Quadrotor control: Implementation, cooperation and Human-vehicle interaction

SALVATORE SCOTTO DI CLEMENTE

KTH Electrical Engineering

Master's Degree Project
Stockholm, Sweden June 2015

XR-EE-RT 2015:XXX

Abstract

In this thesis two cooperative control strategies are designed for a team of two indoor quadrotors and some human-vehicle interaction tasks are developed. First, the process of assembling a quadrotor from parts is undertaken. Then, a controller for a single quadrotor is designed, and implemented using LabView software. Matlab software is used to generate the desired trajectories. A LabView software module acts as a supervisor, allowing the coordination between the two vehicles. Two types of cooperative control are developed: position-based, and position/velocity-based. Both control strategies allow the quadrotors to follow a reference trajectory while maintaining a specified formation. Furthermore, some tasks are developed where one quadrotor responds to online stimuli given by a human via a user-friendly interface.

Contents

Chapter 1

Introduction

An Unmanned Aerial Vehicle (UAV) is an unpiloted aircraft which can either be remotely controlled or fly autonomously based on pre-programmed flight plans. Building upon rapid advances in robotics, control, communications and computer technology, UAVs are foreseen to play increasingly important roles in a multiplicity of civilian and military application scenarios. Military strategic planning has already started to incorporate wide-ranging roles for UAVs, such as tactical surveillance, communications relay, target designation, battle damage assessment and covert payload delivery [1]. Teams of autonomous intelligent vehicles with common mission objectives into military force structures are integrated. The problem of cooperative control of UAVs concerns the coherent and efficient maneuvering of each member of a group of aerial vehicles (the team) to successfully complete a mission with limited human intervention in a highly unstructured environment [1]. This can be achieved by devising control algorithms, implemented on digital hardware, that allocate tasks to each UAV member of the team, select flight paths and generate the trajectory for each member, and set attitude configurations to aerial vehicles at timely instants, at precise positions or during specific maneuvers such as evasion, combat, reconnaissance, take-off, landing, rendezvous and so on. The control objectives can be characterized as the optimization of a set of designer-specified global functions. A central motivation for the development of cooperative control schemes is that enabling UAV teaming should result in a more effective operational capability than that available through independent control of the UAVs. This idea, along with the need to leave it to a group of machines to effectively perform the dull, dangerous and dirty missions, are in fact the main drivers for the research in cooperative control today [1]. An effective cooperative control strategy should provide close-to optimal, robust, real-time performance with a relatively fast response from the team. Team autonomy must be achieved via algorithms for the scheduling of tasks, the planning of vehicle paths and the generation of trajectories for each vehicle.

1.1 Literature review

In the last few years, cooperation in teams of multiple quadrotors has been extensively studied [2, 21]. To perform a complex task a single quadrotor may not be enough. When multiple quadrotors cooperate for the execution of a complex task, a centralized or decentralized control strategy must be designed. In a centralized control strategy, one component is designated as the controller and is responsible for managing the execution of other quadrotors. Figure 1.1 shows the centralized control scheme.

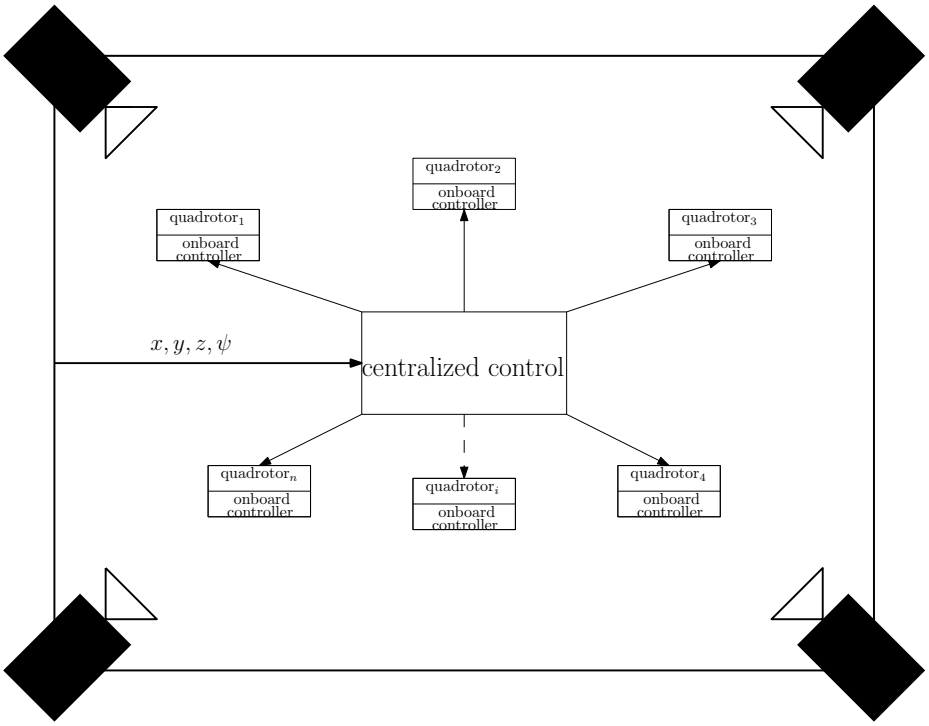


Figure 1.1: Centralized control scheme.

A decentralized system is one which requires multiple parties to make their own decisions. In such a decentralized system, there is no single centralized authority that makes decisions on behalf of all the parties. Figure 1.2 shows the decentralized control scheme.

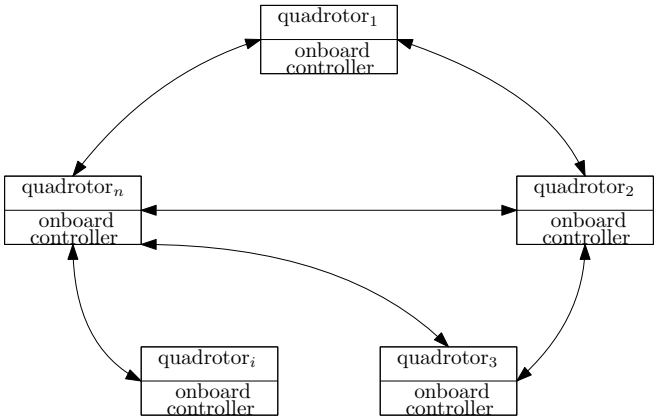


Figure 1.2: Decentralized control scheme.

An example of task with a single quadrotor is given in [2], where the control objective is to have the quadrotor moving a load as showed in Figure 1.3.

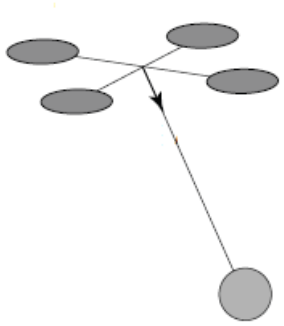


Figure 1.3: Quadrotor with a load [2].

An example of task with more quadrotors is given in [3]. It addresses the problem of cooperative transportation of a cable suspended payload by multiple quadrotors as showed in Figure 1.4.

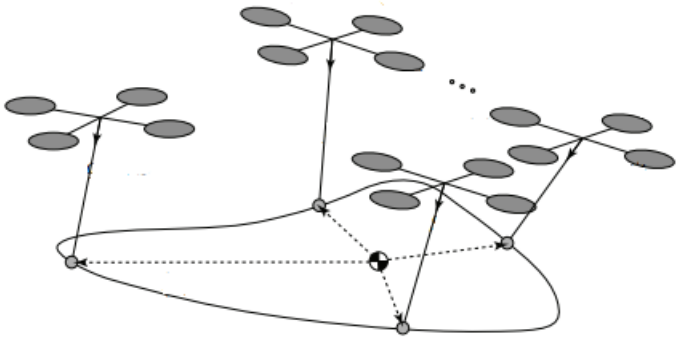


Figure 1.4: Network of quadrotors [3].

In this work, we implemented one feedback linearization called *dynamic inversion with zero-dynamics stabilization* for two quadrotors and one cooperative high level control for the simulation part. Furthermore the PID control for two quadrotors and two high level control strategies for some cooperative tasks in the practical part.

1.2 Thesis outline

The rest of this thesis is organized as follows.

Chapter 2. In this chapter the necessary background for the project is given. This includes a qualitative introduction on the principles of working of a quadrotor.

Chapter 3. In this chapter the hardware and software of the testbed is presented. The quadrotor platform, the motion capture system and the communication link are described in detail.

Chapter 4. This chapter shows the simulated world for two quadrotors using a feedback linearization as controller for both quadrotors. A software module acts as a supervisor is implemented and a SIMULINK toolbox called SIMULINK 3D ANIMATION is used to visualize the behaviour of both quadrotors.

Chapter 5. In this chapter two types of cooperative high level control are developed: position-based, and position/velocity-based. Both control strategies allow the quadrotors to follow a reference trajectory while maintaining a specified formation. Some tasks (i.e. Master/Slave behaviour) in the real word are showed and some videos are linkable.

Chapter 6. In this chapter some tasks (*Gun for motion in cylindrical coordinates*, *Gun for motion in spherical coordinates*, *Gun for projection tracking* and *Gun for trajectory design*) are developed where one quadrotor responds to online stimuli given by a human via a user-friendly interface. Several pictures illustrate the execution of such task. Some videos showing the actual tasks' execution are linked.

Chapter 7. Finally, in chapter 7, conclusions and a discussion for future work are presented.

Chapter 2

Preliminar notions

The quadrotor, an aircraft made up of four engines, holds the electronic board in the middle and the engines at four extremities. The attitude and position of the quadrotor can be controlled to desired values by changing the speeds of the four motors. The following forces and moments can be performed on the quadrotor: the thrust caused by rotors rotation, the pitching moment and rolling moment caused by the difference of four rotors thrust, the gravity, the gyroscopic effect, and the yawing moment. The gyroscopic effect only appears in the lightweight construction quadrotor. The yawing moment is caused by the unbalanced of the four rotors rotational speeds. The yawing moment can be cancelled out when two rotors rotate in the opposite direction. So, the propellers are divided in two groups. In each group there are two diametrically opposite motors that we can easily observe thanks to their direction of rotation. Namely, we distinguish:

- front and rear propellers (numbers 2 and 4 in Figure 2.1), rotating counterclockwise;
- right and left propellers (numbers 1 and 3 in Figure 2.1), rotating clockwise.

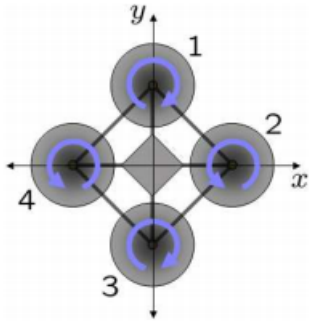


Figure 2.1: Direction of propeller's rotations.

The space motion of the rigid body aircraft can be divided into two parts: the barycenter movement and movement around the barycenter. Six degrees of freedom

are required in describing any time space motion. They are three barycenter movements and three angular motions, namely, three translation and three rotation motions along three axes. The control for six degrees of freedom motions can be implemented by adjusting the rotational speeds of different motors. The motions include forward and backward movements, lateral movement, vertical motion, roll motion, and pitch and yaw motions. The yaw motion of the quadrotor can be realised by a reactive torque produced by the rotor. The size of the reactive torque is relative to the rotor speed. When the four rotor speeds are the same, the reactive torques will balance each other and quadrotor will not rotates, whereas if the four rotor speeds are not absolutely same, the reactive torques will not be balanced, and the quadrotor will start to rotate. When the four rotor speeds synchronously increase and decrease is also required in the vertical movement. Because of four inputs and six outputs in a quadrotor, such quadrotor is considered an underactuated nonlinear complex system. In order to control it, some assumptions are made in the process of quadrotor modeling: the quadrotor is a rigid body; the structure is symmetric; the ground effect is ignored.

Depending on the speed rotation of each propeller it is possible to identify the four basic movements of the quadrotor, which are showed in Figures 2.2 to 2.5.

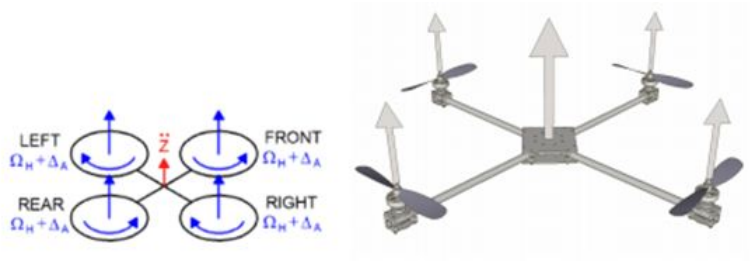


Figure 2.2: Thrust.

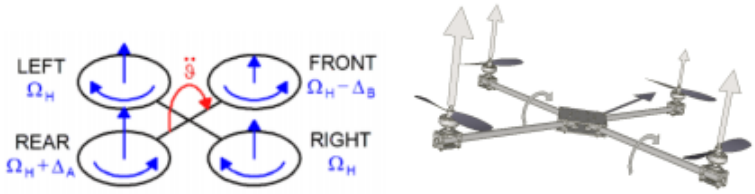


Figure 2.3: Pitch.

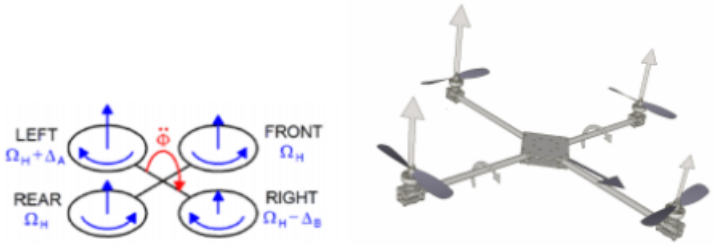


Figure 2.4: Roll.

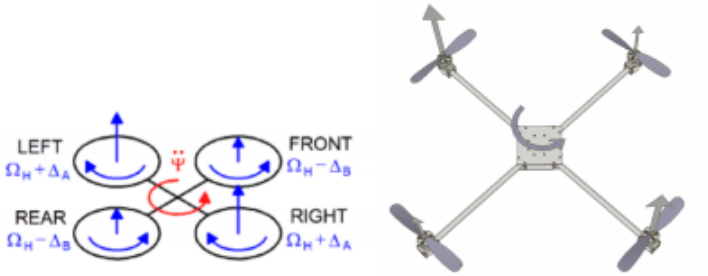


Figure 2.5: Yaw.

Chapter 3

Quadrotor's resources

In this work, we used two JDrones ArduCopter quadrotors (see Figure 3.1).

Figure 3.1: ArduCopter quadrotor [4].

The experiments were made in the Smart Mobility Lab (SML) in KTH. A camera system is used to detect the position and the orientation of the quadrotors.

The main components of the testbed are:

- the quadrotors;
- the Qualisys Motion Capture System;
- a PC running the motion capture software;
- a PC running NI LabView and Matlab;
- a safety net that protects the people in the room when the quadrotors fly.

We summarize in the next lines the representation of the testbed and how its various parts communicate and form a closed loop. To get the current position and orientation of the two quadrotors, we used 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 data 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/Simulink scripts into the LabView code. The reference signals coming from LabView (i.e., one byte for each of desired thrust, 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, 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 (ESC) that set the input voltages of each brushless AC motor individually. The scheme is illustred in Figure 3.2.

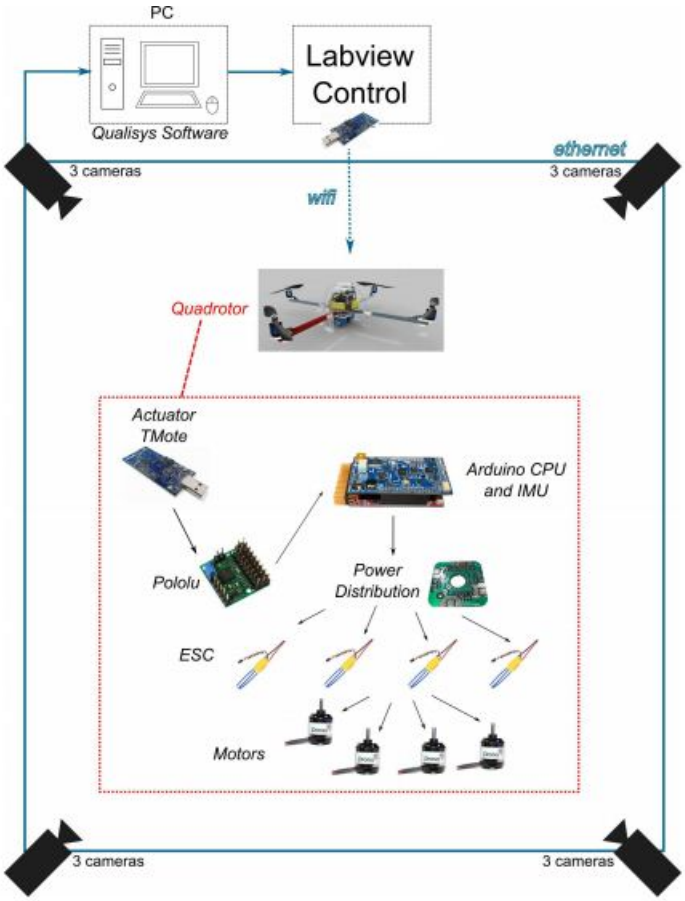


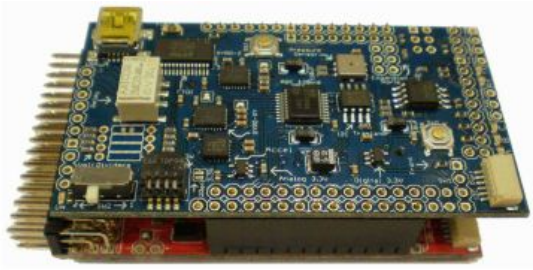
Figure 3.2: Quadrotor's resources.

3.1 Jdrones ArduCopter

Arducopter is a platform for multirotors that provides both manual remote control and an autonomous flight system.

(a) Control board

(b) IMU board.

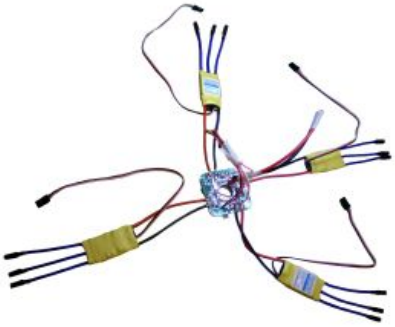


(c) ArduPilot Mega soldered.

Figure 3.3: ArduPilot Mega kit.

The quadrotors have been assembled from the ArduPilot Mega (APM) kit (Figure 3.3c), containing the Arduino controller board (Figure 3.3a) and the IMU (Inertial Measurement Unit) board (Figure 3.3b), and the JDrones ArduCopter kit (Figure 3.4), containing the frame and the motors (Figure 3.4a), the electronic speed controllers (ESCs) and the power distribution board (Figure 3.4b) and the propellers (Figure 3.4c) [4].

(a) A motor connected to the arm.



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

(c) Propellers of the quadrotors.

Figure 3.4: JDrones ArduCopter kit

3.2 Mission Planner software

The Mission Planner (MP) [6] software, is used to upload the ArduCopter firmware to the board, update it and perform some tuning operations. It has several features in addition to the ones we currently use, above all for outdoor flight. Figure 3.5 shows the main screen of the MP software.

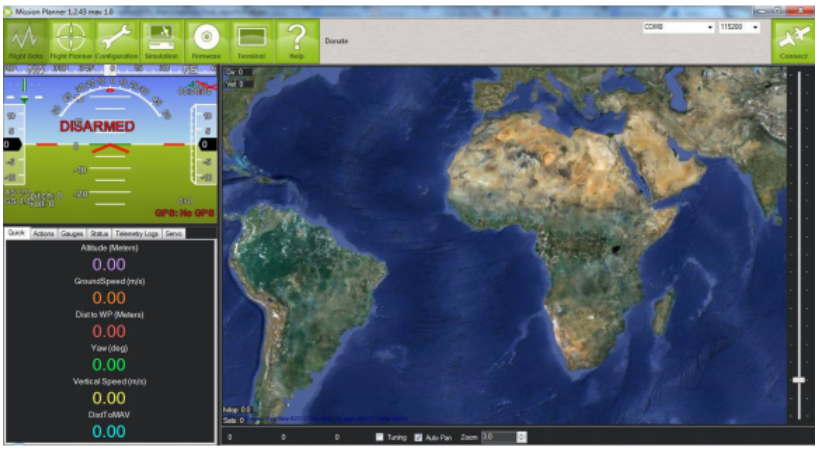


Figure 3.5: Mission Planner software[6].

Before flight, some procedures must be carried out in order to set the quadrotor in a proper way, exploiting the MP software. An example is the calibration of the quadrotor. From the main window of MP it is possible to see if the board has a correct calibration.

(a) Bad calibration.

(b) Good calibration.

Figure 3.6: Examples of how the ground-sky frame of the MP can appear when the board is places on a flat surface.

In order to upload a new firmware to the boards and to calibrate the quadrotors, we use the software Mission Planner. To do this, we connect the boards to an USB port of the PC. Some important calibration are the following.

- Radio range calibration: this operation has to be done before the first flight and whenever the quadrotor does not seem to respond accordingly to the input commands. It consists in sending the minimum and the maximum values (send 0 and 127 respectively) for each of the four channels (thrust, roll, pitch and yaw). To send the minimum and the maximum values we used LabView software.

- Accelerometer calibration: the procedure consists in calibrating the accelerometer of the APM board. We have 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 quadrotor and whenever we suspect that the speed of the motors are not correctly balanced.

3.3 Qualisys Motion Capture System

The motion capture (mocap) system produced by Qualisys consists in twelve IR cameras, each of them equipped with an infrared flash. The light of the flashes is reflected by small reflective balls (markers) that are previously placed on the objects we are interested in tracking, and the cameras capture these reflected beams and compute relative position of the markers. The information collected by every camera is transmitted via ethernet to a computer running the Qualisys Tracking Manager (QTM) software [7] which in turn computes the 3D position of the markers.

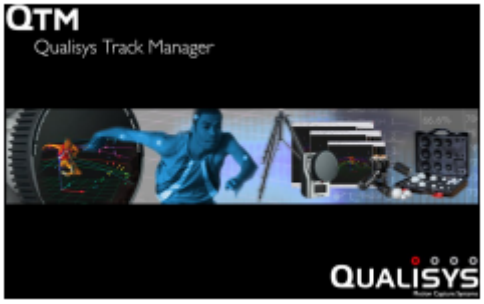


Figure 3.7: Qualisys Tracking Manager [7].

If a group of markers have been saved as rigid body, then the software is able to track not only 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. In our testbed the cameras are twelve, hung in groups of three to the four corners of the ceiling.

Figure 3.8: Group of three cameras in SML.

In our testbed the cameras are twelve, 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 3.1. For all our experiment

Table 3.1: Properties of camera's system [7].

Camera	**Normal mode (full FOV)**	**High speed mode (full FOV)**	**Max fps (reduced FOV)**
Oqus 4	3 MP 280fps	n/a	10000 fps
Oqus 3+	1.3 MP 500fps	0.3 MP 1750 fps	9%

in the testbed, the frequency of the camera has been set to 100 Hz. 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 reliable for our purposes. Therefore there is absolutely no need to filter the measurements to get more reliable information on position.

To define a new body in the QTM software we have to place five markers on the body in a non-symmetrical position. Then we place the quadrotor in the origin of the fixed reference system with the correct position and orientation, that is the frame reference system and the fixed reference system must match. Therefore we start to capture for few seconds and after it we select the markers and we define a new body.

In this thesis we use two quadrotors and one wooden structure called *Gun* (see section 6). The results in the Qualisys system are shown in Figure 3.9, and their measurements are shown in Figure 3.10.

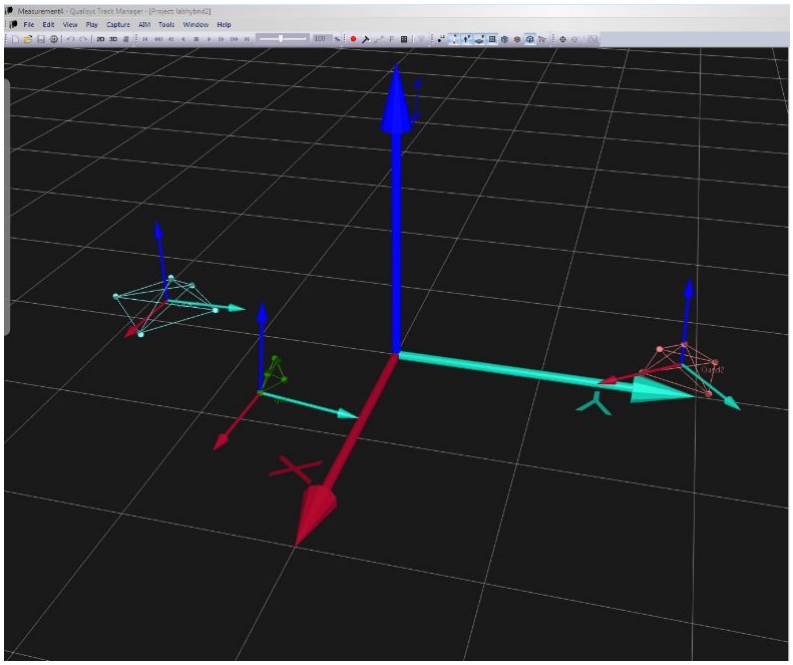


Figure 3.9: Graphical representation of QUALISYS for three rigid body system.

Labeled trajectories (98)

Trajectory	Fill Level	Range	Type	X	Y	Z
gun - 1	< 0.1%	1 - 1	Measured	423.98	-268.64	191.07
gun - 2	< 0.1%	1 - 1	Measured	576.58	-228.91	236.45
gun - 3	< 0.1%	1 - 1	Measured	532.59	-195.43	194.17
gun - 4	< 0.1%	1 - 1	Measured	492.97	-229.76	230.21
gun - 5	< 0.1%	1 - 1	Measured	631.77	-230.48	205.40
Quad2 - 1	< 0.1%	1 - 1	Measured	-49.13	1043.23	108.42
Quad2 - 2	< 0.1%	1 - 1	Measured	-90.42	868.24	106.21
Quad2 - 3	< 0.1%	1 - 1	Measured	164.24	1019.72	120.64
Quad2 - 4	< 0.1%	1 - 1	Measured	80.82	820.76	112.26
Quad2 - 5	< 0.1%	1 - 1	Measured	22.24	935.11	199.70
Quad1 - 1	< 0.1%	1 - 1	Measured	59.75	-1024.73	115.50
Quad1 - 2	< 0.1%	1 - 1	Measured	282.10	-807.45	114.38
Quad1 - 3	< 0.1%	1 - 1	Measured	-107.09	-806.64	104.49
Quad1 - 4	< 0.1%	1 - 1	Measured	40.74	-804.97	209.68
Quad1 - 5	< 0.1%	1 - 1	Measured	65.81	-635.19	117.39

Figure 3.10: QUALISYS system measurements.

In Figure 3.10 the markers of each rigid body on the left column are shown and their positions (x, y, z) on the right columns are also shown.

3.4 Communication link

The wireless communication between the controller computer and the quadrotor 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 [8, 9, 10].

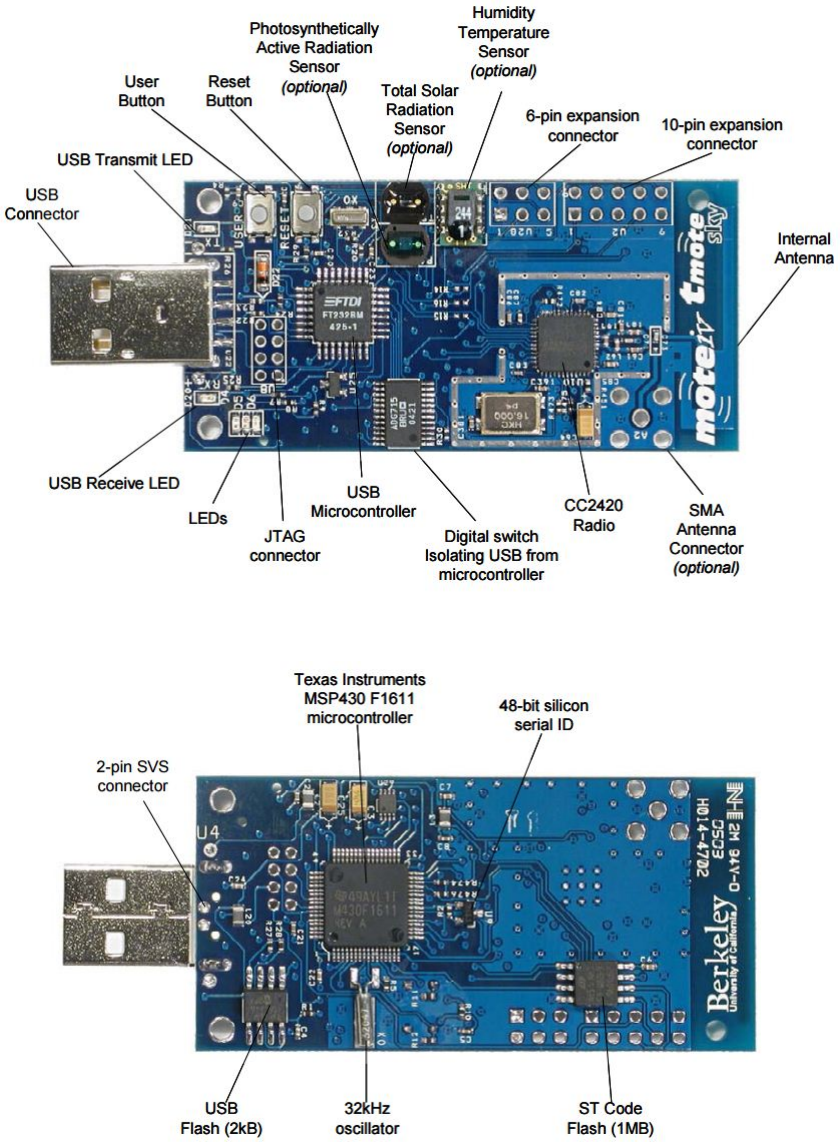


Figure 3.11: Tmote Sky components[9].

One mote is plugged into an USB port of the controller computer and the other one is attached to the quadrotor. They have been programmed using TinyOS, exploiting the code available in the SML repository [11] and refer to the manuals in [12] for

the procedure for programming the motes. The scheme of the communication link is depicted in Figure 3.12.

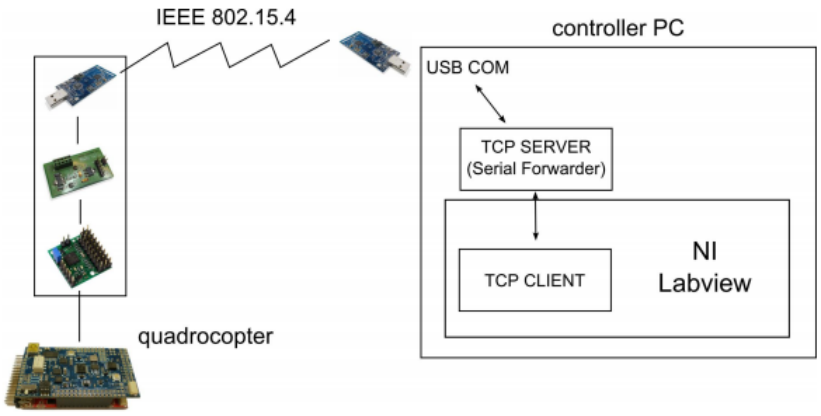


Figure 3.12: Scheme of the communication link between the quadrotor and the controller computer.

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 specified 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 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 to the quadrotor. 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 quadrotor. We use a serial forwarder to send data from computer to the PC-mote. To do so, we used Cygwin in Windows when PC-mote is connected to some port (i.e. COM5) of the PC. Algorithm 1 creates a TCP server process in the computer.

```
sf 9003 COM5 telosb
```

Algorithm 1: Command to create a TCP server process.

In this way the TCP server process listens to port number 9003. Any TCP client can connect to this server and send data to it. The serial forwarder delivers the received data to the PC-mote attached to COM5.

Chapter 4

Simulated world

In this chapter, the simulated cooperative control is discussed. To control the single quadrotor a nonlinear control, feedback linearization, is used. Feedback linearization is an approach to nonlinear control design [13] that has attracted several researches in the last years. The central idea is to algebraically transform nonlinear systems dynamics into (fully or partly) linear ones, so that linear control techniques can be applied. A nonlinear control strategy, dynamic inversion with zero-dynamics stabilization is implemented in this thesis for the simulation control.

4.1 High level cooperative control

The aim of the high level cooperative control is to give the right reference signals $x(t)$, $y(t)$, $z(t)$, $\psi(t)$ to the quadrotors' controllers. In the simulation the high level cooperative control allows the two quadrotors to fly along the circumference. The high level cooperative control has been implemented with MATLAB/SIMULINK. With MATLAB scripts we developed the desired trajectory for both quadrotors [22]. Such MATLAB scripts realize the desired trajectory using a trapezoidal velocity profile described in Appendix A. The reference signals from the high level cooperative control are generated as a function of quadrotors' position. The controller verifies that both quadrotors reach two fixed diametrically opposite reference points, called check points. So, it will compute the next references for both quadrotors. In Figure 4.1 the two check points are shown.

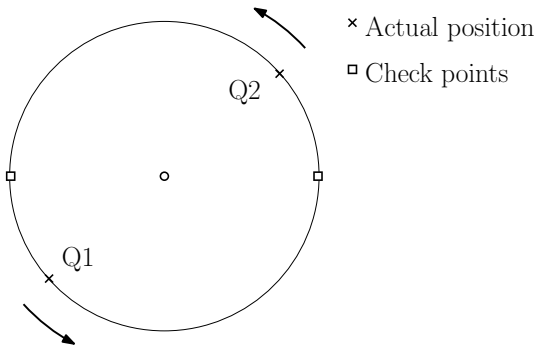


Figure 4.1: Check points for the high level cooperative control.

In Figure 4.2 the block diagram of such high level cooperative controller is shown.

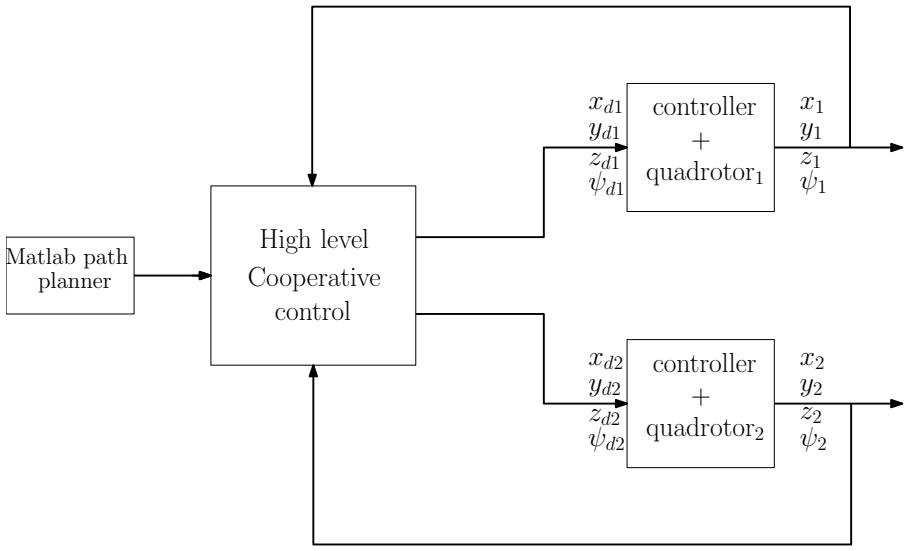


Figure 4.2: Block diagram of the high level cooperative control.

Algorithm 2 shows the pseudo code of the controller.

```
err = constant;
check_point p1;
check_point p2;
while do
    position q1 = get_position_q1;
    position q2 = get_position_q2;
    if (q1 = p1 + err and q2 = p2 + err) or (q1 = p2 + err and q2 = p1 + err)
    then
        give new_reference();
    else
        give old_reference();
    end
end
```

Algorithm 2: Pseudo code of the high level cooperative control.

4.2 Simulation results using dynamic inversion with zero-dynamics stabilization

In our simulation, the high level control allows the two quadrotors to fly along the circumference. The dynamic inversion with zero-dynamics stabilization is implemented for both quadrotors, and the corresponding SIMULINK scheme is shown in Figure 4.3.

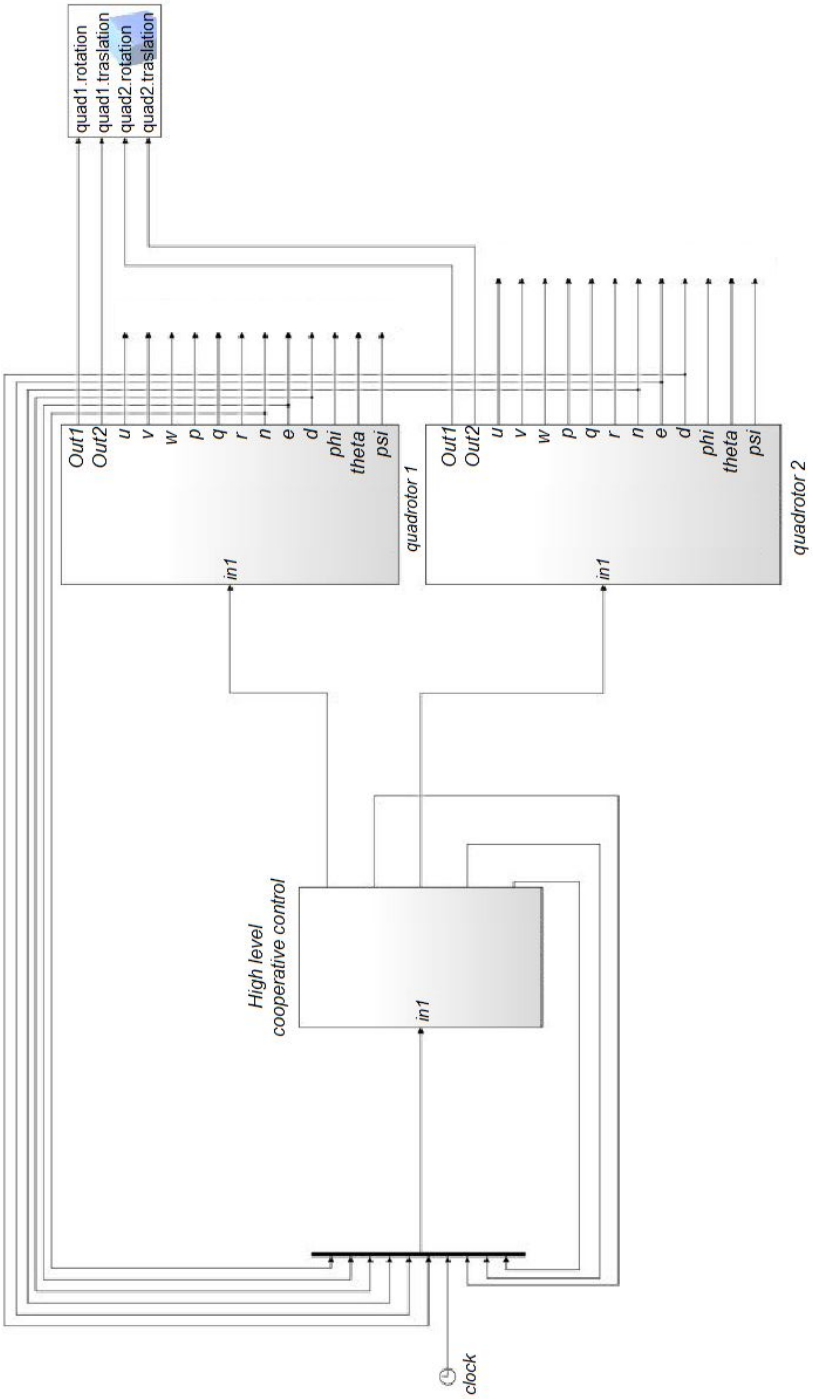


Figure 4.3: High level control scheme in Simulink.

In Figure 4.4 the ideal and the simulated trajectory are shown.

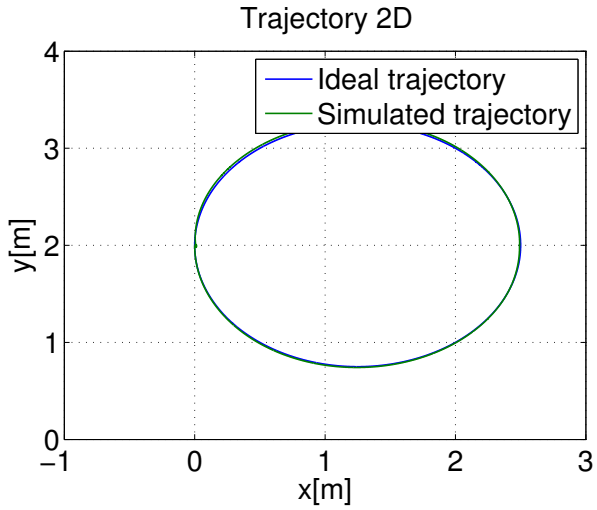
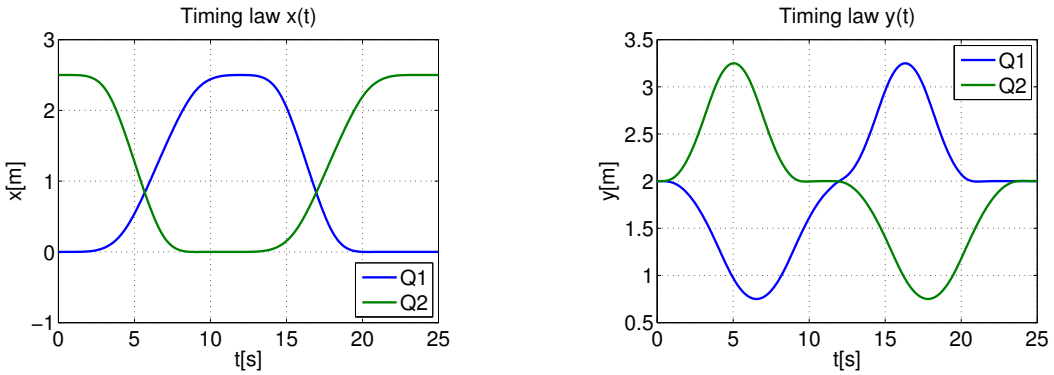


Figure 4.4: Ideal and simulated trajectory for one of the two quadrotors.

The controller verify that both quadrotors reach two fixed diametrically opposite reference points and when the positions are right, both quadrotors restart to follow the desired trajectory. To clarify this task, see Figure 4.1 and Figure 4.6. Called $x_1(t)$, $y_1(t)$ the position variables of the first quadrotor Q_1, and called $x_2(t)$, $y_2(t)$ the one of second quadrotor Q_2, the timing laws are shown in Figure 4.5.



(a) $x_1(t)$ and $x_2(t)$ timing laws.

(b) $y_1(t)$ and $y_2(y)$ timing laws.

Figure 4.5: Q_1 and Q_2 timing laws.

It has been noticed that when quadrotor Q_2 is the faster, it makes a semi-circumference

and it awaits for the second one (Q_1) before restarting again. The same happens when quadrotor Q_1 when is faster than Q_2.

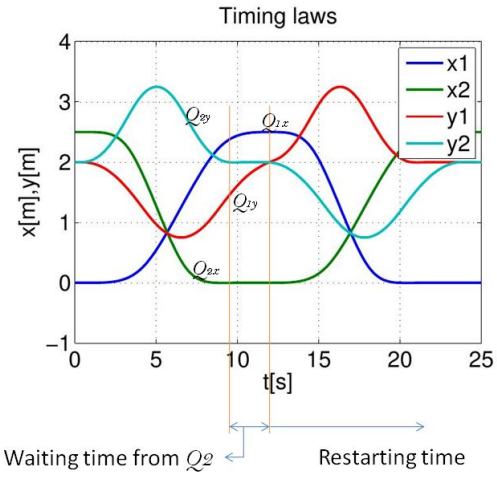


Figure 4.6: Waiting time for the quadrotor Q_2.

To visualize the behaviour of both quadrotors a SIMULINK toolbox called SIMULINK 3D ANIMATION is used. It lets to visualize and verify dynamic system behavior in a virtual reality environment. Objects are represented in the Virtual Reality Modeling Language (VRML), a standard 3D modeling language.
An explicative video is linked here: https://youtu.be/pXXz1LpPCck .

Chapter 5

Real world application for the cooperative control

In this section we will present the controllers that have been implemented in the testbed for both quadrotors. For both quadrotors the same controller's structure is used.

5.1 Single quadrotor control

The aim of the controller is point-to-point navigation in order to make a desired trajectory. The controller is structured in a layered architecture, as we can see in Figure 5.1.

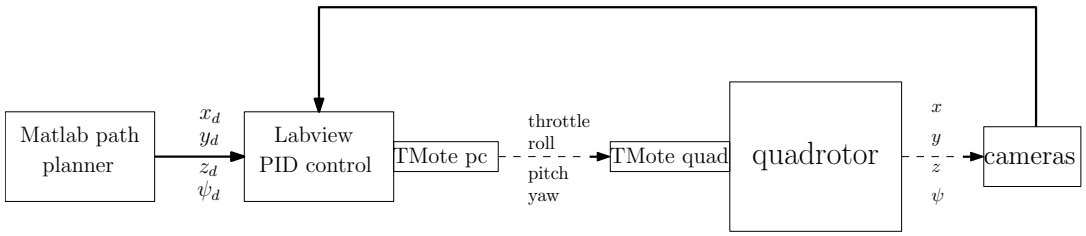


Figure 5.1: Control's structure in the practical application.

The path planning is done exploiting the implementation of a MATLAB function. The references provided by the path planning feed four PID controllers, one for each of the thrust, pitch, roll and yaw movements of the quadrotor and the output of the controllers is sent wireless to the quadrotor. The onboard controllers of the quadrotor stabilize its position exploiting the references for thrust, pitch, roll and yaw. We can say that the quadrotor 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 for at least 4 seconds before the motors can be turned on by a thrust input; moreover, when the thrust is cut and remains null for some time, or the user sends continuously a specific fixed disarm

sequence, the quadrotor needs to be armed again in order to restart to fly. In order to control separately the movements along the x, y and z axis and the yaw rotation, we exploited both the model in (2.22) of [22] and some heuristic considerations we inferred 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 quadrotor to perform off-hand maneuver, and in order to be able to simplify the equations of the model. Thus, we are able to rewrite the equations for the movements in the inertial coordinate system as follows:

$$\begin{cases} \ddot{x} \approx -\frac{f_t}{m}[\theta c(\psi) + \phi s(\psi)] \\ \ddot{y} \approx -\frac{f_t}{m}[\theta s(\psi) - \phi c(\psi)] \\ \ddot{z} \approx g - \frac{f_t}{m} \end{cases} \tag{5.1}$$

where θ is the pitch, ψ is the yaw, m is the mass, g is the gravity acceleration, f_t is the total thrust, s(α)=sin(α), and c(α)=cos(α).
Let us suppose for the moment that the yaw angle is zero. We can assume this without loss of generality, since in our application we are only interested in the position of the quadrotor. This assumption standing, we can rewrite the system (5.1) as shown below:

$$\begin{cases} \ddot{x} \approx -\theta \frac{f_t}{m} \\ \ddot{y} \approx \phi \frac{f_t}{m} \\ \ddot{z} \approx g - \frac{f_t}{m} \end{cases} \tag{5.2}$$

With the previous hypotheses and for a fixed value of the thrust f_t, from (5.2) we see clearly that, the movement along the x and y axes can be controlled independently acting on the pitch and roll inputs. Values of the acceleration $\frac{f_t}{m}$ that are close to the gravity acceleration g are the ones that keep the quadrotor in hovering. The four PIDs that we implement are indicated below.

- One PID controller for the thrust, that controls the position along the z axis in the fixed reference system. We remark here that the control of the thrust influences the magnitude of the accelerations along the x and y axes as well, so higher values for the thrust produce more aggressive movements along these axes, for the same pitch/roll angle.
- One PID controller for the pitch, that controls the position along the x axis of the fixed reference system.
- One PID controller for the roll, that controls the position along the y axis of the fixed reference system.
- One PID controller for the yaw, that controls the yaw angle to zero in order for system (5.2) 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 specific orientation in the xy-plane.

Figure 5.2 shows the control panel of the controller developed in LabView.

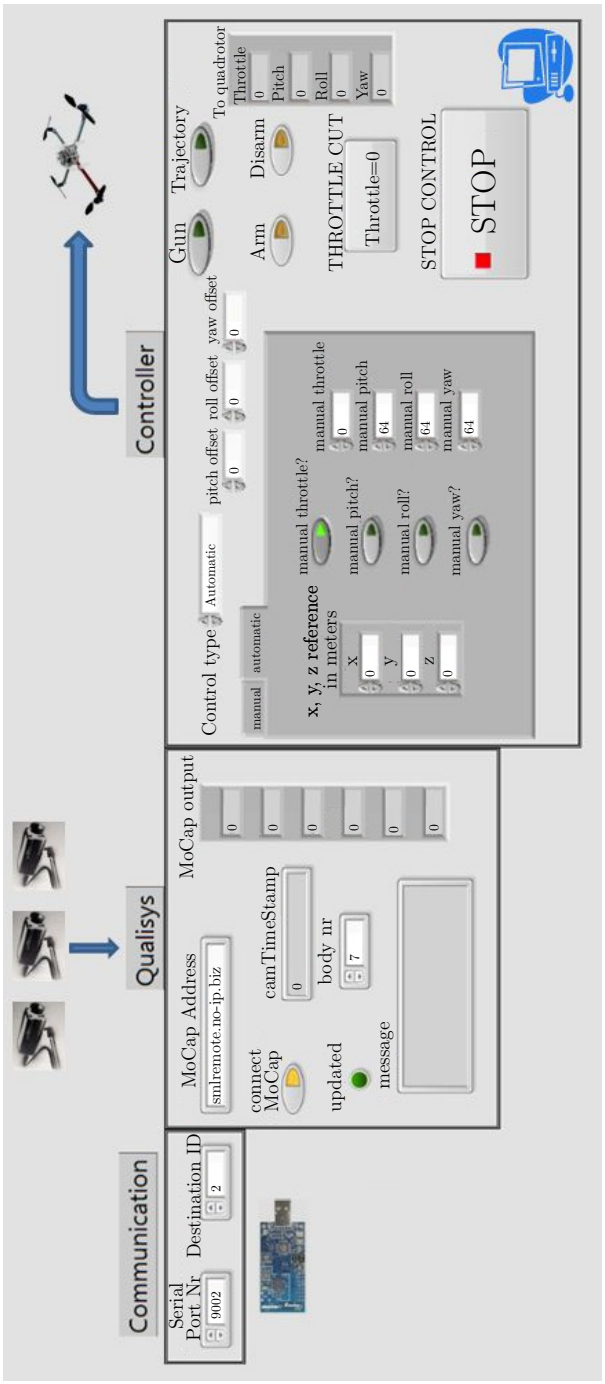


Figure 5.2: Control panel in Labiview.

Note that there are three sections, first is about the communication with the quadrotor, the second one is about Qualisys capture system and the last one is the controller.

From Qualisys capture system section the controller gets the quadrotor position, thus such controller computes the right control signals. Such control signals are sent to the quadrotor by the communication section. Note that such software has two configurations: manual and automatic. The manual mode is used to tune the PID. In the automatic mode there are the PID structures as shown in Figure 5.3. The reference points, the information about positions and orientation of the quadrotor from the cameras and the PID controllers are in a while loop. The sampling time of such loop has been set to 100 ms considering that the bottleneck for the loop speed is the wireless connection between the sender and the receiver Tmotes. In this section we show the methodology used to tune the gains of the PID. To make these operations, the safety net that protects the people in the room was used.

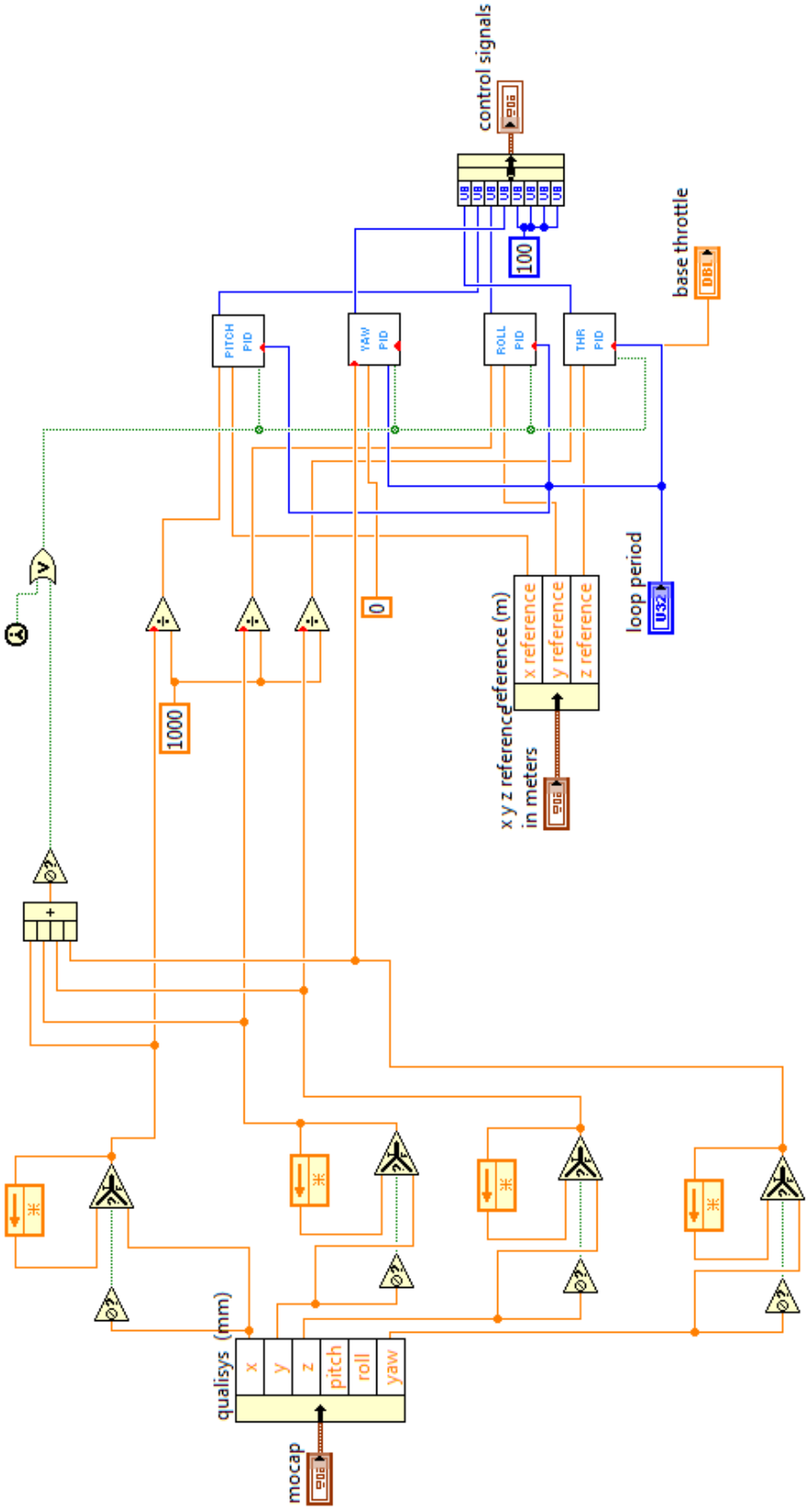


Figure 5.3: Four PID in LabVIEW.

5.1.1 Yaw control

To stabilize the yaw movement (ψ), the PID control law is used:

$$0 = P\psi + I \int \psi + D\dot{\psi}dt. \tag{5.3}$$

We decided to tune the yaw controller first, because its proper functioning is paramount for equations (5.2) to stand and thus for the x and y positions to be controlled independently. The tuning process consisted in placing the quadrotor on the ground with a non-zero angle, give a thrust step for the quadrotor to become airborne and observe the reactivity of the controller to set the desired orientation. The values of the gains of the controller have been set to P=0.025, I=0.001, D=0.

5.1.2 Position control

To stabilize the position (x, y), the PID control law is used:

$$\ddot{x}_d = P(x_d - x) + I \int (x_d - x) + D(\dot{x}_d - \dot{x})dt. \tag{5.4}$$

We gather the tuning controller of the movements along the x and y axes since the dynamics they regulate are equal. This theoretical consideration could not be completely true if the hardware of the quadrotor presents asymmetries along the two longitudinal axes. We noticed anyway that the bending of the arms or imperfections in the propellers do not influence really much the behavior of the controlled system. To tune the PID we set the yaw control in automatic mode (PID mode). Then, placing the quadrotor on the ground and giving a thrust step, we applied some disturbance on the x and y axes. We change the PID gains online until the right behaviour from the quadrotor is obtained. 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. In order to make the response smoother and restrict the overshoot derivative component has been added and an integral term was added to eliminate the steady state error. Once converted to proper input byte to send to the quadrotor, the output of the pitch and roll controllers have been limited to bound the output angles as required from the theory of small oscillations. In order to compensate any asymmetric behavior, for instance due to inaccurate calibration of the accelerometers of the quadrotor, the user is given the possibility to shift (online) the input value corresponding to 0 in the output angles. The values of the gains of the controller have been set to P=1.006, I=0.105, D=0.805.

5.1.3 Altitude control

To stabilize the position (z), the PID control law is used:

$$\ddot{z}_d = P(z_d - z) + I \int (z_d - z) + D(\dot{z}_d - \dot{z})dt. \tag{5.5}$$

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 quadrotor on the ground and activate all the other (already tuned) controllers. A step was given to the thrust 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. After obtaining reasonably reduced oscillations, an integral term was added to eliminate the steady state error. For the height controller the zero output of the PID is not transformed to a zero input for the thrust, indeed the output of the PID controller is added to a base thrust that is suitable to keep the quadrotor in the air. This value oscillates between 48 and 55, depending on the battery status. 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 values of the gains of the controller have been set to P=1.700, I=0.015, D=0.902.

5.2 High level cooperative control

The aim of the high level cooperative control is to give the right reference signals $x(t)$, $y(t)$, $z(t)$, $\psi(t)$ to the quadrotors' controllers. In our project, such control allows the two quadrotors to fly along the circumference in diametrically opposite positions. The high level cooperative control has been implemented with LabView. With Matlab, we compute the desired trajectory for both quadrotors, and the results are exported in two vectors in our LabView program. These two vectors will be the two reference vectors for the two quadrotors. So, having defined the quadrotors in the Qualisys system, from the main panel of our LabView program, we get the information about the position of the quadrotors. The reference signals from the high level cooperative controller are generated in function of the quadrotors' positions. In our project, we implemented two control strategies. The first control strategy is about the position control, while the second control strategy is about the position and velocity control.

5.2.1 Position control

The principle behind this kind of controller is to verify that both quadrotors reach the desired reference points. In the case that the quadrotors reach such reference points (we call this stage *continuous mode*) the controller computes the next references. Instead, in case one of the quadrotors does not reach its reference point (we call this stage *wait mode*) the controller continues to output the previous references for both quadrotors. This allows the quadrotor that is left back, to reach the desired position. More explicative examples are provided in Figure 5.4 and Figure 5.5.

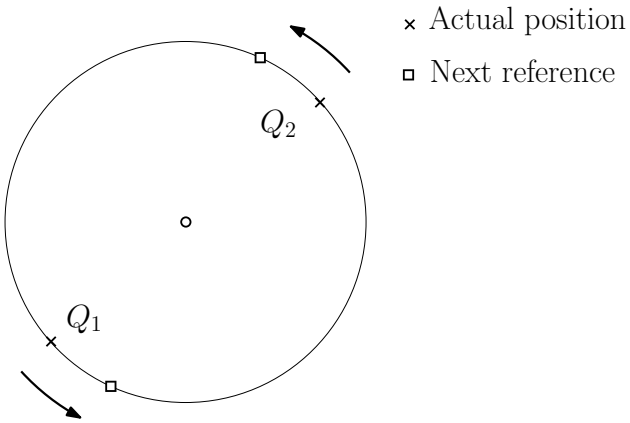


Figure 5.4: Continuous mode in the position control.

In Figure 5.4, the quadrotors are in diametrically opposite positions. Accordingly, the controller is in continuous mode, so it will give the next reference for both quadrotors Q_1 and Q_2.

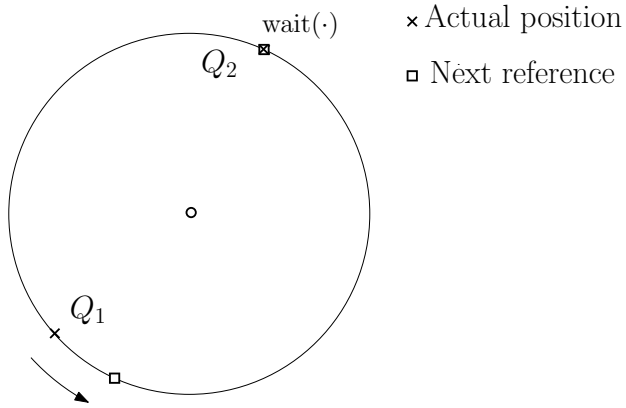


Figure 5.5: Wait mode in the position control.

In Figure 5.5 the two quadrotors are not in diametrically opposite positions, and accordingly the controller is in wait mode. Such controller will give the same reference

for both quadrotors Q_1 and Q_2. Algorithm 3 shows the pseudo code of the position controller.

```
err = constant;
desired_vector d1;
desired_vector d2;
while do
    position q1 = get_position_q1;
    position q2 = get_position_q2;
    if ((q1 = d1(i) + err) and (q2 = d2(i) + err)) then
        give new reference;
    else
        give old reference;
    end
end
```

Algorithm 3: Pseudo code of position control.

5.2.2 Position and velocity control

The high level cooperative controller described above is limited by the fact that whenever the quadrotors are in the wait mode, they are affected by a jerky motion which degrades the overall performance of the task. To overcome this problem, we deployed a high level cooperative controller based on the position and velocity of quadrotors. This time, whenever the quadrotors reach the desired reference points (diametrically opposite position), the controller makes the quadrotors fly at a velocity which depends on how fast the reference vectors is visited. In this case, the velocity will be the same for both quadrotors. We call this velocity *cruise velocity*. Instead, in the case that the quadrotors are not in diametrically opposite positions, the controller slows down the frequency of the updates of the reference for one quadrotor, while it increases the frequency for the other quadrotor. We call this velocity *adaptive velocity*. More explicative examples are provided in the figures below.

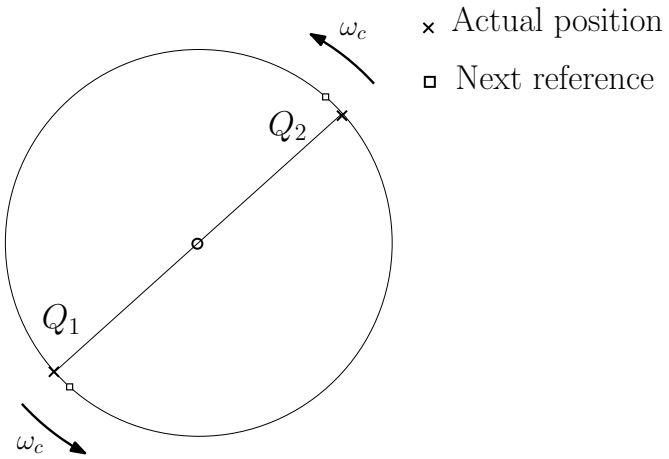


Figure 5.6: Cruise velocity for both quadrotors in the position and velocity control.

In Figure 5.5 the two quadrotors are in diametrically opposite positions. Accordingly, the controller will impose to both quadrotors Q_1 and Q_2 to fly at cruise velocity.

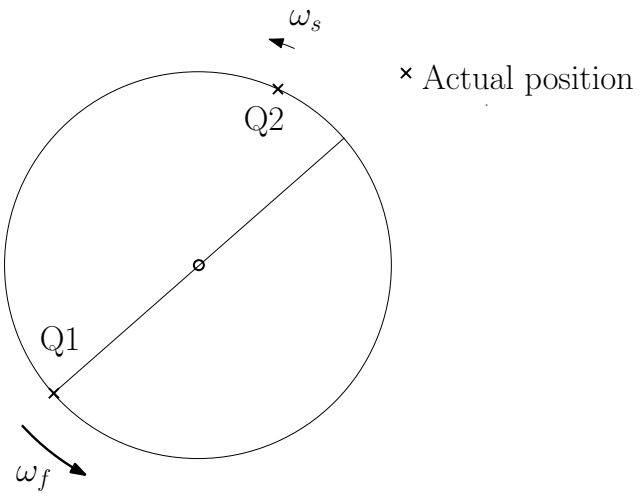


Figure 5.7: Adaptive velocity for both quadrotors in the position and velocity control.

In Figure 5.7 the two quadrotors are not in diametrically opposite positions. Accordingly, the controller will impose the adaptive velocity ω_s for the quadrotor Q_2 that is faster than the quadrotor Q_1, and will impose the adaptive velocity ω_f for the quadrotor Q_2, where $\omega_s < \omega_f$. In order to configure such controller, the user needs to set three time variables (T_c, T_f, T_s) which set the cruise and adaptive velocity. Algorithm 4 shows the pseudo code of the position controller.

```
err = constant;
desired_vector d1;
desired_vector d2;
sampling_time Tc;
adaptive_sampling time Tf;
Tf corresponds to the velocity of the quadrotor that has to speed up;
adaptive sampling time Ts;
Ts corresponds to the velocity of the quadrotor that has to slow down;
while do
    position q1 = get_position_q1;
    position q2 = get_position_q2;
    if (q1 = d1(i) + err) and (q2 = d2(i) + err) then
        give Tc;
    else
        give Ts and Tf;
    end
end
```

Algorithm 4: Pseudo code of position and velocity control.

Notice that $T_c > T_f$ and $T_c < T_s$.

To understand which quadrotor has to slow down and which one has to speed up, we compare the indices of the two reference vectors.

Figure 5.8 shows the control panel of the high level control developed in LabView. On the left there are the information coming out from Qualisys system, then there are the set points for both quadrtotors, and on the right there are the information that the controller send to both quadrotors. The start and stop button are used to start and to end the cooperative control.

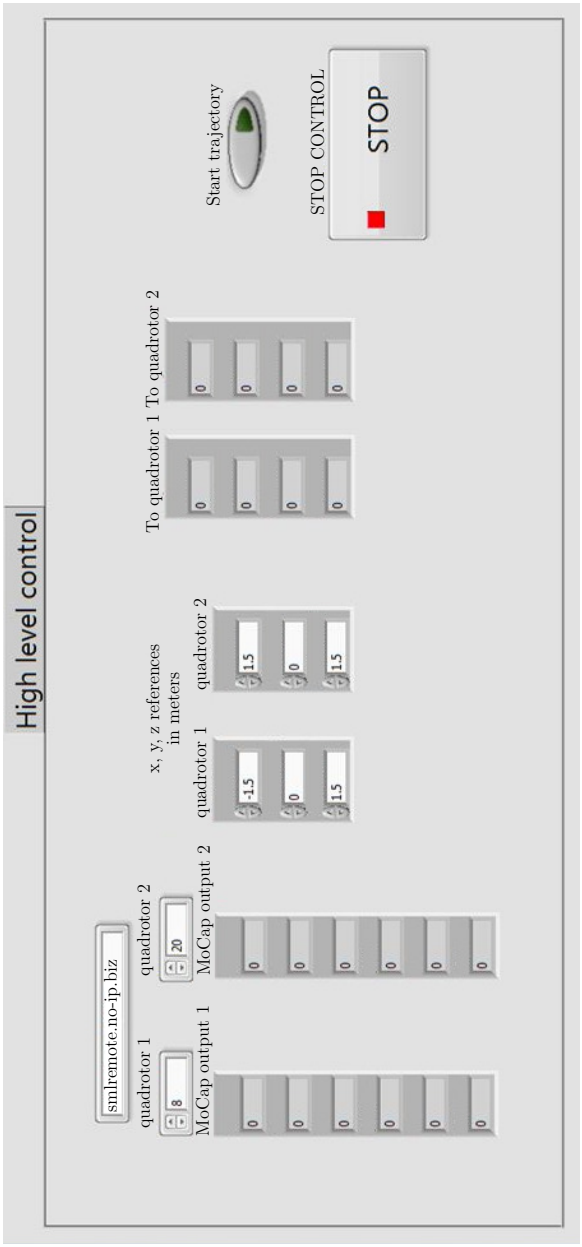
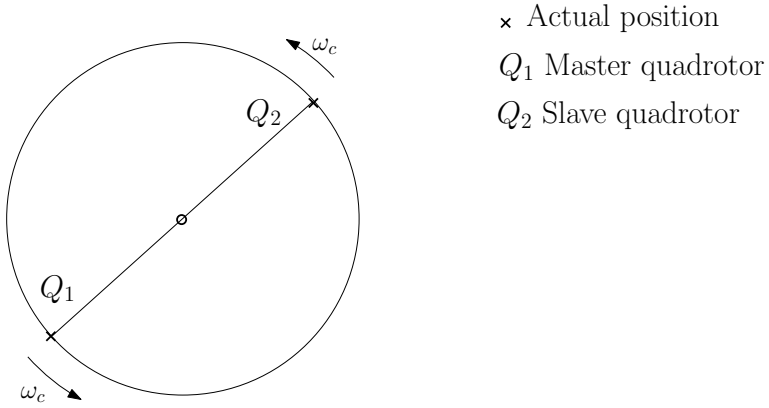


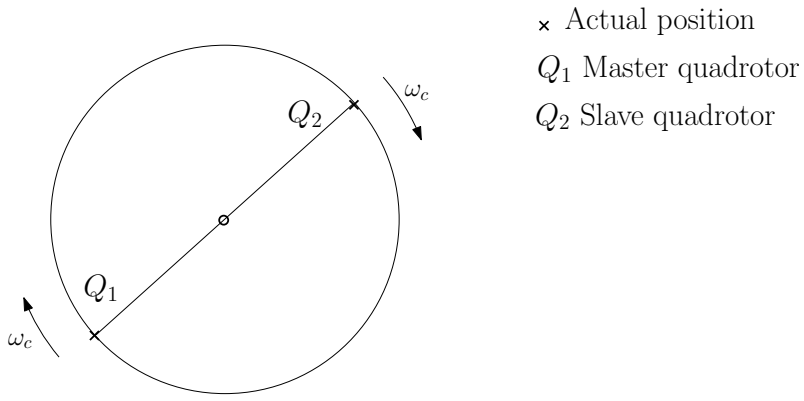
Figure 5.8: Control panel of the high level control.

5.2.3 Master/Slave control

In this section a Master/Slave task is discussed. Master/slave is a model for a communication protocol in which one quadrotor (known as the master) controls one or more other quadrotors (known as slaves). Once the Master/Slave relationship is established, the direction of control is always from the master to the slave(s). In this application, one quadrotor acts as master, and one other quadrotor as slave. The aim is that both quadrotors have to fly in diametrically opposite position around the circumference using the position and velocity control and when the master quadrotor changes direction, the slave quadrotor changes direction as well. To implement this task, the high level cooperative control gives the desired trajectory to the master quadrotor using the MATLAB scripts. Instead the high level cooperative control gives the desired trajectory to the slave quadrotor using the position of the master quadrotor. To clarify this task see Figure 5.9.



(a) The master quadrotor is moving in counterclockwise direction.



(b) The master quadrotor is moving in clockwise direction.

Figure 5.9: Master/Slave behaviour.

5.3 Real world results

To show the results for the real world application, a software called Rviz (ROS visualization) is used. Rviz is a 3D visualizer for displaying the quadrotor's behaviour.

5.3.1 Single quadrotor

The desired path for the quadrotor is a circumference with radius 1.5 meters. Figure 5.10 shows the ideal and the real trajectory using Rviz.

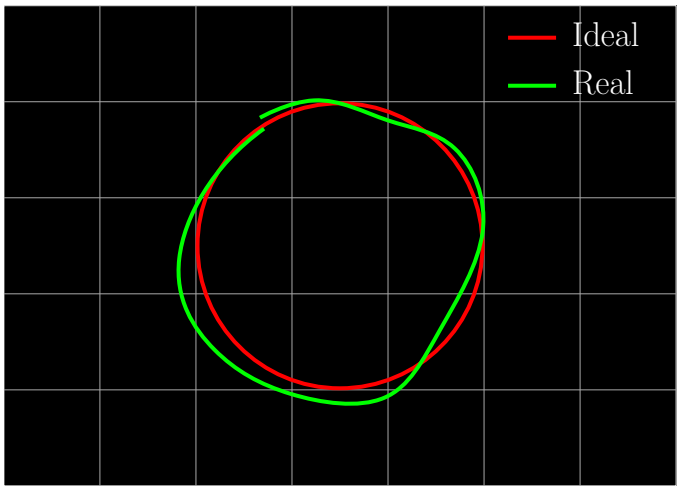


Figure 5.10: Single quadrotor: high velocity.

In the previous figure the speed of the quadrotor is high, consequently the accuracy is not elevated. Figure 5.11 shows the same path with low velocity.

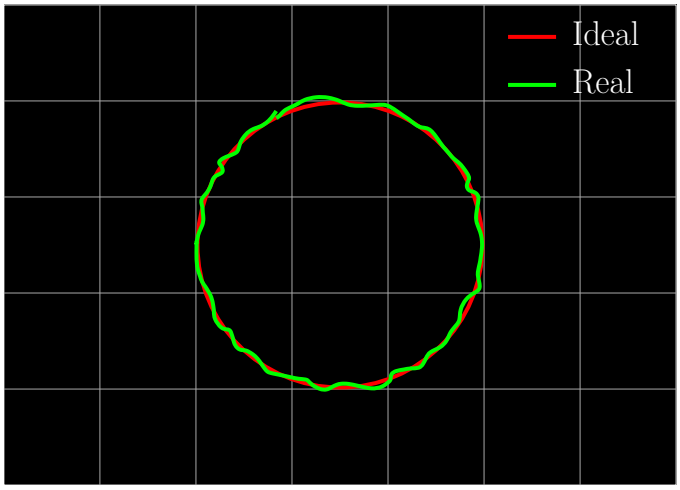


Figure 5.11: Single quadrotor: low velocity.

Notice that the accuracy is better when the quadrotor's velocity is low.
An explicative video is linked here: https://youtu.be/Tp344ig830g .

5.3.2 More quadrotors-circumference

The aim of the first task is that both quadrotors fly in diametrically opposite position on the circumference. Figure 5.12 shows the result with two quadrotors (quadrotor Q_1 and quadrotor Q_2) with low velocity.

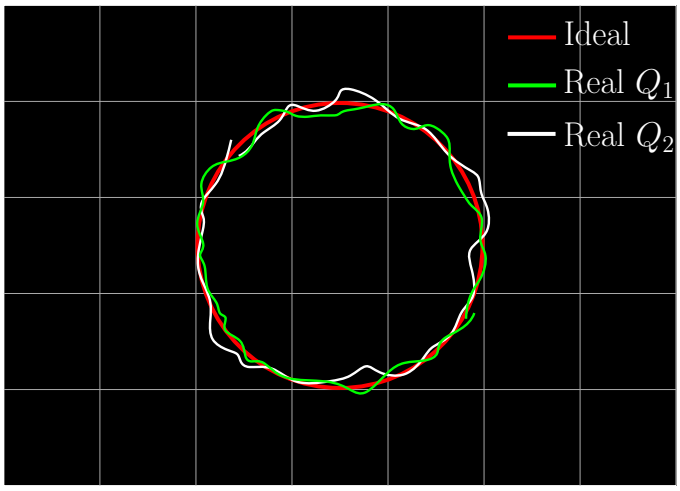


Figure 5.12: Two quadrotors: low velocity.

An explicative video about this task is linked here: https://youtu.be/8UqyTNc39dU .

The aim of the second task is about the Master/Slave bahaviour. The quadrotor Q_1 is the master and quadrotor Q_2 is the slave. Figure 5.13 shows the result using Rviz.

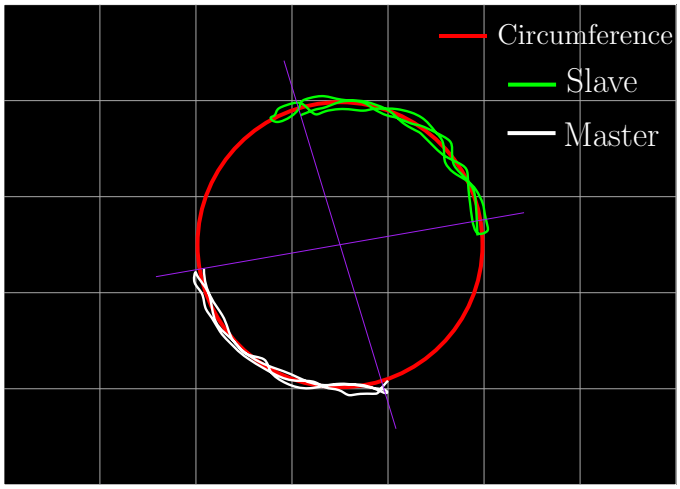


Figure 5.13: Two quadrotors: master/Slave.

Notice that when the quadrotor Q_1 (Master) changes direction, the quadrotor Q_2 (Slave) changes direction as well but with a little time delay.
An explicative video about this task is linked here: https://youtu.be/Dp-KZjgA98c .

Chapter 6

Human-quadrotor interaction

In this section several human-quadrotor interaction tasks will be described. The testbed is composed by:

- the projector;
- twelve infrared cameras;
- a wooden structure called *Gun*;
- an human;
- a quadrotor;
- a safety net that protects the people in the room when the quadrotors fly.

Using SketchUp software, Figure 6.1 shows the testbed.

Figure 6.1: Testbed: Human-quadrotor interaction.

The Gun structure is shown in Figure 6.2.

Figure 6.2: The *Gun* used in the human-quadrotor interaction tasks.

The small gray spherical bodies attached to the gun are the markers that allow the mocap system to track the position and orientation of the gun (see Section 3.3).

6.1 Gun for motion in cylindrical coordinates

This task allows the user to control the quadrotor using cylindrical coordinates; thus, moving the Gun along x, y, z axes, and rotate it around z axis, the quadrotor will move accordingly. A safety offset between the Gun and the quadrotor is also set. Using SketchUp software, Figure 6.3 shows the axes where the user can moves the Gun.

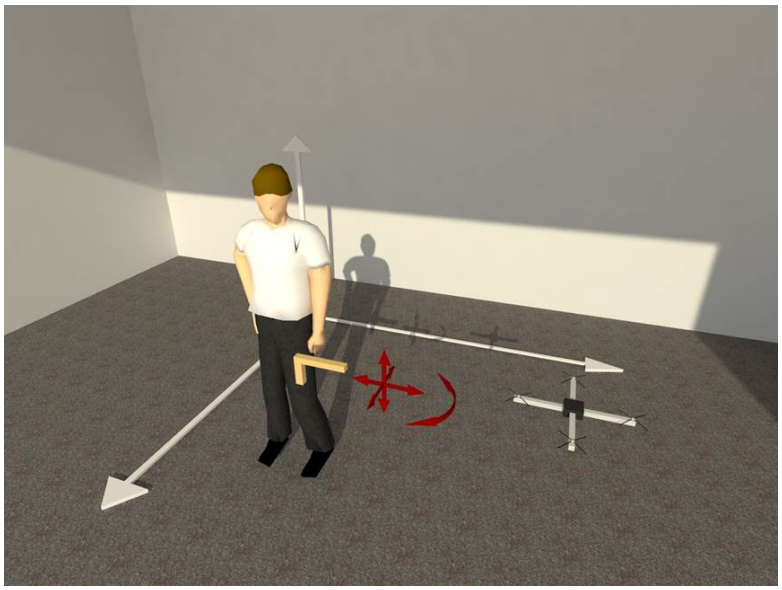
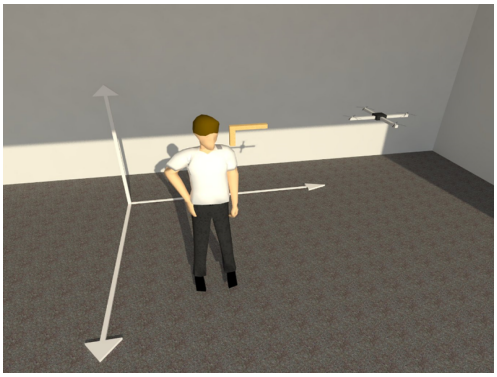
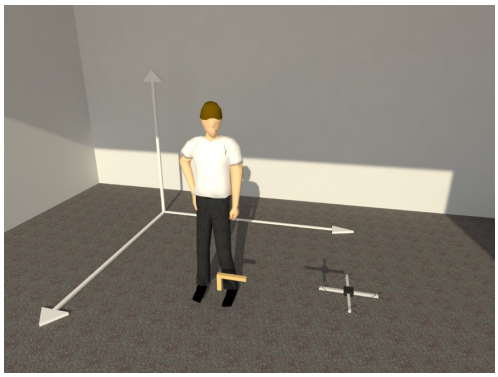


Figure 6.3: Gun for motion in cylindrical coordinates.

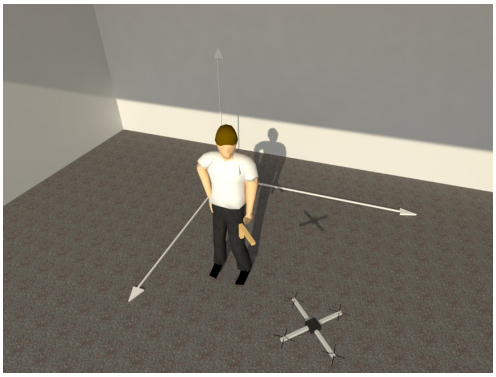
The next figures show some configurations of the Gun for this task.



(a) Movement along z axis.



(b) Movement along z axis.



(c) Movement around z axis.

Figure 6.4: Some configurations of the Gun for motion in cylindrical coordinates.

Figure 6.5 shows the application of this task in the real world.

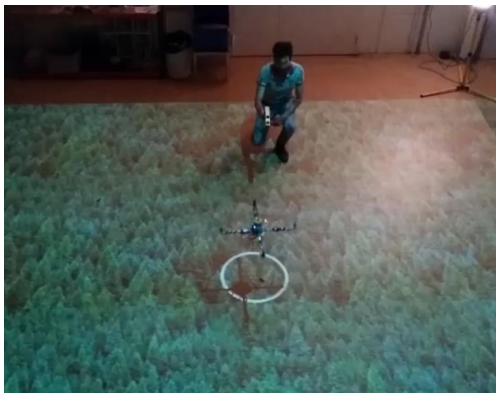


Figure 6.5: Some configurations of the Gun in the real world for motion in cylindrical coordinates.

An explicative video is linked here: https://youtu.be/5-KVyf-dlbk .

6.2 Gun for motion in spherical coordinates

This task allows the user to control the quadrotor using spherical coordinates. In this task, moving the Gun around x, y and z axes, and rotating it around the y and z axes, the quadrotor will move accordingly. This task, in addition to the previous one, allows the user to control the quadrotor by inclining the Gun. Using SketchUp, Figure 6.6 shows the axes where the user can moves the Gun.

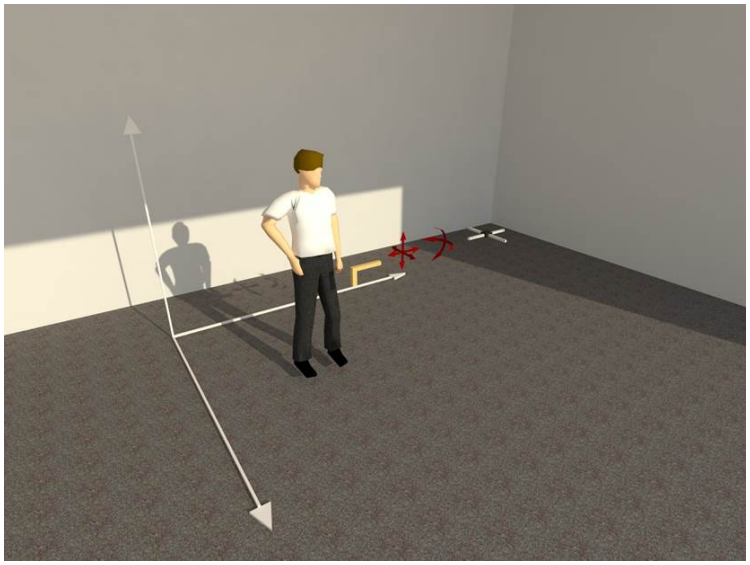
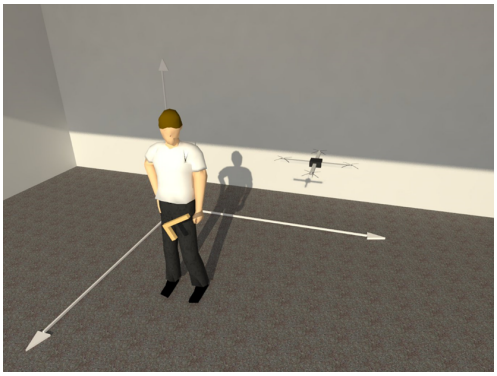
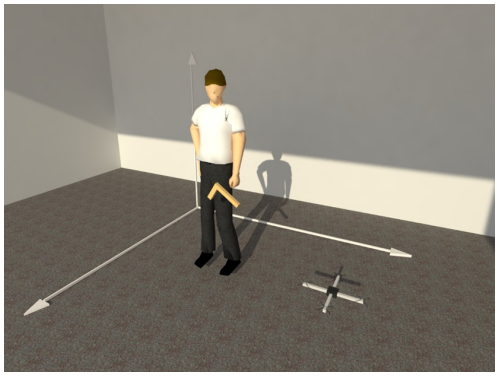


Figure 6.6: Gun for motion in spherical coordinates.

The next figures show some possible configurations of the Gun for motion in spherical coordinates .



(a) Inclining up the Gun.



(b) Inclining down the Gun.

Figure 6.7: Some configurations of the Gun for motion in spherical coordinates.

Figure 6.8 shows the application of this task in the real world.


Figure 6.8: Some configurations of the Gun in the real world for motion in spherical coordinates.

An explicative video is linked here: https://youtu.be/VLzgnlC7kHg .

6.3 Gun for projection tracking

This task uses the projector and spherical coordinates. When pointing the Gun to the ground, the controller is able to compute the coordinates of the position where the gun is pointing to, as illustrated in Figure 6.9. This point will be the reference point for the quadrotor, which will always try to follow it. In order to visualize this point with a red colour on the ground we used a projector in the flying arena, giving to it the spherical coordinates of such point. Using SketchUp, an explicative figure is shown below.

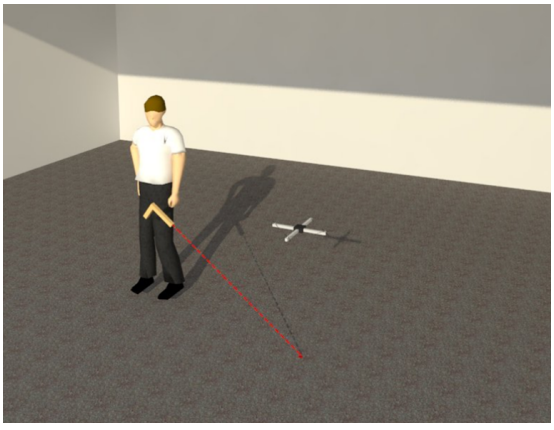


Figure 6.9: Gun for projection tracking.

The controller is able to compute the coordinates of the point where the gun is pointing to, because the Qualisys system knows the position and the orientation of the gun. Figure 6.10 shows the application of this task in the real world.

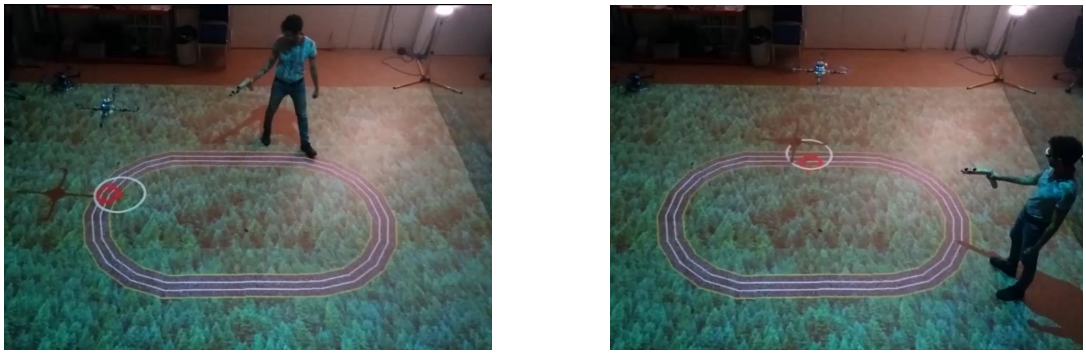
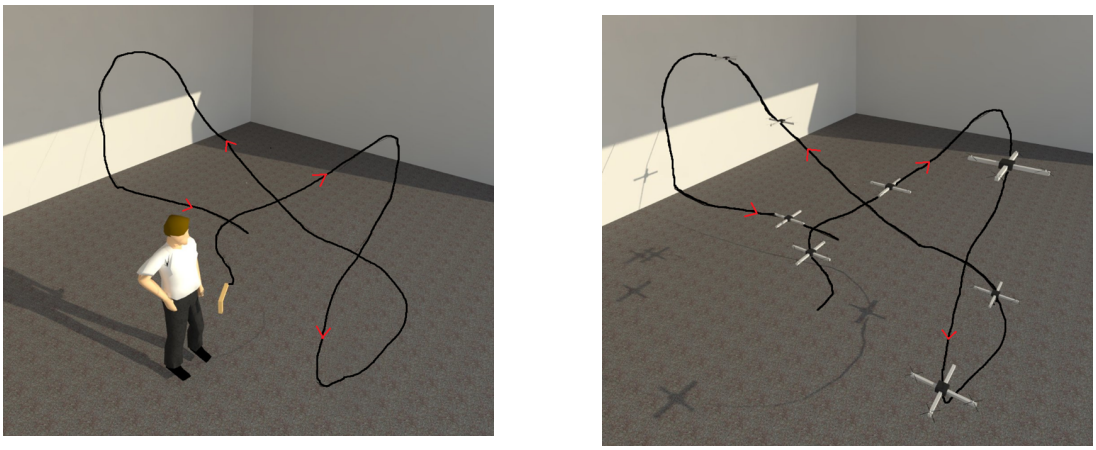


Figure 6.10: Some configurations of the Gun in the real world for projection tracking.

An explicative video is linked here: https://youtu.be/I46AiSMwOYk .

6.4 Gun for trajectory design

This task, differently from the others, does not allow to control a quadrotor with the Gun in real time, but allows instead to draw a trajectory for the quadrotor. The first step of this task is to draw with the Gun the desired path in the arena, and then start the quadrotor motion. So, a human user draws the desired trajectory with the Gun and after a desired time the quadrotor follows such trajectory. To do this we realized with LabView a program that allows to save the locations of the Gun, verify them and give them to the quadrotor, which then performs the recorded trajectory. If the Gun is moved too fast, then the quadrotor would move aggressively. To overcome this problem, a control which checks if the Gun has been previously moved too rapidly has been implemented. Using SketchUp, explicative figures are shown below.



(a) Designing the desired trajectory using the Gun.

(b) Quadrotor is following the desired trajectory.

Figure 6.11: Some configurations of the Gun for trajectory design.

An explicative video is linked here: https://youtu.be/k1DFuK-GMd4 .

Chapter 7

Conclusions and future developments

In conclusion, this thesis is based on cooperative control. This work is not only about simulation, but also on real-world application. Firstly, in this thesis, the simulated cooperative control scheme for trajectory tracking of two quadrotors is implemented. In order to visualize the behaviour of both quadrotors the Simulink 3D Animation toolbox is used. Secondly, the process of assembling a quadrotor from parts is undertaken. Furthermore a controller for a single quadrotor is designed, and implemented using LabView software, and Matlab software is used to generate the desired trajectory. Moreover, two types of cooperative control are developed: on the one hand position-based, and on the other hand position/velocity-based. Both control strategies allow the quadrotors to follow a reference trajectory while maintaining a specified formation. Finally, some developed tasks (Gun for motion in cylindrical and spherical coordinates, Gun for projection tracking, Gun for trajectory design) concern the fact that one quadrotor responds to online stimuli given by a human via a user-friendly interface.

A future project could be the application of the nonlinear control technique applied to the real world cooperative control to increase the task's accuracy. Moreover, it would be useful to improve the Qualisys system for tracking the position and the orientation of a non-rigid body, in order to develop some cooperative manipulation tasks.

Appendix A

Trajectory planning

The minimal requirement for a quadrotor is the capability to move from an initial point to a final given point. In order to avoid confusion between terms often used as synonyms, the difference between *path* and *trajectory* is now explained [14]. A path denotes the locus of points in the operational space, witch the quadrotor has to follow in the execution of the assigned motion; a path is a pure geometric of motion. On the other hand, a trajectory is a path on witch a timing law is specified, for instance in terms of velocities and/or accelerations at each point. If the quadrotor has to follow a prescribed trajectory of motion, this must be expressed analytically. It is then necessary to refer to motion primitives defining the geometric features of the path and time primitives defining the timing law on the path itself.

For the definition of a *path primitive* it is convenient to refer to a parametric description of paths in the space. Then let $\mathbf{p} \in \mathbb{R}^3$ and $\boldsymbol{f}(\sigma)$ be a continuous vector function defined in the interval $[\sigma_i, \sigma_f]$. Consider the equation

$$\mathbf{p} = \mathbf{f}(\sigma), \tag{A.1}$$

with reference to its geometric description, the sequence of values of $\mathbf{p}$ with σ varying in $[\sigma_i, \sigma_f]$ is termed *path* in the space. The equation (A.1) defines the *parametric representation* of the path Γ and the scalar σ is called parameter. As σ increases, the point $\mathbf{p}$ moves on the path in a given direction. This direction is said to be the direction induced on Γ by the parametric representation (A.1). A path is closed when $\mathbf{p}(\sigma_f) = \mathbf{p}(\sigma_i)$; otherwise it is open.

Let $\mathbf{p}_i$ be a point on the open path Γ on which a direction has been fixed. The *arc length* s of the generic point $\mathbf{p}$ is the length of the arc of Γ with extremes $\mathbf{p}$ and $\mathbf{p}_i$ if $\mathbf{p}$ follows $\mathbf{p}_i$, the opposite of this length if $\mathbf{p}$ precedes $\mathbf{p}_i$. The point $\mathbf{p}_i$ is said to be the origin of the arc length.

From the above presentation it follows that for each value of s a well determined in a different parametric representation of the path Γ:

$$\mathbf{p} = \mathbf{f}(s), \tag{A.2}$$

the range of variation of the parameter s will be the sequence of arc length associated with the points of Γ. Consider a path Γ represented by (A.2). Let $\mathbf{p}$ be a point corresponding to the arc length s. Except for special cases, $\mathbf{p}$ allows the definition of three unit vectors characterizing the path. The orientation of such vectors depends exclusively on the path geometry, while their direction depends also on the direction induced by (A.2) on the path. The first of such unit vectors is the *tangent unit vector* denoted by $\mathbf{t}$. This vector is oriented along the direction induced on the path by s. The second unit vector is the *normal unit vector* denoted by $\mathbf{n}$. This vector is oriented along the line interesting $\mathbf{p}$ at the right angle with $\mathbf{t}$ and lies in the so-called *osculating plane O* (Figure A.1); such plane is the limit position of the plane containing the unit vector $\mathbf{t}$ and the point $\mathbf{p}' \in \Gamma$ when $\mathbf{p}'$ tends to $\mathbf{p}$ along the path. The direction of $\mathbf{n}$ is so that the path Γ, in the neighborhood of $\mathbf{p}$ with respect to the plane containing $\mathbf{t}$ and normal $\mathbf{n}$, lies on the same side if $\mathbf{n}$. The third unit vector is the *bionormal unit vector* denoted by $\mathbf{b}$. This vector is so that the frame $(\mathbf{t}, \mathbf{n}, \mathbf{b})$ is right handed (Figure A.1). Notice that it is not always possible to define uniquely such a frame.

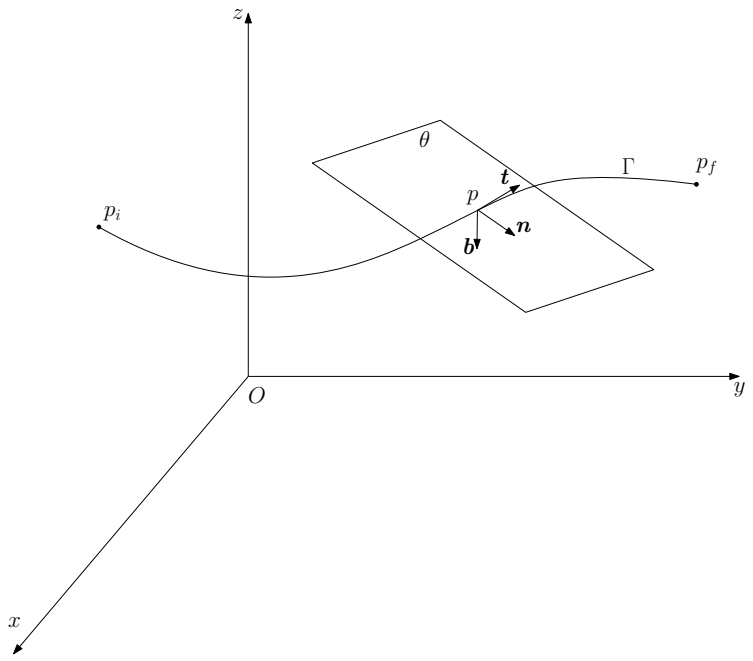


Figure A.1: Parametric representation of a path in the space [14].

It can be shown that the above three unit vectors are related by simple relations to the path representation Γ as a function of the arc length. In particular, it is

$$\begin{cases} \mathbf{t} = \frac{d\mathbf{p}}{ds} \\ \mathbf{n} = \frac{1}{\| \frac{d^2\mathbf{p}}{ds^2} \|} \frac{d^2\mathbf{p}}{ds^2} \\ \mathbf{b} = \mathbf{t} \wedge \mathbf{n} \end{cases} \tag{A.3}$$

A circular path representation is reported in the following section. It is useful for trajectory generation in the space.

A.1 Circular path

Consider a circle Γ in the space. Before deriving its parametric representation, it is necessary to introduce its significant parameters [14]. Suppose that the circle is specified by assigning (Figure A.2):

- the unit vector of the circle axis $\mathbf{r}$;
- the position vector $\mathbf{d}$ of a point along the circle axis;
- the position vector $\mathbf{p}_i$ of a point on the circle.

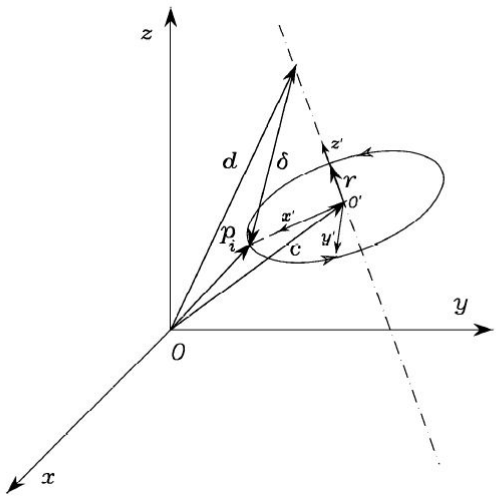


Figure A.2: Parametric representation of a circle in space [14].

With these parameters, the position vector $\mathbf{c}$ of the centre of the circle can be found. Let $\boldsymbol{\delta} = \mathbf{p_i} - \mathbf{d}$; for $\mathbf{p_i}$ not to be on the axis, i.e., for the circle not to degenerate into a point, it must be

$$\|\delta^T \mathbf{r}\| < \|\delta\|, \tag{A.4}$$

in this case it is

$$\mathbf{c} = \mathbf{d} + (\boldsymbol{\delta}^T \mathbf{r})\mathbf{r}. \tag{A.5}$$

It is now desired to find a parametric representation of the circle as a function of the arc length. Notice that this representation is very simple for a suitable choice of the reference frame. To see this, consider the frame $O' - x'y'z'$, where O' coincides with the centre of the circle, axis x' is oriented along the direction of the vector $\mathbf{p}_1\mathbf{c}$, axis z' is oriented along $\mathbf{r}$ and axis y' is chosen so as to complete a right-handed frame. When expressed in this reference, the parametric representation of the circle is

$$\mathbf{p}'(s) = \begin{bmatrix} \rho\cos(\frac{s}{\rho}) \\ \rho\sin(\frac{s}{\rho}) \\ 0 \end{bmatrix}, \tag{A.6}$$

where $\rho = \|\mathbf{p}_i - \mathbf{c}\|$ is the radius of the circle and the point $\mathbf{p}_i$ has been assumed as the origin of the arc length. For a different reference frame, the path representation becomes

$$\mathbf{p}(s) = \mathbf{c} + \mathbf{R}\mathbf{p}'(s), \tag{A.7}$$

where $\mathbf{c}$ is expressed in the frame $O - xyz$ and $\mathbf{R}$ [14] is the matrix of the frame $O' - x'y'z'$ with respect to the frame $O - xyz$. Differentiating (A.7) with respect to s gives:

$$\frac{d\mathbf{p}}{ds} = \mathbf{R} \begin{bmatrix} -\sin(\frac{s}{\rho}) \\ \cos(\frac{s}{\rho}) \\ 0 \end{bmatrix}; \tag{A.8}$$

$$\frac{d^2\mathbf{p}}{ds^2} = \mathbf{R} \begin{bmatrix} -\frac{\cos(\frac{s}{\rho})}{\rho} \\ -\frac{\sin(\frac{s}{\rho})}{\rho} \\ 0 \end{bmatrix}. \tag{A.9}$$

A.2 Timing law

Generating a trajectory means to determine a function $\mathbf{x}_q(t)$ taking the quad frame from the initial to the final position in a time t_f along a given path with a specific motion timing law. Let $\mathbf{p}_q = \mathbf{f}(s) \in \mathbb{R}^3$ a vector of the representation of the path Γ as a function of the arc length s. The arc length goes from the value $s = 0$ at $t = 0$ to the value $s = s_f$(path length) at $t = t_f$. The timing law along the path is described by the function $s(t)$. In our case we chose for $s(t)$ a sequence of linear segments with parabolic blends [14].

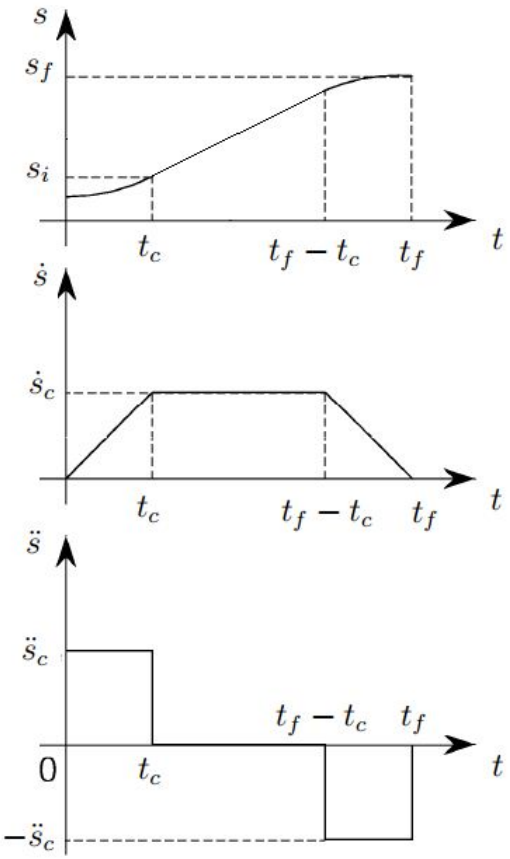


Figure A.3: Characterization of a timing law with trapezoidal velocity profile in terms of position, velocity, and acceleration.

In our case, a *trapezoidal velocity profile* is assigned, which imposes a constant acceleration in the start phase, a cruise velocity, and a constant deceleration in the arrival phase. The resulting trajectory is formed by a linear segment connected by two parabolic segments to the initial and the final positions.

Appendix B

Attachments

In this page there are the link about the simulation and the real word tasks.

Cooperative control, MATLAB simulation:
https://youtu.be/pXXz1LpPCck .

Single quadrotor, real world application:
https://youtu.be/Tp344ig830g .

More quadrotor, real world application - circumference:
https://youtu.be/8UqyTNc39dU .

More quadrotor, real world application - Master/Slave:
https://youtu.be/Dp-KZjgA98c .

Human-vehicle interaction, Gun for motion in cylindrical coordinates:
https://youtu.be/5-KVyf-dlbk .

Human-vehicle interaction, Gun for motion in spherical coordinates:
https://youtu.be/VLzgnlC7kHg .

Human-vehicle interaction, Gun for projection tracking:
https://youtu.be/I46AiSMwOYk .

Human-vehicle interaction, Gun for trajectory design:
https://youtu.be/k1DFuK-GMd4 .

List of Figures

Bibliography

[1] C.A. Rabbath, E. Gagnon, M. Lauzon. On the Cooperative Control of Multiple Unmanned Aerial Vehicles, IEEE.

[2] K. Sreenath, T. Lee, V. Kumar. Geometric Control and Differential Flatness of a Quadrotor UAV with a Cable-Suspended Load, IEEE.

[3] V. kumar. Dynamics, Control and Planning for Cooperative Manipulation of Payloads Suspended by Cables from Multiple Quadrotor Robots. Avaible: http://www.kumarrobotics.org/wp-content/uploads/2014/01/p11.pdf .

[4] http://www.diydrones.com/ .

[5] Pixhawk, https://pixhawk.org .

[6] http://planner.ardupilot.com/ .

[7] http://qualisys.com .

[8] M. Amoozadeh, Smart mobility lab manual, KTH Stockholm, 2013. Available: https://code.google.com/p/kth-smart-mobility-lab/downloads/list.

[9] Tmote sky datasheet, Moteiv corporation, 2006. Available: http://www.eecs.harvard.edu/ konrad/projects/shimmer/references/tmote-sky-datasheet.pdf.

[10] Google code page for KTH Wireless Sensor Networks. Available: https://code.google.com/p/kth-wsn/.

[11] Google code page for KTH Smart Mobility Lab. Available: https://code.google.com/p/kth-smart-mobility-lab/.

[12] Google code page for KTH Wireless Sensor Networks. Available: https://code.google.com/p/kth-wsn/.

[13] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo. Robotics. McGraw-Hill.

[14] Michael A. Henson, Dale E. Seborg. Feedback linearization control, IEEE.

[15] P. Pounds , R. Mahony. Design principles of large quadrotors for practical applications, IEEE.

[16] X. Zhang, X. Li, K. Wang. Survey of Modelling and Identification of Quadrotor Robot.

[17] D. Risberg and P. Henningsson. Networked Control of Unmanned Air Vehicles

[18] J. Swartling. Circumnavigation with a group of quadrotor helicopters

[19] D. Almeida. Event-Triggered Attitude Stabilization of a Quadcopter

[20] M. Vanin. Modeling, identification and navigation of autonomous air vehicles.

[21] M. Hehn, R. D'Andrea. Quadrocopter trajectory generation and control

[22] F.Sabatino. Quadrotor control: modeling, nonlinear control design, and simulation.