - \delta\theta) \\ \psi - \psi_{\mathbf{nh}} - \delta\psi \\ -(\psi - \psi_{\mathbf{nh}} - \delta\psi) \\ -(\psi - \psi_{\mathbf{nh}} - \delta\psi) \end{bmatrix}$$

And applying the chain rule we get

$$\begin{aligned}\dot{\tilde{V}} = &\bigcap_{\xi \in \partial V_k} \xi^T(K[f](x_{ext}, t)) = \\ =&\bar{J}K[u]\nabla\Phi + \omega_\theta(\theta - \theta_{\mathbf{nh}} - \delta\theta) - \dot{\theta}_{\mathbf{nh}}(\theta - \theta_{\mathbf{nh}} - \delta\theta) - \dot{\delta\theta}(\theta - \theta_{\mathbf{nh}} - \delta\theta) + \\ &+ \omega_\psi(\psi - \psi_{\mathbf{nh}} - \delta\psi) - \dot{\psi}_{\mathbf{nh}}(\psi - \psi_{\mathbf{nh}} - \delta\psi) - \dot{\delta\psi}(\psi - \psi_{\mathbf{nh}} - \delta\psi)\end{aligned}$$

Substituting the expressions for ω_θ and ω_ψ (see (5.11)) and defining $P \triangleq \bar{J}\nabla\Phi$, we obtain

$$\begin{aligned}\dot{\tilde{V}}_k &= \bar{J}K[u]\nabla\Phi - k_\theta(\theta - \theta_{\mathbf{nh}} - \delta\theta)^2 - k_\psi(\psi - \psi_{\mathbf{nh}} - \delta\psi)^2 = \\ &= PK\left[-\operatorname{sign}(P)k_u \left|\left|\nabla\Phi\right|\right|^2\right] - k_\theta(\theta - \theta_{\mathbf{nh}} - \delta\theta)^2 - k_\psi(\psi - \psi_{\mathbf{nh}} - \delta\psi)^2 = \\ &= -|P|k_u \left|\left|\nabla\Phi\right|\right|^2 - k_\theta(\theta - \theta_{\mathbf{nh}} - \delta\theta)^2 - k_\psi(\psi - \psi_{\mathbf{nh}} - \delta\psi)^2\end{aligned}$$

The last two terms of the previous sum are always nonpositive. The first term is the product of $k_u \left|\left|\nabla\Phi\right|\right|^2 > 0$ (we are considering $t_k \neq t_f$ and $\nabla\Phi$ is practically null only for $q_1 = q_{1d}$ since the initial conditions that lead to saddle points are a set of measure zero) and $-|P|$, that is strictly negative since P is null only if $||\nabla\Phi|| = 0$ or if the gradient is normal to the aircraft longitudinal axis[2], but this last condition is not verified as we adjusted the sampling sets following Lemma 1.2.

It is straightforward that $V_{k-1}(t_k) = V_k(t_k)$, as a result of Lemma 1.1. So the concatenation of the Lyapunov function candidates that we considered so far is a continuous function of time.

Let us consider now the Lyapunov function candidate for $t \geq t_f$:

$$V_f = \Phi + \frac{1}{2}\left(\theta - \theta_{\mathbf{nh}} - \delta\theta(t \geq t_f)\right)^2 + \frac{1}{2}\left(\psi - \psi_{\mathbf{nh}} - \delta\psi(t \geq t_f)\right)^2.$$

It follows from (5.13) that the concatenation $V = V_1|V_2|\ldots|V_f$ is continuous as well.

Moreover, as we write $\dot{\tilde{V}}_f$ similarly as we did for $\dot{\tilde{V}}_k$, we get $\dot{\tilde{V}}_f \leq 0$, because in this case the first term of the expression can be null.

We now want to apply LaSalle's principle to V, i.e. we want to exploit the fact that the system converges to the largest invariant subset of $S \triangleq \{q|\dot{\tilde{V}} = 0\}$.

[2] i.e. $|\theta - \theta_{\mathbf{nh}}| = \frac{\pi}{2}$ or $|\psi - \psi_{\mathbf{nh}}| = \frac{\pi}{2}$

We have that $\dot{\tilde{V}} = 0 \iff \dot{\tilde{V}}_f = 0$, i.e.

$$-|P|k_u\,||\nabla\Phi||^2 - k_\theta(\theta - \theta_{\mathbf{nh}} - \delta\theta(t \geq t_f))^2 - k_\psi(\psi - \psi_{\mathbf{nh}} - \delta\psi(t \geq t_f))^2 = 0$$

In order for this to hold, the last two terms of the sum must be zero and at least one of $|P|$ and $||\nabla\Phi||$ must be null as well. But actually because of Lemma 1.2 the only possibility to get $|P| = 0$ is that the gradient of the navigation function also gets null, so for the product to be zero it is necessary and sufficient that $||\nabla\Phi||$ is zero. As we already said, the initial conditions leading to saddle points are a set of measure zero, so this practically only happens if $q_1 = q_{1d}$, i.e. at the desired position.

Let us call $S_1 \triangleq \{q | q_1 = q_{1d}\}$ the set of configurations in which the agent has the desired position. If $q \in S$, then

- $q_1 = q_{1d}$, therefore $\theta_{\mathbf{nh}} = \theta_d$ and $\delta\theta = 0$ (for construction of $\theta_{\mathbf{nh}}$ and $\delta\theta$), and similarly for ψ.
- $[(\theta - \theta_{\mathbf{nh}} - \delta\theta(t \geq t_f) = 0) \wedge (\psi - \psi_{\mathbf{nh}} - \delta\psi(t \geq t_f) = 0)]$ implies $\theta = \theta_d, \psi = \psi_d$

So if $q \in S$, the agent has the desired position and orientation and therefore S is the singleton $\{q_d\}$, that is an invariant set since the components of the input u, ω_θ and ω_ψ are null for $q = q_d$.

The convergence to the desired configuration is hence proved.

The collision avoidance comes from the properties of the (dipolar) navigation function. Let us assume no collisions for $t = 0$: so $\Phi(t = 0) < 1$. Now let us write the time derivative of Φ:

$$\dot{\Phi} = \nabla\Phi\dot{q_1} = \nabla\Phi\bar{J}u = P(-\operatorname{sign}(P)k_u||\nabla\Phi||^2) = -|P|k_u||\nabla\Phi||^2 \leq 0 \quad \forall t \geq 0$$

Therefore $\Phi(t) < 1$, $\forall t > 0$ and no collisions occur.

□

Chapter 6

Conclusions

In this thesis convergence and obstacle avoidance navigation for an air vehicle has been presented. After a detailed description of the quadrotor platform and the Smart Mobility Lab testbed, a general model for quadrotor dynamics has been obtained and the relations between the control inputs and the behavior of the agent have been analyzed through system identification. An heuristic controller for obstacle avoidance has been depicted and the experimental results in the testbed have shown its reliability for our control goals. A more elaborated controller for a general model of air vehicle has been presented and its properties have been demonstrated, in order to provide a more general flight scenario, independent from the specific quadrotor setup, that could be exploited in the testbed in the future.

We wish to give some advice wrt the hardware of the testbed. The arms and rotors of the quadrotor were quite fragile and required frequent replacements: to overcome this problem components of more resistant material could be considered. Furthermore, since the height control has been the most challenging and the most imprecise as well, less powerful motors could be tested on the same quadrotor, in order for the response to the throttle input to be smoother and for the battery to have a longer lifetime between charges. For the same reason, a suggestion for future implementation is increasing the input quantization.

As far as the control task is concerned, different controllers for waypoint navigation, for instance LQG controllers, could be tested on the same platform and performances could be compared with the ones of this work. The motion capture system could be integrated with vision sensors onboard the vehicle in

order to provide, for instance, the simulation of inspection or data gathering tasks and IR sensors could be installed in order to improve safety in obstacle avoidance. Nonlinear identification could be performed on the system to check if more accurate relations can be retrieved and a control strategy that exploits explicitly the knowledge of these relations could be studied. The control analyzed in the last chapter or an adaption could be simulated and implemented in the testbed in the future with air vehicle models fitting the depicted dynamics. Cooperative navigation between two or more air vehicles could be implemented, paying attention for the communication chain to be fast enough to provide safe control.

Bibliography

[1] M. Dempsey, "Eyes of the army – u.s. army roadmap for unmanned aircraft systems 2010–2035", 2010. [Online]. Available: http://www-rucker.army.mil/usaace/uas/US%20Army%20UAS%20RoadMap%202010%202035.pdf.

[2] K. Fregene, "Unmanned aerial vehicles and control: lockheed martin advanced technology laboratories", *Control Systems, IEEE*, vol. 32, no. 5, pp. 32–34, 2012, ISSN: 1066-033X. DOI: 10.1109/MCS.2012.2205474.

[3] S. Ferrari, M. Anderson, R. Fierro, and W. Lu, "Cooperative navigation for heterogeneous autonomous vehicles via approximate dynamic programming", in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, 2011, pp. 121–127. DOI: 10.1109/CDC.2011.6161127.

[4] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed", *Robotics Automation Magazine, IEEE*, vol. 17, no. 3, pp. 56–65, 2010, ISSN: 1070-9932. DOI: 10.1109/MRA.2010.937855.

[5] D. Mellinger, N. Michael, M. Shomin, and V. Kumar, "Recent advances in quadrotor capabilities", in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 2964–2965. DOI: 10.1109/ICRA.2011.5980163.

[6] P. Bouffard, A. Aswani, and C. Tomlin, "Learning-based model predictive control on a quadrotor: onboard implementation and experimental results", in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 279–284. DOI: 10.1109/ICRA.2012.6225035.

[7] D. Lee, C. Nataraj, T. Burg, and D. Dawson, "Adaptive tracking control of an underactuated aerial vehicle", in *American Control Conference (ACC), 2011*, 2011, pp. 2326–2331.

[8] A. Saif, M. Dhaifullah, M. Al-Malki, and M. Shafie, "Modified backstepping control of quadrotor", in *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, 2012, pp. 1–6. DOI: 10.1109/SSD.2012.6197975.

[9] T. Madani and A. Benallegue, "Backstepping control for a quadrotor helicopter", in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006, pp. 3255–3260. DOI: 10.1109/IROS.2006.282433.

[10] M. Khelfi and A. Kacimi, "Robust control with sliding mode for a quadrotor unmanned aerial vehicle", in *Industrial Electronics (ISIE), 2012 IEEE International Symposium on*, 2012, pp. 886–892. DOI: 10.1109/ISIE.2012.6237206.

[11] C.-L. Hwang and C. Jan, "Fuzzy decentralized sliding-mode under-actuated trajectory-tracking control for quadrotor unmanned aerial vehicle", in *Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*, 2012, pp. 1–10. DOI: 10.1109/FUZZ-IEEE.2012.6251290.

[12] A. Roza and M. Maggiore, "Path following controller for a quadrotor helicopter", in *American Control Conference (ACC), 2012*, 2012, pp. 4655–4660.

[13] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing", in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 2472–2477. DOI: 10.1109/ICRA.2011.5980095.

[14] J. van den Berg, D. Wilkie, S. Guy, M. Niethammer, and D. Manocha, "Lqg-obstacles: feedback control with collision avoidance for mobile robots with motion and sensing uncertainty", in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 346–353. DOI: 10.1109/ICRA.2012.6224648.

[15] L. Garcia-Delgado, A. Dzul, V. Santibanez, and M. Llama, "Quad-rotors formation based on potential functions with obstacle avoidance", *Control Theory Applications, IET*, vol. 6, no. 12, pp. 1787–1802, 2012, ISSN: 1751-8644. DOI: 10.1049/iet-cta.2011.0370.

[16] M. Amoozadeh, *Smart mobility lab manual*, 1st ed., KTH Stockholm, 2013. [Online]. Available: https://code.google.com/p/kth-smart-mobility-lab/downloads/list.

[17] C. Anderson et al. (2013). Google code page for the arducopter project, [Online]. Available: https://code.google.com/p/arducopter/.

[18] Moteiv corporation, "Tmote sky datasheet", Data sheet, 2006. [Online]. Available: http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf.

[19] T. Bresciani, "Modelling, Identification and Control of a Quadrotor Helicopter", Master's thesis, Lund University, Sweden, 2008.

[20] D. Lee, T. Burg, D. Dawson, D. Shu, B. Xian, and E. Tatlicioglu, "Robust tracking control of an underactuated quadrotor aerial-robot based on a parametric uncertain model", in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, 2009, pp. 3187–3192. DOI: 10.1109/ICSMC.2009.5346158.

[21] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, "Precision flight control for a multi-vehicle quadrotor helicopter testbed", *Control Engineering Practice*, vol. 19, pp. 1023–1036, 9 2011, ISSN: 0967-0661. DOI: 10.1109/MRA.2010.937855.

[22] E. Altug, J. Ostrowski, and R. Mahony, "Control of a quadrotor helicopter using visual feedback", in *Robotics and Automation, 2002. Proceedings. ICRA 2002. IEEE International Conference on*, vol. 1, 2002, 72–77 vol.1. DOI: 10.1109/ROBOT.2002.1013341.

[23] S. Bouabdallah, "Design and control of quadrotors with application to autonomous flying", PhD thesis, EPFL, 2006.

[24] S. Formentin and M. Lovera, "Flatness-based control of a quadrotor helicopter via feedforward linearization", in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, 2011, pp. 6171–6176. DOI: 10.1109/CDC.2011.6160828.

[25] L. Kis and B. Lantos, "Time-delay extended state estimation and control of a quadrotor helicopter", in *Control Automation (MED), 2012 20th Mediterranean Conference on*, 2012, pp. 1560–1565. DOI: 10.1109/MED.2012.6265861.

[26] T. Lee, M. Leok, and N. McClamroch, "Nonlinear robust tracking control of a quadrotor uav on se(3)", in *American Control Conference (ACC), 2012*, 2012, pp. 4649–4654.

[27] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors", in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[28] T. Söderström and P. Stoica, Eds., *System identification*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988, ISBN: 0-138-81236-5. [Online]. Available: http://user.it.uu.se/~ts/sysidbook.pdf.

[29] D. Koditschek and E. Rimon, "Robot navigation functions on manifolds with boundary", *Advances in Applied Mathematics*, vol. 11, no. 4, pp. 412–442, 1990.

[30] G. Roussos, D. V. Dimarogonas, and K. J. Kyriakopoulos, "3d navigation and collision avoidance for nonholonomic aircraft-like vehicles", *International Journal of Adaptive Control and Signal Processing*, vol. 24, no. 10, pp. 900–920, 2010, ISSN: 1099-1115. DOI: 10.1002/acs.1199.

[31] S. Maniatopouloss, "Development of Predictive Navigation Schemes for Aircraft-like Vehicles", Master's thesis, National Technical University of Athens, Greece, 2012.

[32] H. Tanner, S. Loizou, and K. Kyriakopoulos, "Nonholonomic navigation and control of cooperating mobile manipulators", *Robotics and Automation, IEEE Transactions on*, vol. 19, no. 1, pp. 53–64, 2003, ISSN: 1042-296X. DOI: 10.1109/TRA.2002.807549.