DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2016

Multiagent cooperative coverage control

MARIO SPOSATO

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING

KTH Electrical Engineering

Multiagent cooperative coverage control

MARIO SPOSATO

Stockholm 02-25-16

TRITA-EE 2016:065

If you put your mind on it, you can accomplish anything.

Dr. Emmett L. Brown

Heavier-than-air flying machines are impossible

Lord Kelvin

Are you sure? Maybe is the calibration!

Mario, Riccardo, Manuel

Abstract

In this work, the problem of deploying a team of mobile sensing agents to provide coverage of an environment is addressed. We propose a novel distributed algorithm to generate a sequence of waypoints for each agent, based on intermittent communication between the agents. The algorithm is shown to converge to an equilibrium configuration, while a measure of the environment coverage is shown to be monotonically nondecreasing. To fulfill the task of moving the agents to the designated waypoints, we develop a nonlinear control algorithm based on backstepping, as well as a path planning strategy that uses potential field navigation and collision avoidance. All the proposed algorithms are tested in a simulated environment and on real-world aerial robots.

Acknowledgements

For the development of this work, I feel the need to be grateful to several persons. First, I want to thank Professors Karl Henrik Johansson and Dimos Dimarogonas at KTH, for letting me work in such a stimulating environment, side by side with amazing minds, without making me feel out of place. A heartfelt thanks goes to Dr. Antonio Adaldo. I can say, beyond any doubt, that without your generous help I would have end up crying in a corner of the SML Lab for the last five months. I hope to become even the half of the fantastic person you are, and thanks for giving me the opportunity to achieve something that I've only dreamt of. Next, I want to thank my favourite research engineer, Rui, for always be so nice, funny and helpful. The half of the aforementioned achievement is thanks to you. You will be the best Ph.D. candidate the SML has ever seen! Last but not least, I want to thank Riccardo and Manuel for the wonderful experience we share. I grow very fond of you, thanks for all the help, the laugh and the great time spent together! I hope we will meet again.

I've clearly forgot someone (not on purpose) and my words can't really describe how grateful I am for everything happened to me during this adventure. I will remember every single moment and make precious every lesson learnt. Tack så mycket!

Ringraziamenti

Per questo lavoro di tesi sento il dovere di ringraziare prima di tutti il Professor Mario Di Bernardo, per il suo inestimabile aiuto nel permettermi di cominciare questa avventura. È sempre stato disponibile con preziosi consigli e paziente, permettendomi di lavorare in base ai miei limiti personali ma aiutandomi allo stesso tempo a superarli.

Un sentito ringraziamento alla mia famiglia, per avermi costantemente supportato e sopportato, per avermi sempre spinto nella direzione che da solo decidevo di seguire, anche a costo di qualche sacrificio. Questi risultati sono anche vostri, perché non esisterebbero senza di voi.

Poi devo ringraziare Te, Mia. Avrei bisogno di una seconda tesi per ringraziarti di ogni momento, della pazienza, della sopportazione, della dolcezza, della forza di volontà dimostrata in questi 5 mesi e in questi 7 anni, di ogni singolo sorriso, anche quelli a denti stretti e con gli occhi lucidi, della tua mano nella mia anche a 2558 Km di distanza. E soprattutto devo ringraziarti perché quando non mi fido di me, devo fidarmi di te. Se mi riuscirà di essere la persona migliore che posso, sarà principalmente merito tuo. Grazie.

Infine, ma giusto perché così è venuta, voglio ringraziare quella serie di fantastici personaggi che ho avuto la fortuna di incontrare in questi anni a Napoli. Ale, Piggi, Daddi, Andrea e Chiara, siete il motivo principale (al pari delle frittatine di Di Matteo) per il quale credo che Napoli sia la città più bella del mondo. Grazie a voi, gli anni più stressanti della mia carriera universitaria sono diventati quelli con i ricordi più belli e le esperienze più significative che ho (e anche quelli più grassi, credo). Un sentito grazie anche ai nostri ragazzi in India.

Un grazie a tutti quelli che non ho ringraziato singolarmente, e perdonatemi, ma la mia proverbiale e perfetta scelta di tempi per il completamento di questo lavoro hanno accelerato fortemente il tipico decadimento mentale di cui è soggetto ogni studente di ingegneria, quindi non è cattiveria, è vecchiaia. Grazie assaje!

Contents

Nomenclature

α, β, γ, k_1, k_2 Gains for the backstepping controller.

B Body reference frame.

$\boldsymbol{e_p}$ Position error.

$\boldsymbol{e_v}$ Velocity error.

$\mathbf{f}$ Attractive term of the path planner layer.

$\mathbf{g}$ Repulsive term of the path planner layer.

$\hat{\boldsymbol{n}}$ Axis of B reference frame orthogonal to the body of the quadcopter.

$\hat{\boldsymbol{n}}^*$ Desired ϕ, ϑ attitude of the quadcopter.

$\boldsymbol{a}^*$ Desired acceleration as defined in the path planner layer.

$\boldsymbol{\omega}$ Angular velocity.

$\boldsymbol{\omega}^*$ Desired angular velocity.

$\boldsymbol{p}$ The position of the quadcopter center of mass.

$\boldsymbol{p}_{\text{obs1}\ldots\text{obsN}}$ Position of the others object.

$\boldsymbol{p}_{\text{self}}$ Position of the quadcopter as used in the path planner layer.

$\boldsymbol{s}$ Position of a point in $\mathbb{R}^2$.

$\boldsymbol{s}_k$ Landmark k.

$\boldsymbol{u}$ Second virtual control input.

$\boldsymbol{v}$ First virtual control input.

$\hat{\boldsymbol{z}}$ Axis of W reference frame orthogonal to the orizontal plane.

$\boldsymbol{\varepsilon_1}$ First backstepping error.

$\boldsymbol{\varepsilon_2}$ Second backstepping error.

$\boldsymbol{e_\vartheta}$ Attitude error.

$\mathbb{R}^N$ N-dimensional Euclidian space.

$\mathbb{R}_+$ Set of positive real numbers.

$\mathcal{V}(\mathbf{S})$ Voronoi configuration for points in $\mathbf{S}$.

Ω Subset of $\mathbb{R}^2$.

ψ Yaw angle.

$\mathcal{A}$ Set of agents.

$\mathcal{P}$ Set of all the possible ordered subsets of $\mathbf{P}$.

$\mathbf{P}$ Set of disjointed partition of $\mathbf{S}$.

$\mathbf{Q}_i$ Set of available agents as seen by agent i.

$\mathbf{S}$ Ordered set of landmarks $\boldsymbol{s}_k$.

$\mathbf{T}_{cs}$, $\mathbf{T}_{sc}$ Set of landmarks to exchange in a trade.

σ Tolerance for trading and pose optimization algorithms.

τ Throttle value.

T Trust generated by the propellers.

φ Roll angle.

ς Threshold of the repulsive term in the path planner.

ϑ Pitch angle.

W World reference frame.

$\boldsymbol{w}_\mathrm{p}$ Waypoint for the quadcopter.

A_i^* Optimal pose of the agent i.

A_i Transformation matrix of the pose of the agent i.

a_i The agent i.

C Coverage score.

g Gravity acceleration.

k_ψ Gain for the yaw controller.

k_P, k_D, k_ϑ Final controller gains.

$k_{\boldsymbol{w}_\mathrm{p}}$, $k_{\boldsymbol{v}}$, k_{rep} Gains of the path planner.

m Mass of the quadcopter.

R Rotational matrix between W and B

$S(\boldsymbol{\omega})$ Skew-symmetric matrix.

$SE(2)$ Special Euclidian two-dimensional space.

u_ψ Controller input for the yaw angle.

1 Introduction

Aim of the present work is to develop an algorithm to autonomously deploy a team of aerial robots to collect information over a given environment, by means of vision-inspired sensors. Such problem is well known in the literature as the *coverage* problem. For our purpose, a team of quadcopters will be used as mobile sensors. In order to implement an algorithm to accomplish such task, in this work we follow a step-by-step approach, by first dealing with the physics that underlies the flight of quadcopters, then developing a control strategy to drive the quadcopters to a given position, or make them follow a reference trajectory. Since the mission will involves several quadcopters, some safety measures are needed to make the quadcopters fly without interfering with each other - or worse, colliding during the navigation. To this aim, we develop a path planning strategy to make the quadcopters aware of the surrounding environment. Finally, we discuss our approach to solve the coverage problem with mobile sensing agents, and we give theoretical and experimental results implementing our algorithm on the real quadcopters.

In the following sections, we give an overview on several research works and applications involving quadcopters, as well as a deeper insight on related works on coverage.

1.1 Literature Review

In the last decade, much of the research effort in the automatic control and robotic fields has been directed toward the developing of unmanned vehicles that are capable of accomplish complex tasks in unstructured environments, with the aim of substituting the human intervention in hazardous situations. The most common design, on which a lot of research has focused, is the quadcopter [1–3]. A quadcopter is an unmanned aircraft, with four propellers directly attached to small electrical motors, generally along parallel axis. This structure makes the quadcopters capable of vertical take-off and landing, stationary and low speed flight, and generally gives them a high payload/weight ratio, along with easiness in building and control compared to other aerial vehicles. All this attention from the research community has led in turn also to the spreading of such technology from the laboratories onto the shelf of the consumer-electronic products, lowering the prices of the components and giving even more reasons for using quadcopters in research.

A complete survey of all the results in control theory, automation, robotics and bio-inspired engineering that involve quadcopters goes beyond the scope of this work, thus here we only give some hint on the state-of-art of research related to quadcopters. For a wider insight, we relate to [4] and references therein.

Several works focus their attention to develop novel control strategy for the motion control of the quadcopter. In [5–8], classic non-linear control techniques are exploited

on the quadcopters, with the aim of stabilizing the dynamics and reducing the energy effort requested to the motors. In [9–11], real-time attitude estimation of the quadcopter subject to sensor noise is taken into account in the proposed controller. Finally, [12, 13] develop a controller that is able of stabilize the quadcopter under wind disturbances.

Another trending topic relating to the simple motion and trajectory tracking of the quadcopter is the *path planning*, on which remarkable results are [14–16]. Path planning strategies often implement collision avoidance behaviour, as well treated in [17–19].

With the decreasing of the computational effort of the control, and the technological achievements in electronic components' miniaturization, an increasing number of research effort are conducted on the so-called *flocking* and *formation control* of large number of quadcopters, by taking inspiration from the flocking behaviour of several animal species (such as bird, fish and insects) in a relatively new branch of studies named *network control*. A comprehensive introduction to such theme can be found in [20]. On this field, is worth mentioning [21] on cooperative control among robotic agents, with an algebraical approach to the problem, [22], that address the problem of formation control of a swarm of quadcopters with shape-changing specification, [23], that consider the flocking behaviour on very large group of agents, and [24], in which the authors were able to mimic with quadcopters the *nearest neighbor rule*, a typical behaviour of the school of fish and flock of birds, in which each agent change its position and heading according to the behaviour of the nearest neighbor in the flock. A very interesting evolution of the network control, that is also the main topic of this work, lies in the *multiagent cooperative control*, in which a decentralized control strategy is develop in order to make the quadcopters cooperatively accomplish a certain task. Meaningful examples are [25, 26], both addressing the problem of cooperative load-transportation with quadcopters, and [27] in which a coordinate path-planning strategy makes it possible for a team of aerial vehicles to assemble a tensile structures.

1.2 Related works

As aforementioned, the coverage problem is just one of the many applications of multiagent cooperative control. Nonetheless, studies on this topic are increasing, due to the wide variety of applications in which sensors must be placed in a certain area to retrieve information about the environment. Let us think of an inaccessible natural ecosystem that we want to monitor, or what remains after a natural disaster, or a dismantled nuclear plant. In all these cases, the environment of interest is unreachable for the human operators, or even hazardous; for this reason, a way to automatically deploy a team of mobile agents, with sensing capabilities must be defined. Many approaches can be found in literature regarding this problem, several of which start from the classical solution that consider *Voronoi tessellation* and the Lloyd algorithm [28]. See [29–31], to name a few.

In tackling a coverage problem, a model of the sensors used needs to be elaborated, since the definition of how the surroundings are perceived by the mobile sensing agent strongly affect the evolution of the algorithm. Most of the existing results on coverage

consider sensors with symmetric, omnidirectional *field of view* [18,32], and only recently, agents with anisotropic [33–36] and vision based [37] sensors have been considered. The major challenge in the real-world implementation of coverage algorithms lies in the communication and information exchange among the agents. For that, [38] proposes a gossip-based interaction strategy, in which only short-range, asynchronous and unreliable communication between nearby robots are needed. Moreover, the same works proposes to abstract the environment into a finite set of points, which can either correspond to a particular points of interest in the environment to cover, or represent a complete discretization of the environment itself. In the present work, we will propose a novel approach to the coverage problem, that considers agents with anisotropic, vision based sensors, communicating with a gossip-like strategy, in order to accomplish the mission on a discretized environment.

1.3 Simulated and experimental setup

Throughout this work, we will present simulations and experiments related to the theoretical results achieved. In order to make the results of the simulations and of the experiments more readable, we give here the general setup that these were conducted with.

The basic component of each simulation and experiment is ROS(*Robot Operative System*) [39]. ROS, as the acronym suggests, is an operative system created to help the design and implementation of software for robotic systems. Using ROS for programming a robotic platform gives two main advantages. First of all, as an open source platform, most of the software needed for interfacing a certain component - namely a sensor, a motor, a controller and so on - with the physical robot, is already available and ready to use. Secondly, ROS makes available powerful communication primitives between the several component of a system, leaving the user only the need to define the content of the messages, and taking care of all the hardware/low level software setting of the interaction. The basic communication primitives that ROS implements are *Services* and *Topics*. In this work we used those ROS structures, coded with PYTHON [40] under UBUNTU [41] machines.

For the simulations, we used ROTORS [42], which is a real world simulator for UAVs with a physic engine. This simulator is also written using ROS primitives, so that it is easily used within a robotic project involving ROS. Namely, ROTORSmakes it possible to monitor the dynamic behaviour of a quadcopter under user-defined control input, by actually simulating the motion of the quadcopter subject to the dynamics of a rigid body. We used ROTORSin Chapters 3-6 for testing the algorithms presented therein. Some of the simulations results were also plotted using a MATLAB/SIMULINK script for them to be easier to be read.

For the experimental part, we used a commercial quadcopter named IRIS+ of 3D ROBOTICS [43]. The quadcopter is equipped with an onboard control unit (which is also referred as *autopilot*) named PIXHAWK. This control unit has an inner loop control structure that makes possible for the user to directly define the desired *trust* value T^*

Figure 1.1: AEROWORKS 2020 logo

and the angular velocity vector $\boldsymbol{\omega}^*$, by taking care of translating these references into input signals for the motors. This in turn makes it easier to model the quadcopter, as well as developing of an outer loop controller. The quadcopter can be controlled from a computer using a USB-to-radio transmitter, and the radio communication is handled by a protocol named MAVLINK, which comes with a built-in extension for interfacing ROS named MAVROS [44]. Finally, the position of the real quadcopter was measured by means of a motion capture system called QUALISYS [45], that uses 12 infrared cameras to track the position of the quadcopter. The motion capture can be interfaced with ROS as well, and it can *publish* the pose of any marked object on a defined topic.

All the experiments present in this work took place in the SML (Smart Mobility Lab) [46] facility at KTH University.

1.4 Context

This work is part of the theoretical developments that underlie an European project called AEROWORKS 2020. The project, funded by the European Union, has the aim to develop a novel aerial robotic team of agents, with dexterous manipulation capabilities, with the purpose of autonomously conduct cooperative infrastructure inspection and maintenance task, exploiting co-manipulation and intelligent interaction features. Further details about AEROWORKS 2020 can be found at [47].

1.5 Thesis Outline

In the next chapters, we will cover all the steps required to face the implementation of a coverage algorithm on real quadcopters. The work is organized as follows.

In Chapter 2, we will introduce the mathematical description of the dynamics of the quadcopter, starting from some general assumptions. On the model presented, we will then derive a controller via a mechanical procedure, with which we will be able to drive

the quadcopter to a reference position or follow a pre-planned trajectory, as well as a controller for the heading of the quadcopter. Then we test the developed controller with simulations and experiments and show the results.

In Chapter 3, we address the problem of having to fly multiple quadcopters in the same area, with limited space. Thus, we add an extra layer of control by implementing a *collision avoidance* mechanism that generates online, collision-free, trajectory. With this extra layer, each quadcopter can be aware of the current position of other nearby quadcopters and change its position to react to a possible collision scenario. The implementation of this layer makes also the quadcopters follow a smoother trajectory which, in turn, further enhances the flight stability. Simulations and experiments were also made for testing this extra layer of control and in the end of the chapter we give some results.

In Chapter 4, we finally tackle the coverage problem, which is the main objective of this work. We first give a formal definition of a coverage mission, then we settle some definition needed to mathematically describe the objective of the mission, and finally we present our approach to the solution of such problem. Proof that our approach makes the system converge to a optimal solution is given at the end of the chapter.

In Chapter 5, we show the results of the simulation of a coverage mission, as well as an experiment involving a real quadcopter, using the experimental setup described above.

In Chapter 6 we summarize the general results achieved throughout this work, and we give insights on the possible future developments.

2 UAVs Model and Control

In the previous chapter, we described the coverage problem. As we stated there, a mobile sensor is requested to fulfill the task of moving around the targeted area and enhance the surveillance with respect to generic constraints. Thus a system that is able to physically navigate through environments in an autonomous fashion, given a set of way-points and high level input, has to be chosen. In this work, we choose to use quadcopters as the mobile sensing agents needed by the real world implementation of the coverage task described in Chapter 1. In this chapter, a dynamical model of the quadcopter is derived, as well as a non-linear control algorithm for the quadcopter to be able to reach a point in the space or follow a trajectory.

2.1 Modelling

In this section, we are interested in describing the motion of the quadcopter, modeled as a rigid body in the 3D space, in terms of position and orientation. In the following, the position and the orientation of a quadcopter shall be collectively referred to as the *pose* of the quadcopter. To this aim, let us consider two reference frames $W = (\boldsymbol{O_W}, \hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$ and $B = (\boldsymbol{O_B}, \hat{\boldsymbol{l}}, \hat{\boldsymbol{m}}, \hat{\boldsymbol{n}})$, The frame W, which we shall call the *world frame* is inertial, while the frame B, which we shall call the *body frame*, is attached to the quadcopter. In particular, we let the origin of B be attached to the quadcopter's center of mass, and the third axis $\hat{\boldsymbol{n}}$ of B be orthogonal to the plane containing the quadcopter's blades, as illustrated in Figure 2.1. Hence, the pose of the quadcopter is defined by the transformation between

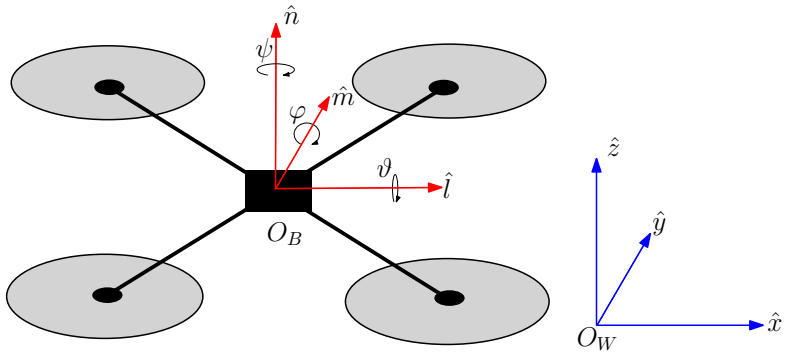


Figure 2.1: Quadcopter sketch with reference frames outlined

the frames $W$and B. Namely, the position of the quadcopter is defined by

$$\boldsymbol{p} = \boldsymbol{O_B} - \boldsymbol{O_W} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \in \mathbb{R}^3,$$

while the orientation is defined by three angles φ, ϑ, ψ which describe the amplitude of rotation along the three body axes, as in Figure 2.1. In order to describe the rotation of the quadcopter in the world frame, we use the following *rotation matrix* (where $s_\vartheta = \sin\vartheta$, $c_\vartheta = \cos\vartheta$ and so on)

$$R = \begin{bmatrix} c_\varphi c_\vartheta & c_\varphi s_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta c_\psi + s_\varphi s_\psi \\ s_\varphi c_\vartheta & s_\varphi s_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta c_\psi - c_\varphi s_\psi \\ -s_\vartheta & c_\vartheta s_\psi & c_\vartheta c_\psi \end{bmatrix}.$$

Therefore, we can describe the rotation of the rigid body with

$$\begin{bmatrix} \hat{\boldsymbol{l}} \\ \hat{\boldsymbol{m}} \\ \hat{\boldsymbol{n}} \end{bmatrix} = R \begin{bmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{y}} \\ \hat{\boldsymbol{z}} \end{bmatrix}.$$

Using Newton's second law for rigid bodies, the motion of the quadcopter can be described as

$$\begin{cases} m\ddot{\boldsymbol{p}} = \tau\hat{\boldsymbol{n}} - mg\hat{\boldsymbol{z}}, \\ \dot{R} = RS(\boldsymbol{\omega}), \end{cases} \tag{2.1}$$

where

- $m \in \mathbb{R}_+$ is the mass of the quadcopter,
- $\tau \in \mathbb{R}_+$ is the force generated by the propellers of the quadcopter,
- $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T \in \mathbb{R}^3$ is the angular velocity of the quadcopter,
- $S(\boldsymbol{\omega})$ is the skew-symmetric matrix that induces the cross product by $\boldsymbol{\omega}$, namely $S(\boldsymbol{\omega}) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \in \mathfrak{so}(3)$; and $S(\boldsymbol{\omega})v = \boldsymbol{\omega} \times v$ for all $v \in \mathbb{R}^3$.

From the definition of R and $\hat{\boldsymbol{n}}$, we have $\hat{\boldsymbol{n}} = R\hat{\boldsymbol{z}}$, and thus

$$\begin{aligned} \dot{\hat{\boldsymbol{n}}} &= \dot{R}\hat{\boldsymbol{z}} \\ &= RS(\boldsymbol{\omega})\hat{\boldsymbol{z}} \\ &= S(R\boldsymbol{\omega})R\hat{\boldsymbol{z}} \\ &= S(R\boldsymbol{\omega})\hat{\boldsymbol{n}} \\ &= S(\boldsymbol{\omega})\hat{\boldsymbol{n}}. \end{aligned}$$

In (2.1) and in what follows, all the quantities are time-dependent, but we omit this dependency in the equations, for the sake of notational simplicity. In this work, we

take the *normalized thrust* $T = \tau/m$ and the *angular velocity* $\boldsymbol{\omega}$ as our control inputs. Therefore, we rewrite (2.1) as

$$\begin{cases} \ddot{\boldsymbol{p}} = T\hat{\boldsymbol{n}} - g\hat{\boldsymbol{z}}, \\ \dot{R} = RS(\boldsymbol{\omega}). \end{cases} \tag{2.2}$$

The reason for choosing this particular model is that it corresponds to the control interface offered by the on-board control unit on the real quads that we will use for the experiments.

2.2 Control

In this section, we present a control strategy that makes the quadcopter reach any arbitrary point in space asymptotically, as well asymptotically track a reference trajectory. The control objective can be expressed as

$$\lim_{t\to\infty} \|\boldsymbol{p}(t) - \boldsymbol{p}^*(t)\| = 0,$$

where $\boldsymbol{p}^*(t)$ is a given reference trajectory. For analytical purposes, we require that $\boldsymbol{p}^*(t)$ is differentiable three times and that $\|\ddot{\boldsymbol{p}}(t)\| < g$ for all $t \geq 0$. Thus, this control strategy can be used, for example, to drive the quadcopters to the waypoints generated during a coverage mission. Therefore we are interested in solving a position and velocity tracking problem. To this aim, we rewrite (2.2) in an equivalent state-space model in terms of error vectors

$$\begin{cases} \dot{\boldsymbol{e}}_{\boldsymbol{p}} = \boldsymbol{e}_{\boldsymbol{v}}, \\ \dot{\boldsymbol{e}}_{\boldsymbol{v}} = T\hat{\boldsymbol{n}} - g\hat{\boldsymbol{z}} - \ddot{\boldsymbol{p}}^*, \\ \dot{\hat{\boldsymbol{n}}} = S(\boldsymbol{\omega})\hat{\boldsymbol{n}}, \end{cases} \tag{2.3}$$

where $\boldsymbol{e}_{\boldsymbol{p}} = \boldsymbol{p} - \boldsymbol{p}^*$, $\boldsymbol{e}_{\boldsymbol{v}} = \dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}^*$, and $\boldsymbol{p}^*$, $\dot{\boldsymbol{p}}^*$ are the desired position and velocity. For (2.3) to be equivalent to model (2.2), we add an equation for the dynamics of the *yaw* angle. This additional equation is needed because the model in (2.3) does not contain the dynamics of the yaw. It can be notice, indeed, that $\hat{\boldsymbol{n}}$ only defines the orientation of the quadcopter in terms of *pitch* (φ) and *roll* (ϑ) angles, because the dynamic of $\hat{\boldsymbol{n}}$ (third equation in (2.3)) is not affected by the rotation of the quadcopter around $\hat{\boldsymbol{n}}$ itself. Physically, this translates into noticing that, according to (2.2), a quadcopter can fly towards a point in space while maintaining any desired yaw angle, without this affecting the flight dynamics. The dynamics of the yaw is defined by

$$\dot{\psi} = \boldsymbol{\omega}^T\hat{\boldsymbol{n}}.$$

We will discuss the design of a controller for tracking a yaw reference trajectory later on in this section. In this section, our aim is to stabilize $\boldsymbol{e}_{\boldsymbol{p}}$ and $\boldsymbol{e}_{\boldsymbol{v}}$ to $\mathbf{0}_3 = [0,0,0]^T$, i.e., to make the origin of the state-space ($\boldsymbol{e}_{\boldsymbol{p}}$, $\boldsymbol{e}_{\boldsymbol{v}}$) asymptotically stable. This control objective suggests a Lyapunov-based control approach, namely finding a *candidate Lyapunov function* (CLF) and designing a control input for the system that makes the time derivative of such function negative definite whenever the error is nonzero. Finding a

CLF is usually tricky, thus we propose the use of the so-called *backstepping control* [48], which can be summarized as follows.

First consider the single integrator in (2.3)

$$\dot{\boldsymbol{e}}_p = \boldsymbol{e_v}. \tag{2.4}$$

We will stabilize (2.4) with the virtual input $\boldsymbol{v}$ as

$$\begin{aligned}\dot{\boldsymbol{e}}_p &= \boldsymbol{v} + \boldsymbol{e_v} - \boldsymbol{v} \\ &= \boldsymbol{v} + \boldsymbol{\varepsilon_1},\end{aligned}$$

where

$$\boldsymbol{\varepsilon_1} = \boldsymbol{e_v} - \boldsymbol{v}$$

is the first backstepping error. If we assume that $\boldsymbol{\varepsilon_1} = 0$ in this step, then we have the single integrator system $\dot{\boldsymbol{e}}_p = \boldsymbol{v}$, and we can design the virtual input v by using the following CLF

$$V_1(\boldsymbol{e_p}) = \frac{1}{2}\alpha \boldsymbol{e_p}^T \boldsymbol{e_p},$$

where α is an arbitrary positive scalar. Choosing $v = -k_1 \boldsymbol{e_p}$, with $k_1 > 0$ leads the system to the locally asymptotically stable equilibrium $\boldsymbol{e_p} = 0$. In fact

$$\begin{aligned}\dot{V}_1(\boldsymbol{e_p}) &= \alpha \boldsymbol{e_p}^T \dot{\boldsymbol{e}}_p \\ &= -\alpha \boldsymbol{e_p}^T v \\ &= -\alpha k_1 \boldsymbol{e_p}^T \boldsymbol{e_p} = -W_1(\boldsymbol{e_p}) \leq 0,\end{aligned} \tag{2.5}$$

while the *real* $\dot{V}_1(\boldsymbol{e_p})$, i.e., considering $\boldsymbol{\varepsilon_1} \neq 0$, is given by

$$\dot{V}_1(\boldsymbol{e_p}) = -W_1(\boldsymbol{e_p}) + \alpha \boldsymbol{e_p}^T \boldsymbol{\varepsilon_1}.$$

In the second step, we consider the first and second equation of (2.3)

$$\begin{cases}\dot{\boldsymbol{e}}_p = \boldsymbol{e_v}, \\ \dot{\boldsymbol{e}}_v = T\hat{\boldsymbol{n}} - g\hat{\boldsymbol{z}} - \ddot{\boldsymbol{p}}^*.\end{cases} \tag{2.6}$$

To stabilize the new system (2.6), we introduce a new virtual input u

$$\begin{aligned}\dot{\boldsymbol{e}}_v &= \boldsymbol{u} - g\hat{\boldsymbol{z}} - \ddot{\boldsymbol{p}}^* + T\hat{\boldsymbol{n}} - \boldsymbol{u} \\ &= \boldsymbol{u} - g\hat{\boldsymbol{z}} - \ddot{\boldsymbol{p}}^* + \boldsymbol{\varepsilon_2},\end{aligned} \tag{2.7}$$

where $\boldsymbol{\varepsilon_2} = T\hat{\boldsymbol{n}} - \boldsymbol{u}$ is the second backstepping error. The CLF associated to system (2.6) is

$$\begin{aligned}V_2(\boldsymbol{e_p}, \boldsymbol{e_v}) = V_2(\boldsymbol{e_p}, \boldsymbol{\varepsilon_1}) &= \frac{1}{2}\alpha \boldsymbol{e_p}^T \boldsymbol{e_p} + \frac{1}{2}\beta {\boldsymbol{\varepsilon_1}}^T \boldsymbol{\varepsilon_1} \\ &= V_1(\boldsymbol{e_p}) + \frac{1}{2}\beta {\boldsymbol{\varepsilon_1}}^T \boldsymbol{\varepsilon_1},\end{aligned} \tag{2.8}$$

Differentiating (2.8) with respect to time, we have

$$\dot{V}_2(\boldsymbol{e_p}, \boldsymbol{\varepsilon_1}) = -W_1(\boldsymbol{e_p}) + \alpha \boldsymbol{e_p}^T \boldsymbol{\varepsilon_1} + \beta \boldsymbol{\varepsilon_1}^T \dot{\boldsymbol{\varepsilon}}_1, \tag{2.9}$$

Next, using (2.7), we express $\dot{\boldsymbol{\varepsilon}}_1$ as

$$\begin{aligned} \dot{\boldsymbol{\varepsilon}}_1 &= \dot{\boldsymbol{e}}_v - \dot{\boldsymbol{v}} \\ &= \boldsymbol{u} - g\hat{\boldsymbol{z}} - \ddot{\boldsymbol{p}}^* + \boldsymbol{\varepsilon_2} + k_1 \boldsymbol{e_v}. \end{aligned} \tag{2.10}$$

Substituting (2.10) in (2.9), and neglecting $\boldsymbol{\varepsilon_2}$ we have

$$\begin{aligned} \dot{V}_2(\boldsymbol{e_p}, \boldsymbol{\varepsilon_1}) &= -W_1(\boldsymbol{e_p}) + \alpha \boldsymbol{\varepsilon_1}^T \left(\boldsymbol{e_p} + \frac{\beta}{\alpha} \dot{\boldsymbol{\varepsilon}}_1 \right) \\ &= -W_1(\boldsymbol{e_p}) + \alpha \boldsymbol{\varepsilon_1}^T \left(\boldsymbol{e_p} + \frac{\beta}{\alpha} \big(\boldsymbol{u} - g\hat{\boldsymbol{z}} - \ddot{\boldsymbol{p}}^* + k_1 \boldsymbol{e_v} \big) \right), \end{aligned} \tag{2.11}$$

where we used (2.5) and (2.7). Now we need to choose $\boldsymbol{u}$ in order to make (2.11) negative definite. One option is to set

$$\boldsymbol{e_p} + \frac{\beta}{\alpha} \boldsymbol{u} + \frac{\beta}{\alpha} \big(-g\hat{\boldsymbol{z}} - \ddot{\boldsymbol{p}}^* + k_1 \boldsymbol{e_v} \big) = -k_2 \boldsymbol{\varepsilon_1},$$

which leads to

$$\begin{aligned} u &= -k_1 \boldsymbol{e_v} + g\hat{\boldsymbol{z}} + \ddot{\boldsymbol{p}}^* - \frac{\alpha}{\beta} \boldsymbol{e_p} - k_2 \frac{\alpha}{\beta} \boldsymbol{\varepsilon_1} \\ &= \ddot{\boldsymbol{p}}^* + g\hat{\boldsymbol{z}} - (\frac{\alpha}{\beta} k_1 k_2 - \frac{\alpha}{\beta}) \boldsymbol{e_p} - (k_1 + \frac{\alpha}{\beta} k_2) \boldsymbol{e_v} \\ &= \ddot{\boldsymbol{p}}^* + g\hat{\boldsymbol{z}} - k_P \boldsymbol{e_p} - k_D \boldsymbol{e_v} \quad k_2 > 0, \end{aligned} \tag{2.12}$$

The addends in (2.12) can be interpreted as follows. The term $\ddot{\boldsymbol{p}}^*$ corresponds to a feed-forward term on the desired acceleration. The term $g\hat{\boldsymbol{z}}$ is the gravity compensation of the control input, and finally the term $k_P \boldsymbol{e_p} - k_D \boldsymbol{e_v}$ is the feedback PD action. Replacing (2.12) in (2.11) yields

$$\dot{V}_2(\boldsymbol{e_p}, \boldsymbol{e_v}) = -W_1(\boldsymbol{e_p}) - k_2 \alpha \boldsymbol{\varepsilon_1}^T \boldsymbol{\varepsilon_1} = -W_2(\boldsymbol{e_p}, \boldsymbol{e_v}) \leq 0, \tag{2.13}$$

while considering $\boldsymbol{\varepsilon_2} \neq 0$ leads to

$$\begin{aligned} \dot{V}_2(\boldsymbol{e_p}, \boldsymbol{e_v}) &= -W_1(\boldsymbol{e_p}) + \alpha \boldsymbol{\varepsilon_1}^T \left(\frac{\beta}{\alpha} \boldsymbol{\varepsilon_2} - k_2 \boldsymbol{\varepsilon_1} \right) \\ &= -W_2(\boldsymbol{e_p}, \boldsymbol{e_v}) + \beta \boldsymbol{\varepsilon_1}^T \boldsymbol{\varepsilon_2}. \end{aligned}$$

Note that in (2.13) we stressed the recursive structure of the Lyapunov function created by the backstepping procedure, where the function $W_2(\boldsymbol{e_p}, \boldsymbol{e_v})$ depends explicitly on the function $W_1(\boldsymbol{e_p})$.

In the third step, we consider as non-zero the second backstepping error $\boldsymbol{\varepsilon_2}$. Then, we add the third equation of (2.3) to the backstepping procedure, and finally use the control inputs T and $\boldsymbol{\omega}$ to stabilize (2.3). To this aim, we can write

$$\begin{cases} \dot{\boldsymbol{e}}_{\boldsymbol{p}} = \boldsymbol{e}_{\boldsymbol{v}}, \\ \dot{\boldsymbol{e}}_{\boldsymbol{v}} = T\hat{\boldsymbol{n}} - g\hat{\boldsymbol{z}} - \ddot{p}^* = \boldsymbol{\varepsilon_2} + \boldsymbol{u} - g\hat{\boldsymbol{z}} - \ddot{\boldsymbol{p}}^*, \\ \dot{\hat{\boldsymbol{n}}} = S(\boldsymbol{\omega})\hat{\boldsymbol{n}}. \end{cases} \tag{2.14}$$

Now we propose to write T as the projection of $\boldsymbol{u}$ on $\hat{\boldsymbol{n}}$

$$T = \boldsymbol{u}^T\hat{\boldsymbol{n}}. \tag{2.15}$$

Using (2.15), we can rewrite (2.7) as

$$\dot{\boldsymbol{e}}_{\boldsymbol{v}} = (\boldsymbol{u}^T\hat{\boldsymbol{n}})\hat{\boldsymbol{n}} - g\hat{\boldsymbol{z}} - \ddot{\boldsymbol{p}}^*,$$

and the error $\boldsymbol{\varepsilon_2}$ as

$$\begin{aligned} \boldsymbol{\varepsilon_2} &= T\hat{\boldsymbol{n}} - \boldsymbol{u} \\ &= (\boldsymbol{u}^T\hat{\boldsymbol{n}})\hat{\boldsymbol{n}} - \boldsymbol{u} \\ &= (\hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T - \mathrm{I})\boldsymbol{u}. \end{aligned} \tag{2.16}$$

Note that our aim is to have $\boldsymbol{u}$ aligned to $\hat{\boldsymbol{n}}$, in order to have the maximum thrust T for a given magnitude of $\boldsymbol{u}$. Therefore, defining the desired attitude of the quadcopter as

$$\hat{\boldsymbol{n}}^* = \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|},$$

we can redefine (2.16) as

$$\begin{aligned} \boldsymbol{\varepsilon_2} &= (\hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T - \mathrm{I})\hat{\boldsymbol{n}}^*\|\boldsymbol{u}\| \\ &= -(\mathrm{I} - \hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T)\hat{\boldsymbol{n}}^*\|\boldsymbol{u}\| \\ &= -\boldsymbol{e}_{\boldsymbol{\vartheta}}\|\boldsymbol{u}\|, \end{aligned} \tag{2.17}$$

where $\boldsymbol{e}_{\boldsymbol{\vartheta}}$ represents the error in the attitude. It is straightforward to prove that when $\boldsymbol{u}$ and $\hat{\boldsymbol{n}}$ are aligned, $\boldsymbol{\varepsilon_2} = 0$. We must also note that the desired pose $\hat{\boldsymbol{n}}^*$ is not defined when $\boldsymbol{u} = 0$, i.e.

$$\boldsymbol{u} = 0 \Rightarrow k_P\boldsymbol{e}_{\boldsymbol{p}} - k_D\boldsymbol{e}_{\boldsymbol{v}} = -\ddot{\boldsymbol{p}}^* - g\hat{\boldsymbol{z}},$$

For the system in (2.14) we propose the following CLF, where again we stress the recursive form of the Lyapunov function that the backstepping provides

$$V_3(\boldsymbol{e}_{\boldsymbol{p}}, \boldsymbol{e}_{\boldsymbol{v}}, \hat{\boldsymbol{n}}) = V_2(\boldsymbol{e}_{\boldsymbol{p}}, \boldsymbol{e}_{\boldsymbol{v}}) + \gamma(1 - \hat{\boldsymbol{n}}^T\hat{\boldsymbol{n}}^*). \tag{2.18}$$

Differentiating (2.18) with respect to time, we have

$$\begin{aligned}\dot{V}_3(\boldsymbol{e_p}, \boldsymbol{e_v}, \hat{\boldsymbol{n}}) &= \dot{V}_2(\boldsymbol{e_p}, \boldsymbol{e_v}) - \gamma \frac{d(\hat{\boldsymbol{n}}^T \hat{\boldsymbol{n}}^*)}{dt} \\ &= -W_2(\boldsymbol{e_p}, \boldsymbol{e_v}) + \beta \boldsymbol{\varepsilon_1}^T \boldsymbol{\varepsilon_2} - \gamma \frac{d(\boldsymbol{u}^T \hat{\boldsymbol{n}})}{dt} \\ &= -W_2(\boldsymbol{e_p}, \boldsymbol{e_v}) - \beta \boldsymbol{\varepsilon_1}^T \boldsymbol{e_\vartheta} \|\boldsymbol{u}\| - \gamma \frac{d(\hat{\boldsymbol{n}}^T \hat{\boldsymbol{n}}^*)}{dt}.\end{aligned} \tag{2.19}$$

Computing the last term of (2.19) leads to

$$\begin{aligned}\frac{d(\hat{\boldsymbol{n}}^T \hat{\boldsymbol{n}}^*)}{dt} &= \dot{\hat{\boldsymbol{n}}}^T \hat{\boldsymbol{n}}^* + \hat{\boldsymbol{n}}^T \dot{\hat{\boldsymbol{n}}}^* \\ &= (S(\boldsymbol{\omega})\hat{\boldsymbol{n}})^T \hat{\boldsymbol{n}}^* + \hat{\boldsymbol{n}}^T S(\boldsymbol{\omega}^*)\hat{\boldsymbol{n}}^* \\ &= -\hat{\boldsymbol{n}}^T S(\hat{\boldsymbol{n}}^*)^T \boldsymbol{\omega} - \hat{\boldsymbol{n}}^T S(\hat{\boldsymbol{n}}^*)\boldsymbol{\omega}^* \\ &= \hat{\boldsymbol{n}}^T (S(\hat{\boldsymbol{n}}^*)\boldsymbol{\omega} - S(\hat{\boldsymbol{n}}^*)\boldsymbol{\omega}^*) \\ &= \hat{\boldsymbol{n}}^T S(\hat{\boldsymbol{n}}^*)(\boldsymbol{\omega} - \boldsymbol{\omega}^*),\end{aligned} \tag{2.20}$$

where we used the expression of $\dot{\hat{\boldsymbol{n}}}$ as in the third equation of (2.14) and the properties of the skew-symmetric matrices. Namely $S(\boldsymbol{x})\boldsymbol{y} = -S(\boldsymbol{y})\boldsymbol{x}$ and $S(\boldsymbol{x})^T = -S(\boldsymbol{x})$, $\forall x, y \in \mathbb{R}^3$. From (2.16), (2.17), using the skew-symmetric matrix definition $\boldsymbol{x} \times \boldsymbol{y} = S(\boldsymbol{x})\boldsymbol{y}$, we have

$$\begin{aligned}\boldsymbol{e_\vartheta} &= (\mathrm{I} - \hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T)\hat{\boldsymbol{n}}^* \\ &= \hat{\boldsymbol{n}}^*(\hat{\boldsymbol{n}}^T \hat{\boldsymbol{n}}) - \hat{\boldsymbol{n}}(\hat{\boldsymbol{n}}^T \hat{\boldsymbol{n}}^*) \\ &= \hat{\boldsymbol{n}} \times (\hat{\boldsymbol{n}}^* \times \hat{\boldsymbol{n}}) \\ &= S(\hat{\boldsymbol{n}})(\hat{\boldsymbol{n}}^* \times \hat{\boldsymbol{n}}) \\ &= S(\hat{\boldsymbol{n}})S(\hat{\boldsymbol{n}}^*)\hat{\boldsymbol{n}}.\end{aligned} \tag{2.21}$$

Substituting (2.20), (2.21) in (2.19) yields

$$\begin{aligned}\dot{V}_3(\boldsymbol{e_p}, \boldsymbol{e_v}, \hat{\boldsymbol{n}}) &= -W_2(\boldsymbol{e_p}, \boldsymbol{e_v}) - \beta(\boldsymbol{e_v} + k_1 \boldsymbol{e_p})^T S(\hat{\boldsymbol{n}})S(\hat{\boldsymbol{n}}^*)\hat{\boldsymbol{n}}\|\boldsymbol{u}\| - \gamma \hat{\boldsymbol{n}}^T S(\hat{\boldsymbol{n}}^*)(\boldsymbol{\omega} - \boldsymbol{\omega}^*) \\ &= -W_2(\boldsymbol{e_p}, \boldsymbol{e_v}) + \gamma \hat{\boldsymbol{n}}^T S(\hat{\boldsymbol{n}}^*)^T \left(-\frac{\beta}{\gamma} S(\hat{\boldsymbol{n}})^T (\boldsymbol{e_v} + k_1 \boldsymbol{e_p})^T \|\boldsymbol{u}\| + \boldsymbol{\omega} - \boldsymbol{\omega}^* \right).\end{aligned} \tag{2.22}$$

In this step we try to find an expression for $\boldsymbol{\omega}$ that makes the expression (2.22) negative definite. Thus, we propose

$$\boldsymbol{\omega} - \boldsymbol{\omega}^* - \frac{\beta}{\gamma} \|\boldsymbol{u}\| S(\hat{\boldsymbol{n}})^T (\boldsymbol{e_v} + k_1 \boldsymbol{e_p})^T = -k_\vartheta S(\hat{\boldsymbol{n}}^*)\hat{\boldsymbol{n}},$$

where $k_\vartheta > 0$ is an arbitrary positive scalar. This leads to

$$\boldsymbol{\omega} = \boldsymbol{\omega}^* + \frac{\beta}{\gamma} \|\boldsymbol{u}\| S(\hat{\boldsymbol{n}})(\boldsymbol{e_v} + k_1 \boldsymbol{e_p}) + k_\vartheta S(\hat{\boldsymbol{n}})\hat{\boldsymbol{n}}^*. \tag{2.23}$$

Substituting (2.23) into (2.22) yields

$$\dot{V}_3(\boldsymbol{e_p}, \boldsymbol{e_v}, \hat{\boldsymbol{n}}) = -W_2(\boldsymbol{e_p}, \boldsymbol{e_v}) - k_\vartheta \hat{\boldsymbol{n}}^T S(\hat{\boldsymbol{n}}^*)^T S(\hat{\boldsymbol{n}}^*)\hat{\boldsymbol{n}} \leq 0. \tag{2.24}$$

Summarizing the results, we have

$$\begin{aligned} T &= \boldsymbol{u}^T \hat{\boldsymbol{n}}, \\ \boldsymbol{u} &= \ddot{\boldsymbol{p}}^* + g\hat{\boldsymbol{z}} - k_P \boldsymbol{e_p} - k_D \boldsymbol{e_v}, \\ \boldsymbol{\omega} &= \boldsymbol{\omega}^* + \frac{1}{\gamma}\|\boldsymbol{u}\| S(\hat{\boldsymbol{n}})(\boldsymbol{e_v} + k_1 \boldsymbol{e_p}) + k_\vartheta S(\hat{\boldsymbol{n}})\hat{\boldsymbol{n}}^*. \end{aligned} \tag{2.25}$$

These results also confirm aforementioned properties of $\hat{\boldsymbol{n}}$ regarding the evolution of ψ. In fact the component $\omega_{\hat{\boldsymbol{n}}}$ of $\boldsymbol{\omega}$ along $\hat{\boldsymbol{n}}$, is crossed out from (2.23) because it is multiplied by $\hat{\boldsymbol{n}}^T S(\hat{\boldsymbol{n}}^*)^T$ in (2.22).

2.2.1 Yaw control

With the controller developed up to this point, we are able to control the position of the quadcopter. Now we want to track a reference trajectory for the yaw angle ψ defined by a given $\psi^*(t)$, first order differentiable ($\dot{\psi}^*$ exist). Hence, we can define a controller input u_ψ which will affect the dynamic of ψ through $\omega_{\hat{\boldsymbol{n}}}$ and writing

$$\dot{\psi} = u_\psi,$$

where $u_\psi = \boldsymbol{\omega}^T \hat{\boldsymbol{n}}$. We choose

$$u_\psi = \dot{\psi}^* - k_\psi \sin(\psi - \psi^*),$$

and therefore

$$\omega_{\hat{\boldsymbol{n}}} = \dot{\psi}^* - k_\psi \sin(\psi - \psi^*). \tag{2.26}$$

It is straightforward to notice that the only stable equilibrium of the closed loop system is $\psi = \psi^* + k\pi$ with $k \in \mathbb{Z}$.

2.3 Simulation and Experiments

2.3.1 Simulations in Matlab

In this section, we present some simulation made to test the model and the control algorithm. For this purpose, the proposed model and controller were implemented in the MATLAB/Simulink environment.

The first test was a regulation to a constant position $\boldsymbol{p}^*$ given by

$$\boldsymbol{p}^* = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}.$$

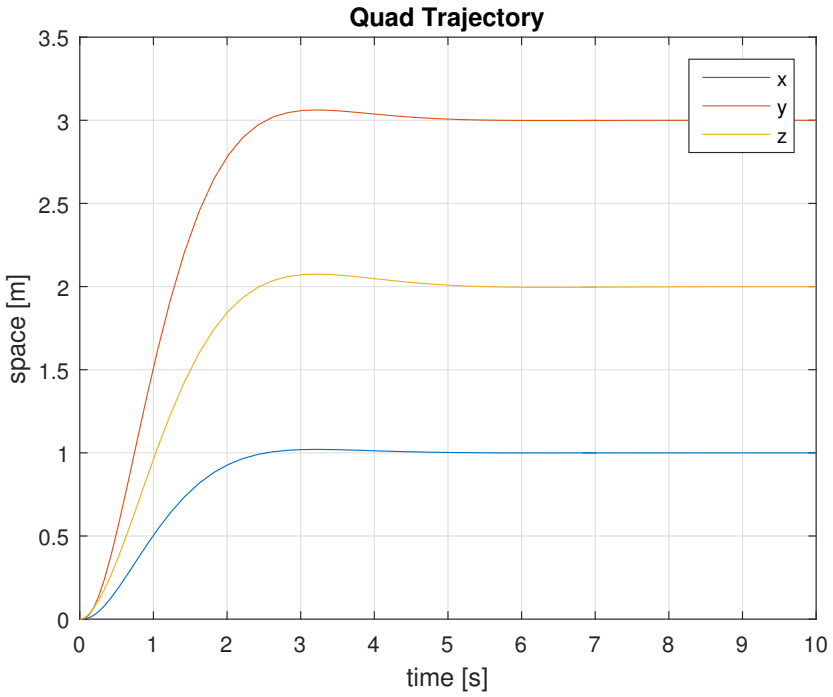


Figure 2.2: Constant reference regulation. The quadcopter is supposed to reach $\boldsymbol{p}^*$.

In Figures 2.2-2.4 it can be seen that the quad reaches the desired position with an asymptotically vanishing error. Figure 2.5 shows the evolution of Lyapunov function (2.24), which decreases in value as predicted. In the second test, we used a circular trajectory as reference defined as

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} \cos(\omega(t)) \\ -\sin(\omega(t)) \\ 0.5 \end{bmatrix},$$

where $\omega = 1$. Figures 2.6-2.9 show the result for this simulation.

2.3.2 Simulation in RotorS

In this section we present a simulation made using RotorSsimulator, in order to test the controller developed in Section 2.2 in an more realistic environment. In this simulation, two quadcopters are used, both controlled by T and $\boldsymbol{\omega}$ as in (2.25) and (2.26). The first quadcopter acts as the *leader* of the team, and it has to follow a pre-planned circular trajectory, while the second quadcopter, acting as the *follower*, takes as referenced position the position of the leader, and put to zero the further derivative of the desired trajectory. A video of the experiment can be found at [49]. Since real physics is implemented in RotorS, thus the quadcopters have defined dimensions, the follower

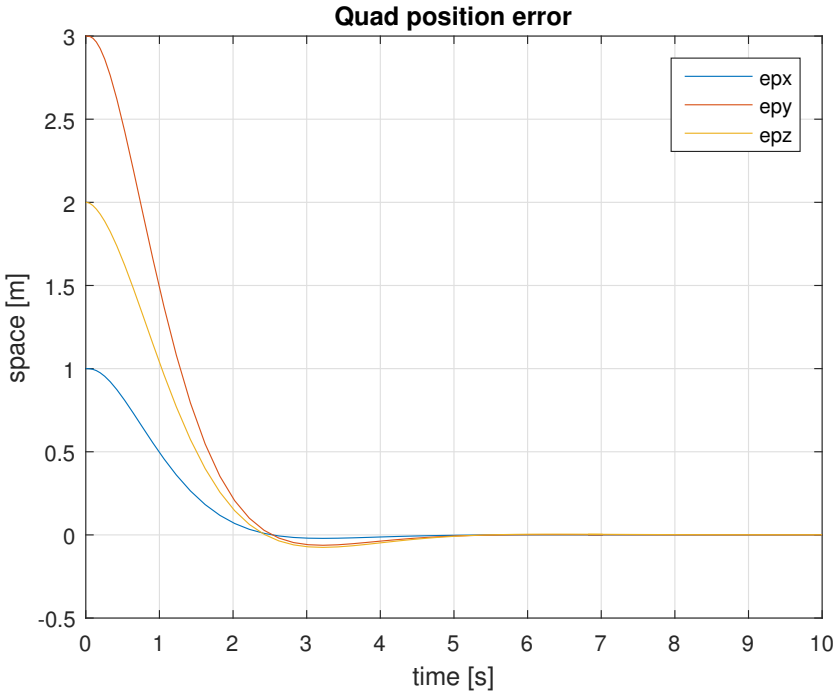


Figure 2.3: Position error with a constant reference.

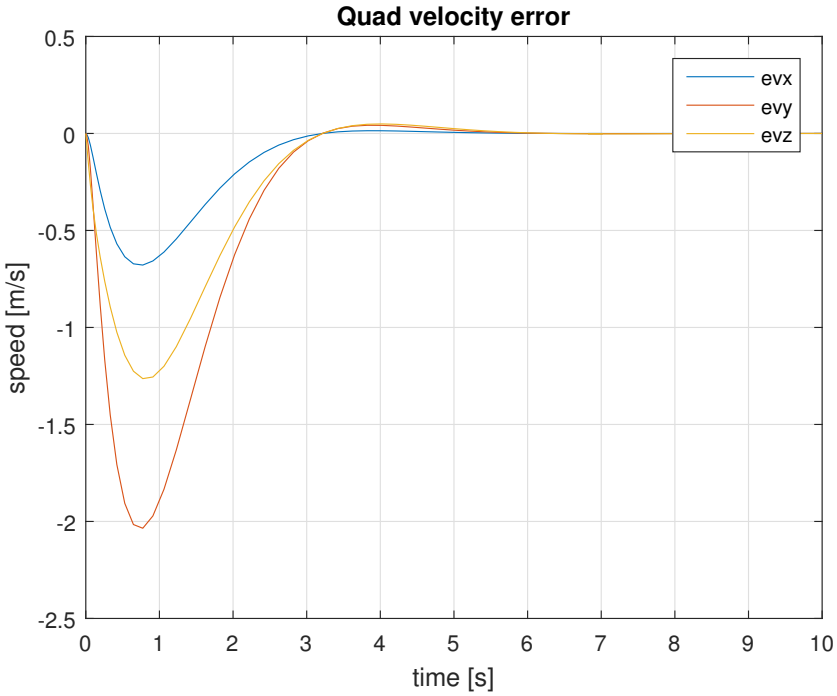


Figure 2.4: Velocity error with a constant reference.

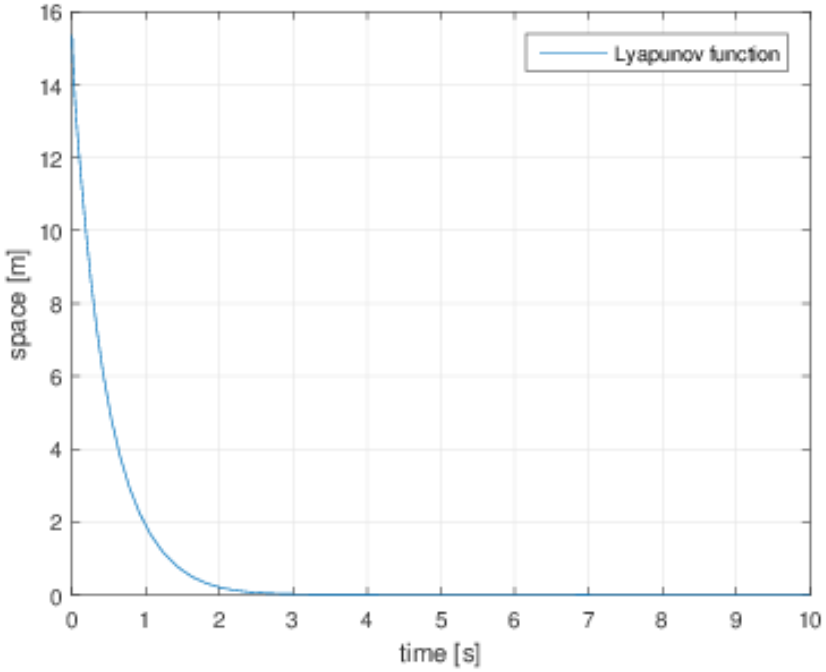


Figure 2.5: Lyapunov function for the constant reference.

quadcopter follows the trajectory of the leader with an offset given by the body of the quadcopter itself.

2.3.3 Experiments

In this Section, we show the implementation of the same task as in Section 2.3.2, using a marked stick as the leader of the team, and a real quadcopter as the follower. In this case, for safety constraints, the follower mimics the trajectory of the leader while maintaining a given offset from it. The further derivatives of the trajectory of the follower are set as zero as well. A video of the experiment can be found at [50].

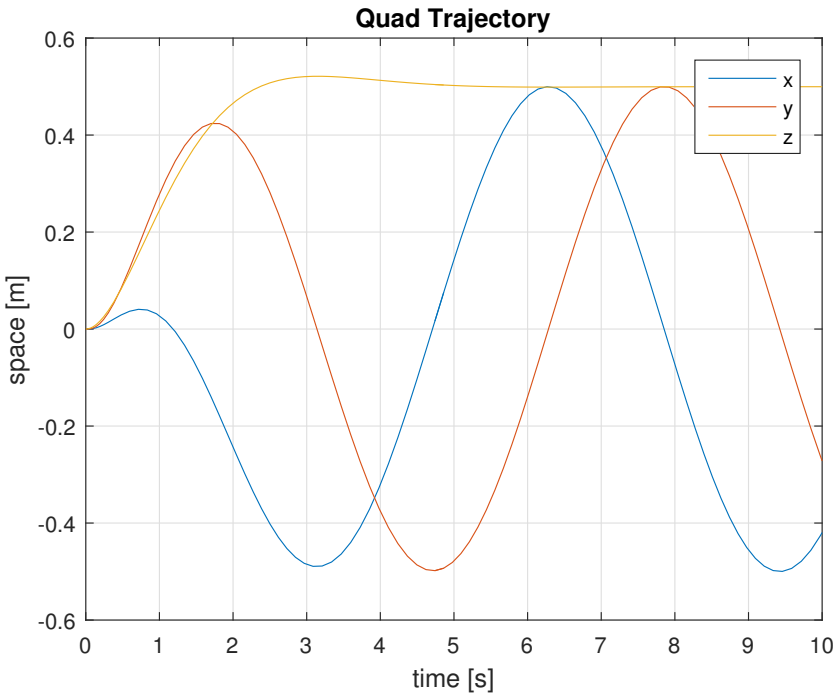


Figure 2.6: Quadcopter position for a trajectory tracking.

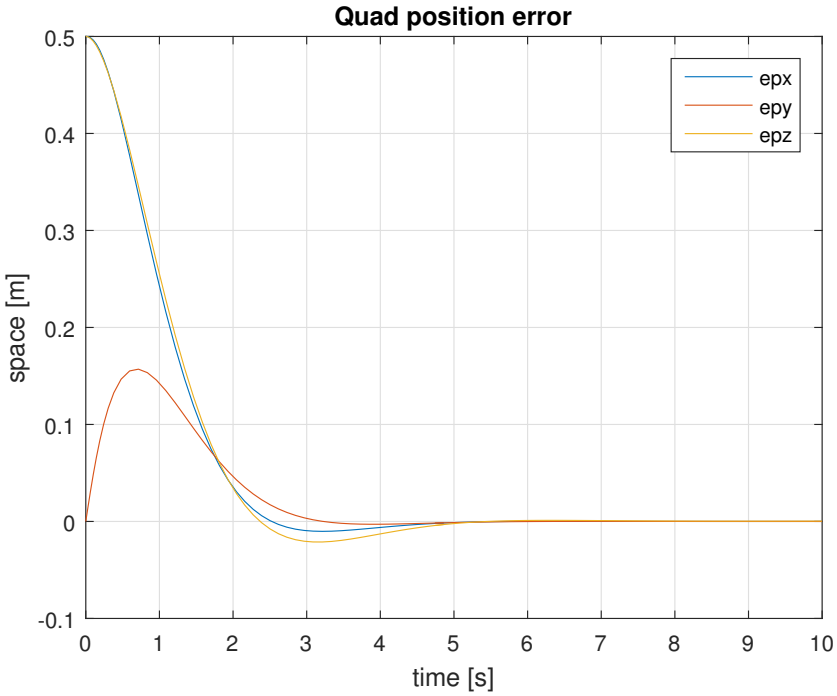


Figure 2.7: Position error for a circle trajectory tracking.

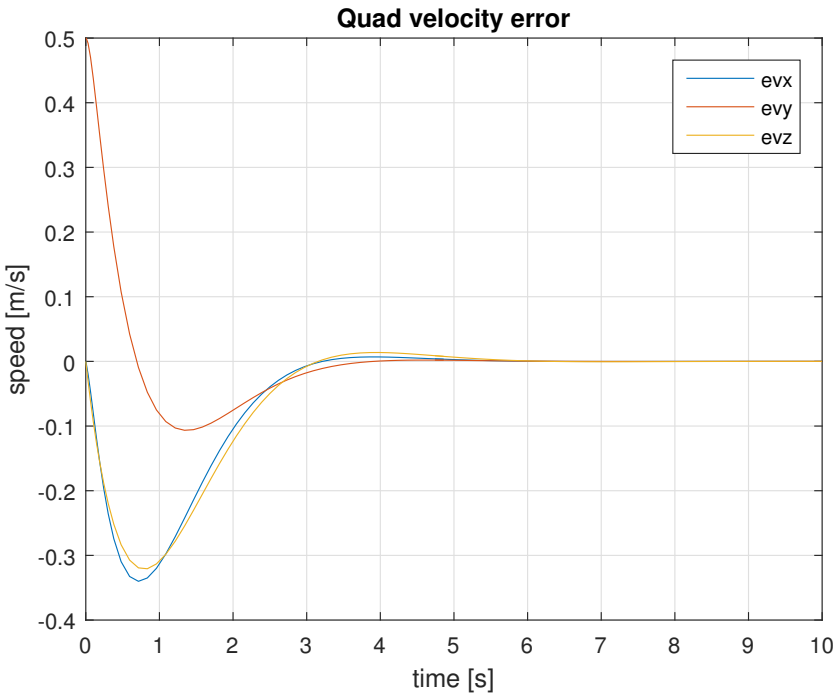


Figure 2.8: Velocity error for a circle trajectory tracking.

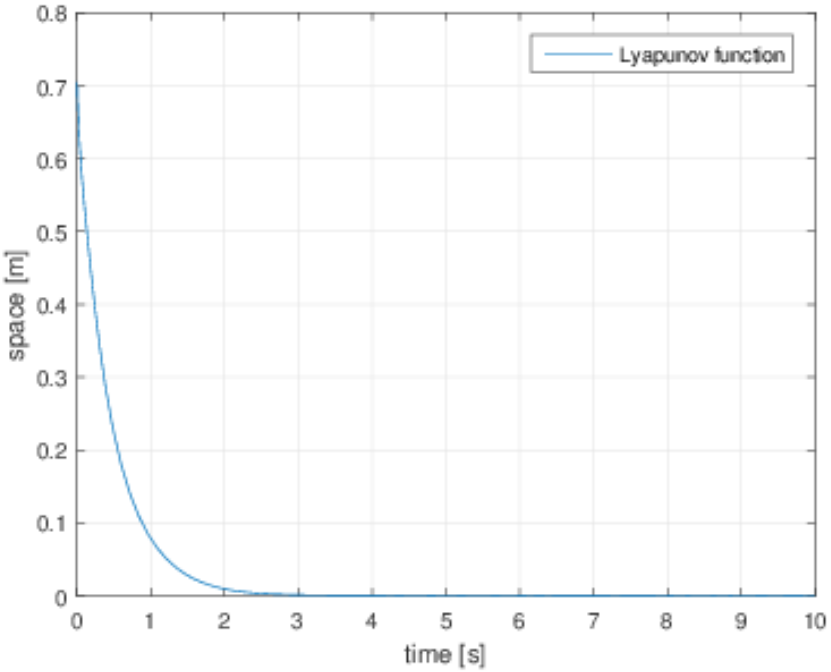


Figure 2.9: Lyapunov function for a circle trajectory tracking.

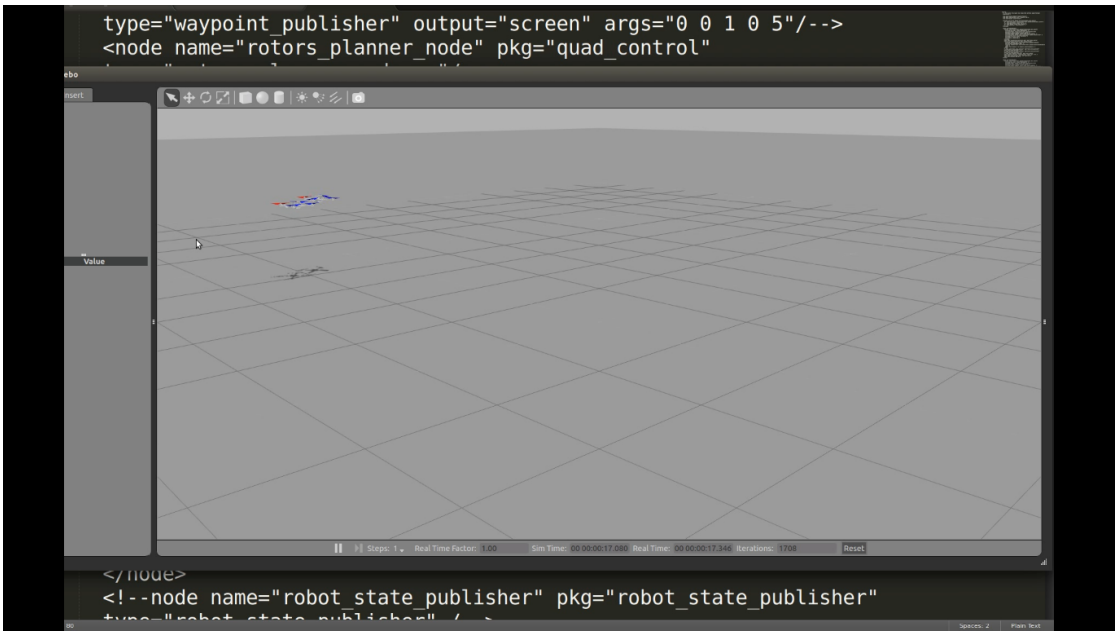


Figure 2.10: Screenshot of leader Following simulation in RotorS

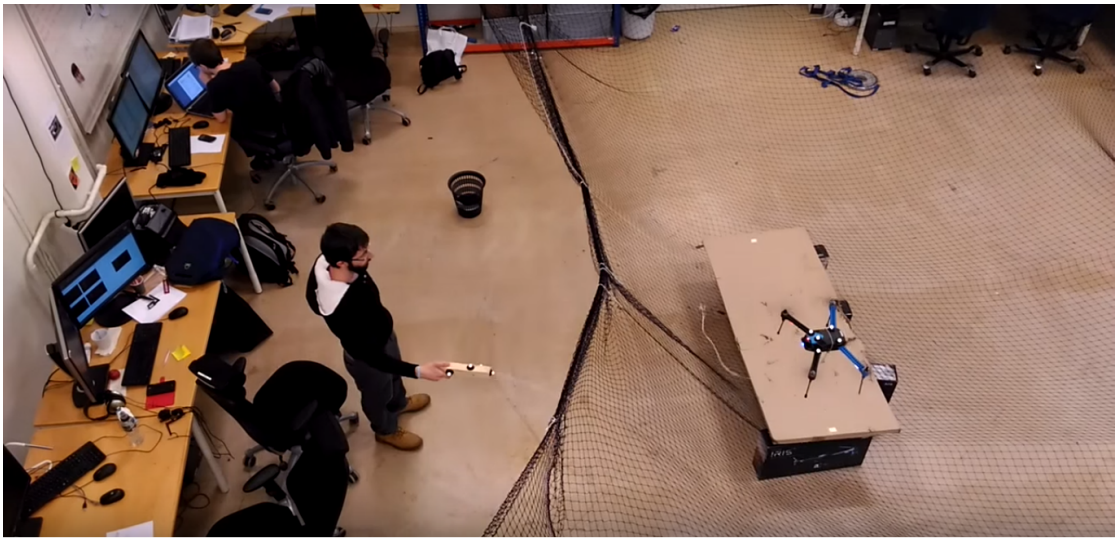


Figure 2.11: Screenshot of leader Following with Iris+ and a marked stick

3 Path Planning

In the previous chapter, we were able to define a dynamical model for describing the attitude of a rigid body (in our case, the quadcopter), and to develop a non linear controller with which we are able to make the quadcopter follow a desired, pre-planned trajectory. We recall that the main aim of this work is to implement of a coverage algorithm on a set of real quadcopters. Since the output of the algorithm (i.e. the set of new position assigned in each iteration to the quadcopters) cannot be foreseen, we need to implement some safety measure that let the quadcopters reach the given position while avoiding the others agents.

In this chapter, we address the problem of deriving a path planning procedure that takes into account all the available information about the single-agent goal position and the others agents current positions.

In what follows we refer to the goal position of a quad also with the word *waypoint*. That depends on the fact that, in a coverage mission, the goal position of each navigation segment is a waypoint generated by the proposed coverage algorithm.

3.1 Potential field navigation

Needless to say, the problem of deploying a set of robotic agent in physical environment implies the challenge of avoiding collisions among the agents as well as with other obstacles. Many approaches to this problem can be found in literature. For example, collision avoidance flight can be treated as a constraint in a on-line optimization process [18,19,32,51]. However, this approach is computational demanding, and not suitable for on-board implementation on the real robots. Our approach instead is to consider each agent moving in a dynamic potential field of forces, in which the goal position is treated as an attractive pole and the current positions of others agents (and any other object to avoid) as repulsive surfaces [17, 52].

To obtain a collision-free navigation toward the waypoint, the controller in Section 2.2 takes as input an acceleration defined as

$$\boldsymbol{a}^* = \mathbf{f}(\boldsymbol{w}_\mathrm{p} - \boldsymbol{p}_\mathrm{self}, \boldsymbol{v}_\mathrm{self}) + \mathbf{g}(\boldsymbol{p}_\mathrm{obs1} - \boldsymbol{p}_\mathrm{self}, \boldsymbol{p}_\mathrm{obs2} - \boldsymbol{p}_\mathrm{self}, \dots, \boldsymbol{p}_\mathrm{obsN} - \boldsymbol{p}_\mathrm{self}), \tag{3.1}$$

where $\mathbf{f} : \mathbb{R}^6 \to \mathbb{R}^3$, $\mathbf{g} : \mathbb{R}^{(3\times N-1)} \to \mathbb{R}^3$, $\boldsymbol{w}_\mathrm{p} \in \mathbb{R}^3$ is the assigned waypoint, $\boldsymbol{p}_\mathrm{self}$ is the current position of the agent, $\boldsymbol{v}_\mathrm{self} \in \mathbb{R}^3$ is its velocity and $\boldsymbol{p}_\mathrm{obs1} \dots \boldsymbol{p}_\mathrm{obsN} \in \mathbb{R}^3$ are the current positions of the obstacles, either others agents or fixed objects.

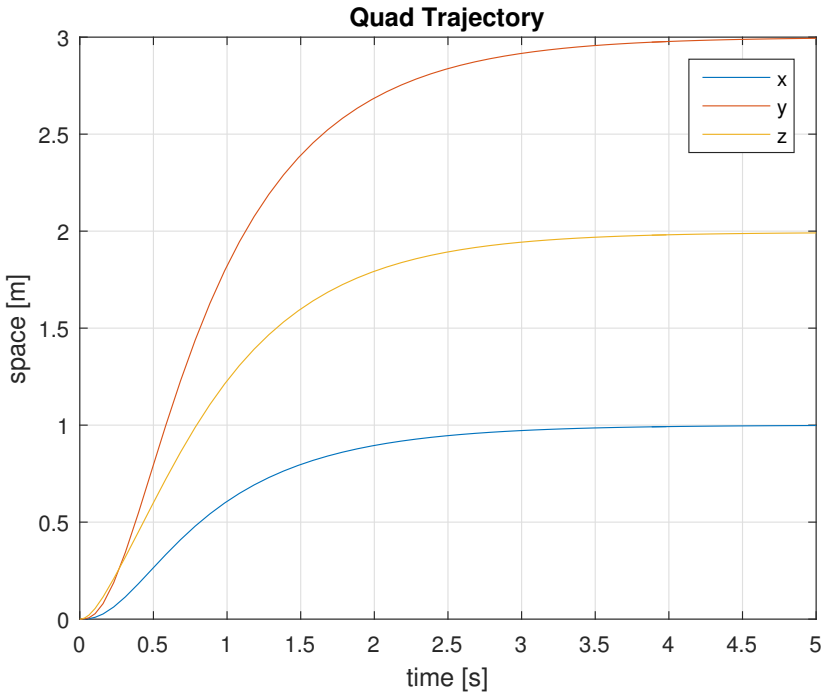


Figure 3.1: Position dynamics under attractive field.

3.2 Attractive term

The first term in the right-hand side of $\boldsymbol{a}^*$ in (3.1) is the *attractive* contribution toward the waypoint, as briefly stated in Section 3.1. This term works like a spring, that pulls the agent to the goal position with an intensity proportional to the current distance from it, as

$$\mathbf{f}^* = k_{\boldsymbol{w}_\mathrm{p}}(\boldsymbol{w}_\mathrm{p} - \boldsymbol{p}_\mathrm{self}) - k_{\boldsymbol{v}}\boldsymbol{v}_\mathrm{self}.$$

Figure 3.1 clarifies the evolution of the position of the quadcopter to the waypoint. In this simulation, the controller defined in Chapter 2 takes as input reference the trajectory created with the attractive term, plus an *attractive-like* contribution that tends to bring the velocity to zero. In this way, we can also tune the dynamic of the velocity through $k_{\boldsymbol{v}}$. In the simulations, we chose $k_{\boldsymbol{w}_p}, k_{\boldsymbol{v}} = 5$. Figure 3.2 show speed dynamic.

3.3 Collision Avoidance

Although it is possible to plan off-line safe trajectories for a set of robots by adding constraints on the path of each single agent [3, 22], it is compulsory to deal with this problem every time that the set of new waypoints comes from an unpredictable, high level task implementation. As stated in Section 3.1, with the potential field approach we treat any obstacle to avoid as a repulsive surface. The strength of the repulsive action depends

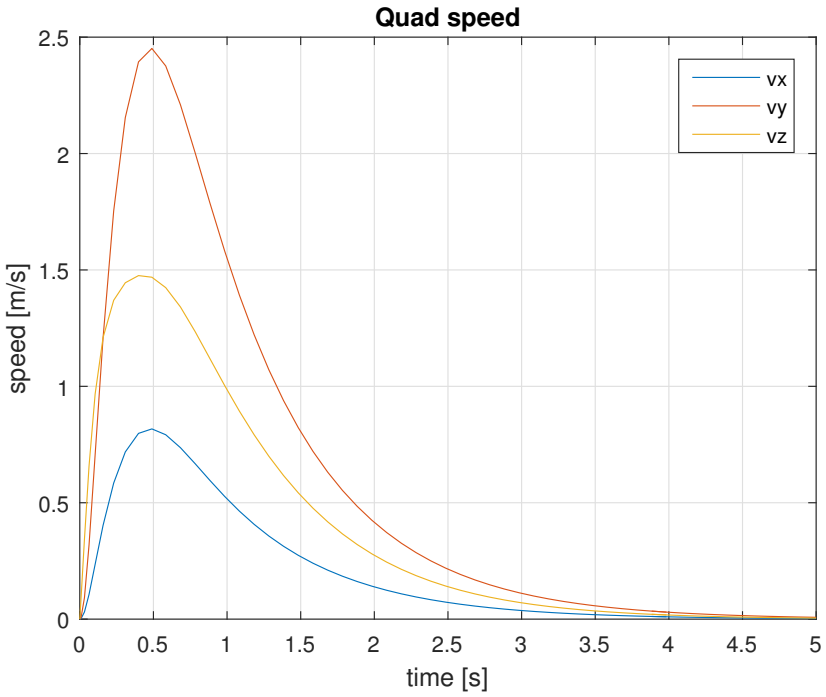


Figure 3.2: Velocity dynamic under attractive field

on the distance, namely, the stronger the trajectories of the quadcopters will be affected by it. Since the quadcopters can approach each other from any possible direction, it seems reasonable to virtually surround them with a spherical surface centered in the quadcopter's center of mass. The radius of the sphere is the same for every quadcopter, and is denoted as ς and referred to as *threshold* of the collision avoidance. The condition $\|\boldsymbol{p}_{\text{obs}} - \boldsymbol{p}_{\text{self}}\| < \varsigma$ discriminates the presence of the second term in the right-hand side of (3.1). The component $\mathbf{g}$ in (3.1) is computed as follows:

$$\mathbf{g}^* = \mathbf{g}^{*\parallel} + \mathbf{g}^{*\perp}, \tag{3.2}$$

where

$$\mathbf{g}^{*\parallel} = \begin{bmatrix} \mathrm{g}_x^{*\parallel} \\ \mathrm{g}_y^{*\parallel} \\ 0 \end{bmatrix} = \sum_{\boldsymbol{p}_{\text{obs}} \in \boldsymbol{P}_{obs}} \begin{cases} \frac{1}{2} k_{rep} \frac{(\boldsymbol{p}_{\text{obs}} - \boldsymbol{p}_{\text{self}})}{\|\boldsymbol{p}_{\text{obs}} - \boldsymbol{p}_{\text{self}}\|} \left(\frac{1}{\|\boldsymbol{p}_{\text{obs}} - \boldsymbol{p}_{\text{self}}\|} - \frac{1}{\varsigma} \right), & \text{if } \|\boldsymbol{p}_{\text{obs}} - \boldsymbol{p}_{\text{self}}\| < \varsigma, \\ 0 \text{ otherwise}, \end{cases}$$

and

$$\boldsymbol{g}^{*\perp} = \begin{bmatrix} -\mathrm{g}_y^{*\parallel} \\ \mathrm{g}_x^{*\parallel} \\ 0 \end{bmatrix},$$

where we have underlined that the acceleration reference has two component : $\boldsymbol{a}^{*\parallel}$ which define an acceleration vector that pushes the quad away from the obstacle (or two quad

away from each other) in the direction parallel to the previous motion direction, and $\boldsymbol{a}^{*\perp}$ which pushes the quad in the orthogonal direction of the previous motion. This design allows the quadcopter to handle some critical situations, of which we give an example in the next section.

Algorithm 1 Collision Avoidance implementation.

procedure COLLISION AVOIDANCE
  $k_{rep} \leftarrow$ gain for the repulsive behavior
  $s \leftarrow$ threshold
  $quad\_pos \leftarrow$ current position of the quad
  $obs\_pos \leftarrow$ current position of the obstacle
  $quad\_vel \leftarrow$ current velocity of the quad
  $d = \|obs\_pos - quad\_pos\| \leftarrow$ Eucledian distance between obstacle and quad
  if $d < s$ **and** $quad\_vel * (obs\_pos - quad\_pos) < 0$ **then**
    $acceleration$eq. (3.2).
  else
    $acceleration \leftarrow 0$

3.4 Simulation and Experiments

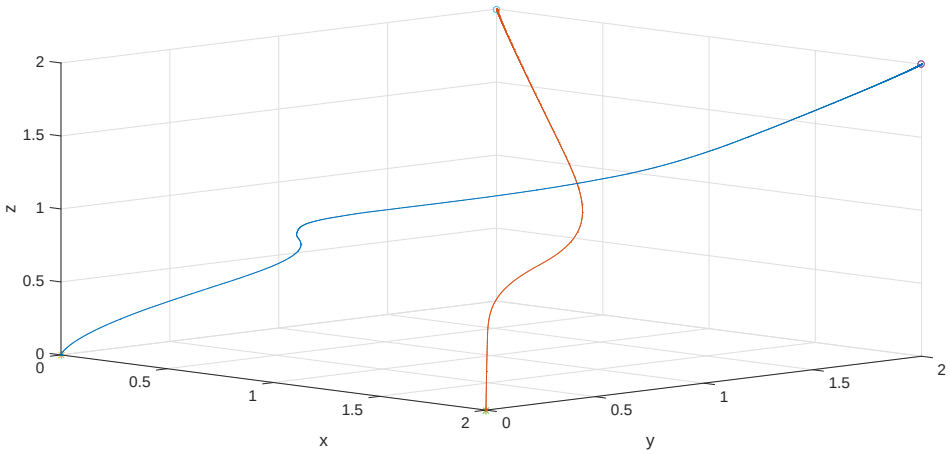
3.4.1 Simulations in Matlab

In this section, we give the results of some simulations done for testing the collision avoidance and the path planning with potential fields. Figures 3.3 show the navigation of two quadcopters when the start position and the goal position are

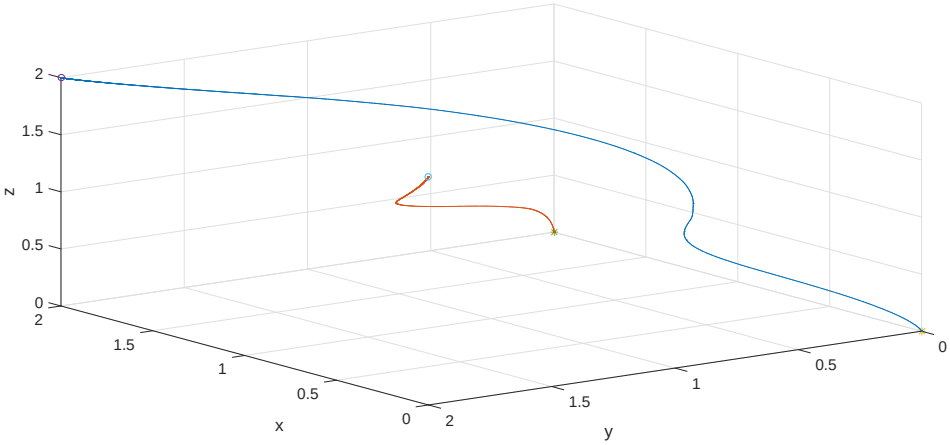
- $\boldsymbol{p}^s_{\text{quad1}} = \begin{bmatrix}0\\0\\0\end{bmatrix}, \boldsymbol{p}^s_{\text{quad2}} = \begin{bmatrix}2\\0\\0\end{bmatrix},$
- $\boldsymbol{p}^g_{\text{quad1}} = \begin{bmatrix}2\\2\\1\end{bmatrix}, \boldsymbol{p}^g_{\text{quad2}} = \begin{bmatrix}0\\2\\1\end{bmatrix},$

where $\boldsymbol{p}^s_{\text{quadi}}, \boldsymbol{p}^g_{\text{quadi}}$ for $i = 1, 2$ are the start and the goal position of the trajectory, respectively. where we have used the following parameters: $k_{\boldsymbol{w}_\text{p}} = 1$, $k_{\boldsymbol{v}} = 5$, $k_{\text{rep}} = 2$, $\varsigma = 1.5$. In Figure 3.4 we tested the algorithm with the three simulated quad. In this case we have:

- $\boldsymbol{p}^s_{\text{quad1}} = \begin{bmatrix}0\\0\\0\end{bmatrix}, \quad \boldsymbol{p}^s_{\text{quads}} = \begin{bmatrix}2\\0\\0\end{bmatrix}, \quad \boldsymbol{p}^s_{\text{quad3}} = \begin{bmatrix}-2\\0\\0\end{bmatrix},$
- $\boldsymbol{p}^g_{\text{quad1}} = \begin{bmatrix}0\\3\\2\end{bmatrix}, \quad \boldsymbol{p}^g_{\text{quad2}} = \begin{bmatrix}-2\\3\\2\end{bmatrix}, \quad \boldsymbol{p}^g_{\text{quad3}} = \begin{bmatrix}2\\3\\2\end{bmatrix},$

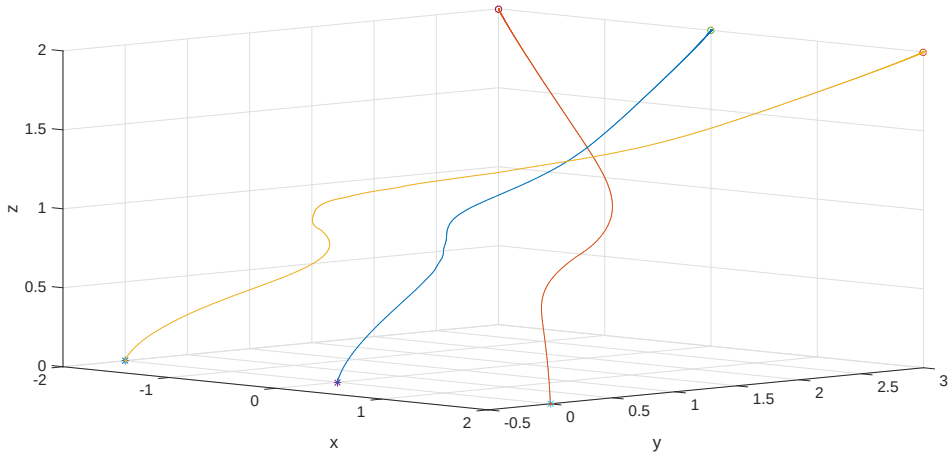


(a)

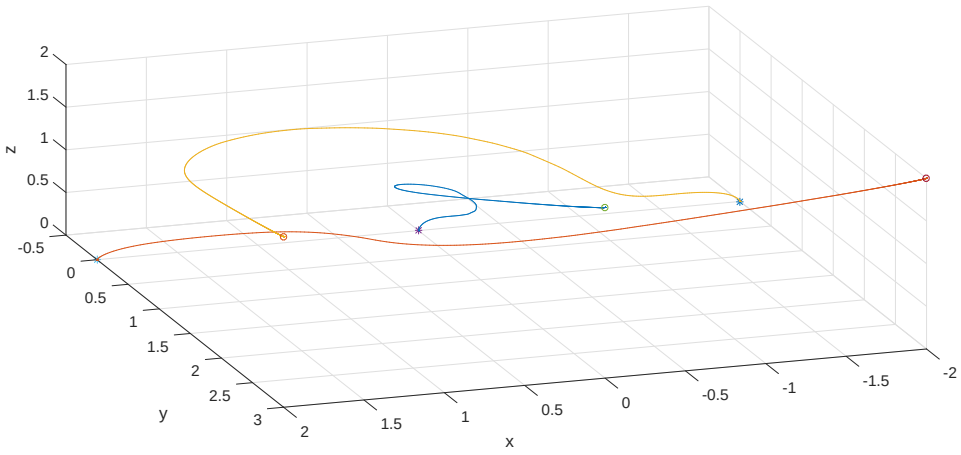


(b)

Figure 3.3: Potential field navigation with two quad



(a)



(b)

Figure 3.4: Potential field navigation with three quad

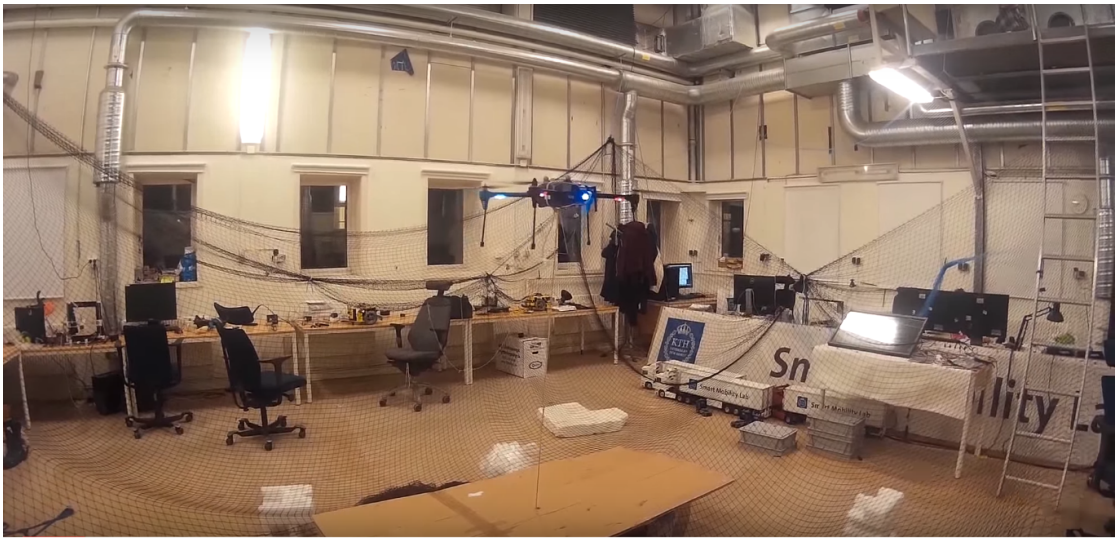


Figure 3.5: Screenshot of Collision Avoidance with Iris+

while we set $k_{\text{rep}} = 3$ and we keep the others gains unchanged.

3.4.2 Simulations in RotorS

As the last test for the potential field navigation, we simulated a *swapping* task. In other words, we defined as goal position for one quad the initial position of the other and vice versa. The attractive term in (3.1) pulls each quad toward the other on the same direction, and thanks to the terms $\boldsymbol{g}^{*\parallel}$ and $\boldsymbol{g}^{*\perp}$ in (3.2), the quadcopters will avoid collision by flying in a seemingly circular path. A video of the simulation made with RotorS can be seen at [53].

3.4.3 Experiments

In this Section, we test the Collision Avoidance by combining the controller implemented in Chapter 2 with the path planner in (3.1) on the real quadcopter. A video of the experiment can be seen at [54]. In the test, the quadcopter has to stay still at $\boldsymbol{p} = [0, 0, 1.2]^T$, while avoiding collision with a moving object, for which we use again the marked stick of Section 2.3.3. The gains of the experiment are set as follows: $k_{\boldsymbol{w}_{\text{p}}} = 1$, $k_{\boldsymbol{v}} = 5$, $k_{\text{rep}} = 1$, $\varsigma = 1$.

4 Coverage Algorithm

Throughout the previous chapters, we faced the steps required to implement a high-level task on real robots. In this chapter, we finally tie all the results accomplished so far by presenting in detail the proposed coverage algorithm.

4.1 Visibility Function

As previously mentioned, the purpose of a coverage algorithm is to deploy a set of mobile sensing agents, in order to enhance as much as possible the perception of a given environment. To this aim, a model of the sensing device is needed, since every sensor has is own characteristics, and these affect the perception of the environment.

Since we are interested in surveiling the given area with vision-based sensors (such as cameras), we shall define the *field of view* of the optical sensor, as the extent of observable world that it can collect at any given time [55]. This relies on many factor (which we won't treat here for the sake of brevity), but the main feature that the model of a sensor has to provide is the ability to estimate the **visibility** of a object at a given point in the field of view, according to a certain measurement index related to the sensor itself. In this work we develop the coverage algorithm as an application for bi-dimensional surveillance, hence we define the *visibility* of a point $\boldsymbol{s}$ as a function $\text{vis} : \mathbb{R}^2 \to \mathbb{R}_+$. First we partition $\mathbb{R}^2$ as follows:

$$\begin{aligned} \mathrm{R}_1 &= \{\boldsymbol{s} \in \mathbb{R}^2 : s_x \leq 0\}, \\ \mathrm{R}_2 &= \{\boldsymbol{s} \in \mathbb{R}^2 : s_x > 0 \quad \text{and} \quad \|\boldsymbol{s}\| \leq 1\}, \\ \mathrm{R}_3 &= \{\boldsymbol{s} \in \mathbb{R}^2 : s_x > 0 \quad \text{and} \quad \|\boldsymbol{s}\| > 1\}. \end{aligned} \tag{4.1}$$

Then we introduce the following definitions.

Definition 1 (Agent)**.** In this chapter, an agent a is fully characterized by a pose $A \in SE(2)$. This means that a is fully defined by a tuple $(\boldsymbol{p}, \psi)$, where $\boldsymbol{p} \in \mathbb{R}^2$ denotes the position of the agent, and $\psi \in [-\pi, \pi)$ denotes the orientation of a sensing device. Therefore, we will refer to the orientation of an agent also as the *yaw* of the agent.

Definition 2 (Visibility)**.** Let us consider an agent a, that has a pose $A = I_3$ (with I_3 three-dimensional identity matrix), i.e., lays in the origin with yaw angle $\psi = 0$. Then, for any given point $\boldsymbol{s} \in \mathbb{R}^2$ and for the agent a, we define the **visibility** vis_{I_3} of the point

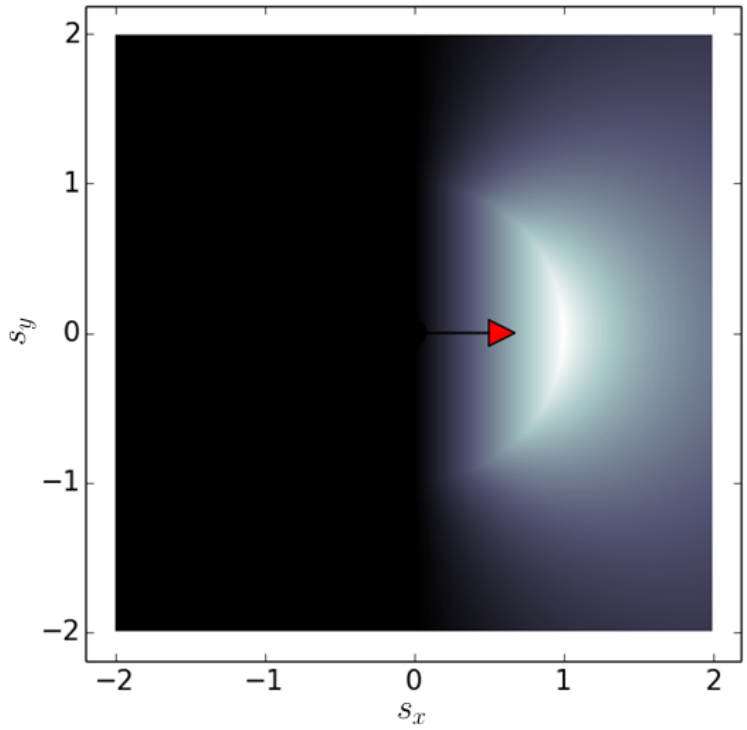


Figure 4.1: Heat map of the visibility function. Lighter points are seen better by the agent.

$\boldsymbol{s}$, as [56]:

$$\text{vis}_{I_3}(\boldsymbol{s}) : \begin{cases} 0, & \boldsymbol{s} \in \text{R}_1, \\ s_x, & \boldsymbol{s} \in \text{R}_2, \\ s_x/\|\boldsymbol{s}\|^2, & \boldsymbol{s} \in \text{R}_3. \end{cases} \tag{4.2}$$

Equation (4.2) captures a reasonable behavior for a sensing device (and thus, in this context, for the agent). The visibility of all the points in R_1 is 0, because the region R_1 lays on the back of the agent a. The points in R_2 are on front of the agent and close to it, so the visibility increases with the distance from the agent ($\|\boldsymbol{s}\|$) and with the centrality in the agent's field of view ($s_x/\|\boldsymbol{s}\|$); thus the visibility can be written as $\|\boldsymbol{s}\|s_x/\|\boldsymbol{s}\| = s_x$. The points in R_3 are in front of the agent and far from it, so the visibility is inversely proportional to the distance squared ($1/\|\boldsymbol{s}\|^2$), and directly proportional to the centrality in the field of view ($s_x/\|\boldsymbol{s}\|$), so, it can be written as $(1/\|\boldsymbol{s}\|)(s_x/\|\boldsymbol{s}\|)$. Figure 4.1 shows a heat map of the *visibility function*, vis_{I_3} for an agent with pose $A = I_3 \in SE(2)$. Lighter points are seen better by the agent. The visibility of a point $\boldsymbol{s} \in \mathbb{R}^2$ with respect to an agent a with generic pose A can be derived by applying (4.1) and (4.2) to the coordinates of the point $\boldsymbol{s}$ in a reference frame attached to the agent a, as

$$\text{vis}(A, \boldsymbol{s}) = \text{vis}_{I_3}((A^{-1}\tilde{\boldsymbol{s}})_{xy}) \tag{4.3}$$

where $\tilde{\boldsymbol{s}}$ is the *homogeneous coordinates* of $\boldsymbol{s}$, as $\tilde{\boldsymbol{s}} = [s_x, s_y, 1]^T$. It can be shown that such function is Lipschitz in $\Omega \times [-\pi, \pi)$ for any compact set $\Omega \subset \mathbb{R}^2$. However, here we leave out a formal proof of the above statement. For the purpose of the mission, we abstract the environment in a finite set of $n \in \mathbb{N}$ points of interest $\boldsymbol{s}_k \in \mathbb{R}^2$, with $k = 1, \ldots, n$, and we call each of this $\boldsymbol{s}_k$ *landmark*. Let $\mathbf{S} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n]$ be the set of landmarks. First, let us consider a coverage mission that involves a single agent a, with a variable pose $A \in SE(2)$, and the set $\mathbf{S}$ of landmarks. Follows that each landmark $\boldsymbol{s}_k$ has a certain value of the visibility, with respect to the current pose A of the agent a, as in (4.2). Thus, we define a *coverage score* of the agent as

$$C(A, \mathbf{S}) = \sum_{\boldsymbol{s}_k \in \mathbf{S}} \text{vis}(A, \boldsymbol{s}_k), \tag{4.4}$$

where $\text{vis}(A, \boldsymbol{s}_k)$ is defined as (4.2). The visibility $\text{vis}(A, \boldsymbol{s}_k)$ of a landmark, as well as the pose A of a are time-dependent, but we omit this in the notation for a better readability. Hence, the aim of the mission is to find the pose A^* of the agent that maximizes the measure of the coverage score defined in (4.4). This can be mathematically expressed as

$$A^*(\mathbf{S}) = \underset{A \in SE(2)}{\text{argmax}}\, C(A, \mathbf{S}),$$

where A^* is the maximum value of the coverage score obtained with the optimization. Solution to such problem are well treated in [57].

4.2 Deployment

Now we extend the problem to consider a team of agents $a_i \in \mathcal{A}$, where $i = 1 \ldots m$ and $m \in \mathbb{N}$, with a set of poses $\mathbf{A} = (A_1, A_2, \ldots A_m)$. Next, we define a set of m *disjointed* partitions of $\mathbf{S}$, as $\mathbf{P} = [\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_m]$. In this scenario, each agent a_i is *responsible* of a subset $\mathbf{P}_j$ for $j = 1 \ldots m$ of landmarks, with $\mathbf{P}_1 \bigcup \cdots \bigcup \mathbf{P}_m = \mathbf{S}$ and $\mathbf{P}_i \cup \mathbf{P}_j = \emptyset$, meaning that each agent will *watch over* its own set of landmarks. Moreover, the visibility of a landmark is calculated with respect to the pose A_i of the agent a_i that *owns* the landmark. The coverage score of a_i is computed as in (4.4), with the difference that it considers only the landmarks $\mathbf{P}_i$, i.e. only the ones in its own partition of landmarks, as

$$C(A_i, \mathbf{P}_i) = \sum_{\boldsymbol{s}_k \in \mathbf{P}_i} \text{vis}(A_i, \boldsymbol{s}_k),$$

while the coverage score of the whole team of agents is calculated as the sum of the coverage score of each agent, as

$$C(\mathbf{A}, \mathbf{P}) = \sum_{i=1}^{m} C(A_i, \mathbf{P}_i). \tag{4.5}$$

For implementation purposes, we assume that the agents' positions are restricted to a subset $\Omega \subset \mathbb{R}^2$, and so $A_i \in SE(2)_\Omega$ for $i = 1 \ldots m$, where $SE(2)_\Omega$ is the restriction of

$SE(2)$ to those poses whose position is in Ω, as

$$SE(2)_\Omega = \{A \in SE(2) \; s.t. \; \boldsymbol{p}(A) \in \Omega\}.$$

This means that the optimization problem is limited to a subset on $SE(2)$. Such limitation does not apply to the position of landmarks, so, $\boldsymbol{s}_k \in \mathbb{R}^2$. Moreover, for the coverage mission we assume that Ω and $\mathbf{S}$ are assigned and constant, while the pose $\mathbf{A}$ of the agents and the *ownership* of the landmarks are variable. If we define the *state* of the system as the tuple $(\mathbf{A}, \mathbf{P})$, then in this configuration, the aim of the mission is to find the optimal tuple $(\mathbf{A}^*, \mathbf{P}^*)$, which means the optimal set of pose $A_i, \dots A_m$ and the optimal repartition of the landmarks sets $\mathbf{P}_i \dots \mathbf{P}_m$, associated to these poses, such that

$$C^*(\mathbf{A}^*, \mathbf{P}^*) = \max_{\substack{A \in SE(2)_\Omega \\ \mathbf{P} \in \mathcal{P}_n(\mathbf{S})}} C(\mathbf{A}, \mathbf{P}),$$

where $\mathcal{P}_n(\mathbf{S})$ is the set of all the possible partitions of $\mathbf{S}$ into n ordered subsets. It is well known in literature that, for coverage problems, the solution comes in the form of a *Voronoi configuration* $\mathcal{V}(\mathbf{S})$ [58, 59]. Is this work, we tailor the definition of a Voronoi configuration to our setup. Namely, a Voronoi configuration is defined as follows.

Definition 3 (Voronoi configuration)**.** Given a set of agents $\mathcal{A} = (a_i, \dots, a_m)$, a set of landmarks $\mathbf{S} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \dots, \boldsymbol{s}_n]$ and the region $\Omega \subset \mathbb{R}^2$. Let us call $A_i \in SE(2)_\Omega$ the pose of the agent a_i. Thus, $\mathbf{A} = (A_i, A_2, \dots, A_m)$ and we recall that $\mathbf{P} = [\mathbf{P}_1, \mathbf{P}_2, \dots, \mathbf{P}_m]$ is the set of partition of landmarks among the agents. Finally, let $\sigma > 0$.

.In this work we attest that the coverage algorithm explained further on, leads the system in a configuration $(\mathbf{A}^*, \mathbf{P}^*)$, which we define as a Voronoi configuration in Ω if the following two condition are true:

$$C(A_i^*, \mathbf{P}_i^*) \geq \max_{A_i \in SE(2)_\Omega} C(A_i, \mathbf{P}_i) - \sigma \; \forall a_i \in \mathcal{A},$$

and

$$\text{vis}(A_i^*, \boldsymbol{s}_k) \geq \text{vis}(A_j^*, \boldsymbol{s}_k) - \sigma \; \forall i, j \in \mathcal{A},$$

where $\sigma > 0$ is a numerical tolerance defined for the purpose of the optimization process, while the coverage score through each iteration is always non-decreasing. In order for the algorithm to take the landmarks partitions to a Voronoi configuration, while improving the visibility of each landmark and thus the overall coverage score, it has to be designed for handling non-trivial interaction between the agents, as well as simultaneous communications between different pairs of agents. The algorithm is shown to complete the coverage mission after a finite number of successful interaction between the agents, namely In the next sections, we will present all the different part of the algorithm implementation, clarifying each single procedure with the help of pseudo-code.

4.3 Initialization

The first step of the coverage mission is the assignment of the initial pose and the initial set of landmarks to the whole team of agents. This procedure can be done either by a human operator, or automatically by assigning each landmark in the area to cover to a randomly chosen agent. Note that the random assignment can lead to having an agent responsible for an empty set of landmarks, but even in this unlikely case, the agent that does not own landmarks after the initialization process, is able to acquire landmarks by means of the algorithm. Furthermore, even if the initial pose of the aforementioned agent is such that it is not able to see any landmark (this happens, for example, when the agent lays at the boundary of Ω and points outside of it), the algorithm will end in a consistent, suboptimal configuration. After the initial assignment, each agent a_i creates a set $\mathbf{Q}_i$ of available agents a, in order to keep track of which of them it can interact with during the trading procedure. $\mathbf{Q}_i$ is initialized as $\mathbf{Q}_i = \mathcal{A} \setminus a_i$. Finally, each agent calls a *pose optimization* procedure. The code listed in Algorithm 2 gives an insight of the initialization procedure.

Algorithm 2 Initialization procedure for the coverage control algorithm.

```
1: procedure INITIALIZATION
2:     for s_k ∈ S do
3:         choose a_i ∈ A
4:         P_i ← P_i ⋃ s_k
5:     for a_i ∈ A do
6:         Q_i ← A \ a_i
7:         call Algorithm 3
```

4.4 Pose optimization

After the initialization procedure has finished, all the agents first calculate the visibility of each landmark in their own set $\mathbf{P}_i$ from their current positions, then optimize the pose in order to maximize the coverage score on that set. For the optimization procedure we chose to use an optimization method called *differential evolution* [60], limiting the area in which we seek the solution to Ω and with a tolerance between two solutions σ. The optimization procedure gives as output a new pose $(\boldsymbol{p}, \psi)$ that the agent has to reach for the algorithm to continue. In other words, in our implementation this procedure generate a new waypoint and gives it to the planner in (3.1) of the quadcopter.

The optimization process takes place every time the set of landmarks $\mathbf{P}$ of an agent changes, i.e. every time two agents *trade* some landmark succesfully. Algorithm 3 articulates what explained up to this point.

Algorithm 3 Pose Optimization procedure for each agent $a_i \in \mathcal{A}$

procedure POSE OPTIMIZATION
    $old\_score \leftarrow$ Old coverage score
    $new\_score \leftarrow$ New coverage score
    $\boldsymbol{p} \leftarrow$ current position of the agent
    $\psi \leftarrow$ current orientation of the agent
    $\boldsymbol{p}_n \leftarrow$ optimized position of the agent
    $\psi_n \leftarrow$ optimized orientation of the agent
    $(\boldsymbol{p}_n, \psi_n) \leftarrow$ optimization routine on $(\boldsymbol{p}, \psi)$
    for $\boldsymbol{s}_k \in \mathbf{P}$ **do**
        calculate visibility of $\boldsymbol{s}_k$ as in (4.2), (4.3) from $(\boldsymbol{p}_n, \psi_n)$
    calculate the coverage score of the agent from $(\boldsymbol{p}_n, \psi_n)$
    if $new\_score > old\_score - \sigma$ **then**
        new waypoint $\leftarrow (\boldsymbol{p}_n, \psi_n)$
    else
        new waypoint $\leftarrow (\boldsymbol{p}, \psi)$

4.5 Trading

The *trading* procedure is the main part of the coverage task. In this part of the algorithm, we deal with most of the challenging aspects that a coverage mission with a multi-agent cooperative control offers. This part involves actual communication between agents, with exchange of informations about current pose of each agent and on the partition of landmarks currently own by each agent. Namely, each agent a_i involved in a inter-agent communication send its state $(A_i, \mathbf{P}_i)$ to the other end-point of the communication.

An agent can start its trading routine only if it has reached the last waypoint generated by the optimization algorithm. The waypoint is considered reached if the distance between the waypoint and the current position of the agent is below a certain threshold. After the position is reached within the specified threshold, the agent can start the procedure by picking the first agent name in its set $\mathbf{Q}_i$ and sending it a request of trading, with its own state $(A_i, \mathbf{P}_i)$ attached as a parameter of the request. Condition with which we avoid the agents, in our case the real quadcopter, to remain indefinitely unavailable for the trading, because the last waypoint is not physically reachable. In order to maintain consistence in the information about the ongoing coverage mission, a trading procedure can involve only two agents at a time; stated this, we say that an agent already enrolled in a trading routine cannot *serve* another request. If the agent that send the request of trading in the first place cannot be served, it start polling request to the next agents in $\mathbf{Q}_i$ until either it can be served, in which case the actual trading start, or no one of the agents in $\mathbf{Q}_i$ are available to fulfill its request. In the latter case, it change its state to available to accept requests and wait before starting over.

If one of the requests goes through, the first agent, which we call *client* and indicate with c, sends its state to the other agent (the *server*), s, according to Algorithm 4. The latter, implemented as in Algorithm 5, starts a subroutine in which it calculate

the visibility of each landmark in $\mathbf{P}_c$ and $\mathbf{P}_s$ according to poses $(\boldsymbol{p}_c, \psi_c)$ and $(\boldsymbol{p}_s, \psi_s)$, and assign to itself and to the client the new sets $\mathbf{P}_c^*$ and $\mathbf{P}_s^*$ of landmarks that each of them can see better than the other one, as Algorithm 6 shows. In this way, the coverage score associated to new set $\mathbf{Q}_c$, $\mathbf{Q}_s$ from the old poses $(\boldsymbol{p}_c, \psi_c)$ and $(\boldsymbol{p}_s, \psi_s)$ improves. After the landmarks trading, the client and the server start an *optimization routine* as in Algorithm 3, to further improve the coverage score on the new set on landmarks.

It can also happen that two agents are not able to transfer landmarks with each other, because no trading between them, at that point of the mission, will improve the coverage score. In this case, the agents remove the partner of the failed trading from the list of partner with which they can exchange landmarks. That list will be replenished completely after a successful trading with one of the other agent, because in this case the overall partition $\mathbf{P}$ has changed an thus the coverage can be further improved. Namely, if a trading routine involves again the same couple of agents a_i and a_j, the sets $\mathbf{P}_i, \mathbf{P}_j$ will be different from the previous ones. When an agent has $\mathbf{Q} = \emptyset$, it cannot start sending requests of trading and it simply remains available to fulfill other agents' requests. Algorithm 4-6 sums up the trading procedure.

4.6 Termination

From the previous section, it is easy to see that when every agent has $\mathbf{Q} = \emptyset$, meaning that no succesful trading can be further accomplished, the coverage mission is completed.

4.7 Convergence analysis

In this section, we give proof that the coverage score of the team, as defined in (4.5), is monotonically increasing, i.e. it increases after each successful trading procedure and each optimization procedure. Infact the optimization procedure in Algorithms 3 return a pose A_i^* such that

$$C(A_i^*, \mathbf{P}_i) \geq \max_{A \in SE(2)_\Omega} C(A, \mathbf{P}_i) - \sigma.$$

This is due to the property of the visibility function (4.1) and (4.2) to be globally Lipschitz, with respect to the position and orientation of the agent, for any position of the landmark. For such functions, there exist well-known routines for global optimization that are guaranteed to converge to a global optimum, under mild assumption on the optimization domanin. Therefore, we can assume that the optimization algorithm brings the coverage score of a certain agent close to the optimal score for that agent. Furthermore, by Algorithm 6, whenever a landmark is transferred, its visibility improves by at least σ, and therefore the global coverage score (4.5) improves by at least σ after each successful trading.

Moreover, we can prove that when the algorithm terminate, the system is in a Voronoi configuration with tolerance σ. This can be formalized as follows

Algorithm 4 Trading algorithm for client agent $a_c \in \mathcal{A}$

procedure TRADING: CLIENT
    $o\_w \leftarrow$ last waypoint generated by the optimization routine
    $q_c \leftarrow$ client agent
    $q_s \leftarrow$ server agent
    $A_c \leftarrow$ the pose of q_c
    $\mathbf{P}_c \leftarrow$ the landmarks partition of q_c
    $\mathbf{Q}_{c\_in} \leftarrow$ initial $\mathbf{Q}_c$
    $state_c \leftarrow (A_c, \mathbf{P}_c)$
    if o_w is reached **then**
        if $\mathbf{Q}_i \neq \emptyset$ **then**
            pick an item $q_i \in \mathbf{Q}_c$
            $q_i \leftarrow q_s$
            send to q_s *state*
            if q_s is available and $n \neq 0$ landmarks can be trade **then**
                call the optimization routine 3
                $\mathbf{Q}_c = \mathbf{Q}_{c\_in}$
            else if q_s is available and $n = 0$ landmarks can be trade **then**
                $\mathbf{Q}_c = \mathbf{Q}_c \setminus q_s$
        else if $\mathbf{Q}_i = \emptyset$ **then**
            all possible trading completed
            remains available

Algorithm 5 Trading algorithm for server agent $a_s \in \mathcal{A}$

procedure TRADING: SERVER
    $o\_w \leftarrow$ last waypoint generated by the optimization routine
    $q_c \leftarrow$ client agent
    $q_s \leftarrow$ server agent
    $A_s \leftarrow$ the pose of q_s
    $\mathbf{P}_s \leftarrow$ the landmarks partition of q_s
    $state \leftarrow (A_s, \mathbf{P}_s)$
    $\mathbf{Q}_{c\_in} \leftarrow$ initial $\mathbf{Q}_c$
    $state_s \leftarrow (A_s, \mathbf{P}_s)$
    $service\_state \leftarrow$ state of the service
    if o_w is reached and $service\_state$ =True **then**
        wait for requests
        when the first request come, $service_state \leftarrow$False
        take $(A_c, \mathbf{P}_c)$
        call transfer routine 6
        if $n \neq 0$ landmarks can be trade **then**
            call the optimization routine 3
            $\mathbf{Q}_c = \mathbf{Q}_{c\_in}$
        else if $n = 0$ landmarks can be trade **then**
            $\mathbf{Q}_c = \mathbf{Q}_c \setminus q_s$

Algorithm 6 Transfer routine for $a_s, a_c \in \mathcal{A}$

function TRANSFER($state_s, state_c$)
    $\sigma \leftarrow$ threshold for transfers
    $\mathbf{T}_{sc} \leftarrow \{\boldsymbol{s}_k \in \mathbf{P}_s : \text{vis}(A_s, \boldsymbol{s}_k) < \text{vis}(A_c, \boldsymbol{s}_k) - \sigma\}$
    $\mathbf{T}_{cs} \leftarrow \{\boldsymbol{s}_k \in \mathbf{P}_c : \text{vis}(A_c, \boldsymbol{s}_k) < \text{vis}(A_s, \boldsymbol{s}_k) - \sigma\}$
    $\mathbf{P}_s^* \leftarrow (\mathbf{P}_s \setminus \mathbf{T}_{sc}) \bigcup \mathbf{T}_{cs}$
    $\mathbf{P}_c^* \leftarrow (\mathbf{P}_c \setminus \mathbf{T}_{cs}) \bigcup \mathbf{T}_{sc}$
return $\mathbf{P}_s^*, \mathbf{P}_c^*$

Lemma 1. *Consider the solution to a coverage problem implemented as in Algorithms 2-6. When the algorithm terminates, the system is in a Voronoi configuration with tolerance σ.*

Proof. Suppose by contradiction that the Algorithms 2-6 does not bring the system in a Voronoi configuration. This can happen in the two following situations:

1. The coverage score of some agent $a_i \in \mathbf{A}$ is not the maximum possible associated to the partition $\mathbf{P}_i$, i.e. $C(A_i, \mathbf{P}_i) < \max_{\mathbf{A} \in SE(2)_\Omega} C(\mathbf{A}, \mathbf{P}_i) - \sigma$.

2. The visibility of a landmark $\boldsymbol{s}_k \in \mathbf{P}_i$ that belongs to the agent $a_i \in \mathbf{A}$ is lower than the visibility of the same landmark $\boldsymbol{s}_k$ seen by $a_j \in \mathbf{A}$, as $\text{vis}(A_i, \boldsymbol{s}_k) < \text{vis}(A_j, \boldsymbol{s}_k) - \sigma$.

None of these can really happen. Infact, 1. means that the agent can still improve the coverage score on its own partition of landmarks, but according to Algorithms 4-5, each agent is bound to start the Algorithm 3 after each successful trading and before moving to the next waypoint. Regarding 2., if an agent a_i has terminated the task, it means that $\mathbf{Q}_i = \emptyset$, implying that, before terminating, it tried to trade landmarks with each of the other agents without success. This in turns contradicts 2. □

In the next chapter, results of simulations and experiments of the coverage algorithms 2–6 are presented.

5 Simulation and Experiments

In this chapter we present some simulation and experimental results of the coverage algorithm developed in Chapter 4. In each section, we give some general informations about the implementation of the task presented, such as the setup of the agent and the dimensions of the targeted area of the coverage mission. In what follows, we use the words *agents* and *quadcopters* interchangeably.

5.1 Simulations

For the simulations of the coverage mission, we use again RotorSfor monitor the team of quadcopters' movements during navigation and routines in Algorithm 2-6,and ROS/Python to code the coverage task with ROS *nodes*. In this work, we use ROS *Services* to handle the one-to-one interaction between agents that is needed in the trading procedure –cfr. Algorithms 4 and 5– and ROS *Topics*, to collect information about the state of the quadcopters and the ongoing mission. In the simulations, as well as in the experiment with real quadcopters, each agent implementation is splitted up in two part, *Controller* and *Planner*, each of which is treated as a ROS *node*. The Planner node implement all the Algorithms 3-6 with Services, taking in input the current pose of the associated agent through a *subscription* to a topic *published* by the simulator, and giving as output a new waypoint every time the optimization procedure 3 returns the new pose after a successful trading procedure. This waypoint is collected by the Controller node, that implements the controller as in (2.25)-(2.26) with the potential field path planning as in Algorithm 1 of Chapter 3. Finally, this node publishes command to drive the simulated quadcopters in RotorS. An extra node, called *Playground*, is responsible to create the set $\mathbf{S}$ of landmarks, and thus implements the *Initialization* procedure, as in Algorithm 2. This node also takes care of printing the current state $(\mathbf{A}, \mathbf{P})$ of the mission, collecting informations from each Planner node. A sketch of the structure of a single agent is given in Figure 5.1. A video of a coverage mission can be seen at [61]. In this video four agents are used to complete the task. The initial poses of the agents are

$$\boldsymbol{p}^s_{\text{agent1}} = \begin{bmatrix} 3.5 \\ -2 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{p}^s_{\text{agent2}} = \begin{bmatrix} 3.5 \\ 4.5 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{p}^s_{\text{agent3}} = \begin{bmatrix} 3.5 \\ -4 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{p}^s_{\text{agent4}} = \begin{bmatrix} -5 \\ 4 \\ 0 \\ 0 \end{bmatrix}.$$

The Playground initialize the environment as in Figure 5.3. Here, we consider an area of 10×10 meters, and, for the sake of symmetry, each landmark is depicted as a 0.5×0.5

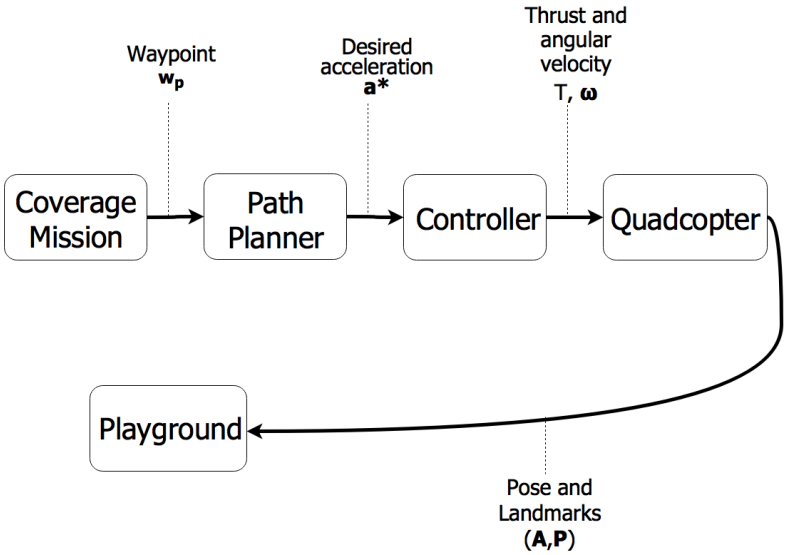


Figure 5.1: Sketch of structure of an agent node.

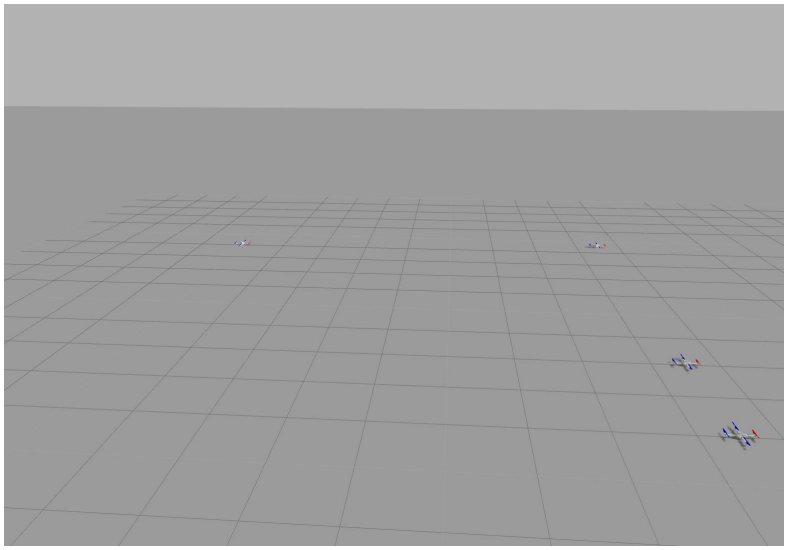


Figure 5.2: Screenshot from the coverage mission simulation in RotorS

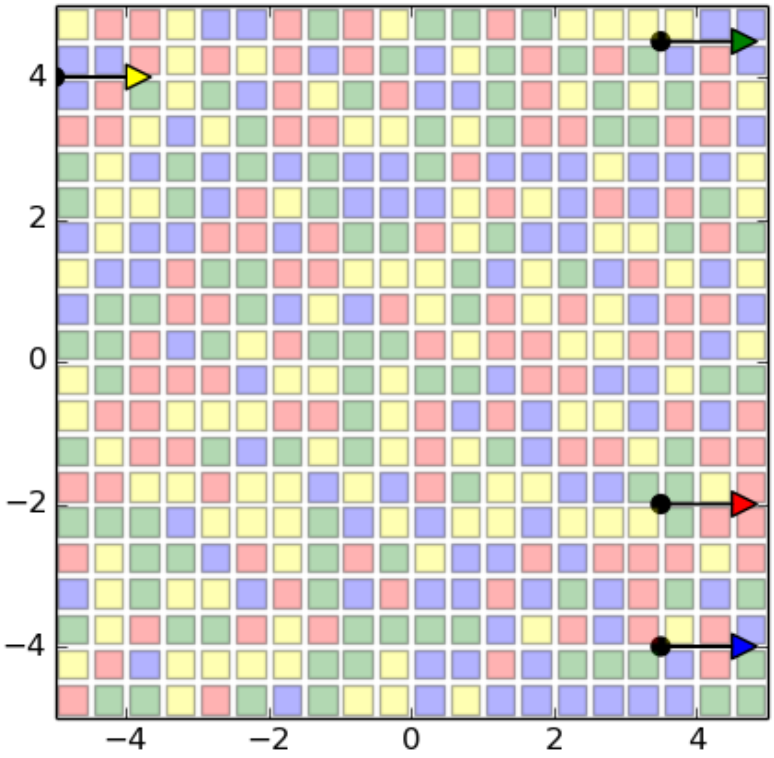


Figure 5.3: Environment as initialized from the Playground node. The coverage score is 21.39

meters square, thus, the environment is abstracted as a set of 400 landmarks. Each landmark is coloured according to the agent to which at every moment it belongs. The agents can optimize their pose within the whole environment. In this initial configuration, one agent lies at the left border of the environment facing inside it, while the others are looking outside of it, so these latter start with a very poor coverage score on their own initial set of landmark. Here, the coverage score $C(\mathbf{A}, \mathbf{P})$ of the whole team start from 21.39. After the initialization process, each agent a_i perform a pose optimization routine on their own partition of landmarks $\mathbf{P}_i$, so to maximize the coverage score on $\mathbf{P}_i$, as in Algorithm 3. Figure 5.4 shows the results of the procedure. The coverage score $C(\mathbf{A}, \mathbf{P})$ of the whole team is now 72.55. A first trading of landmark then happen between the blue and the red agent, and after the optimization process has given the new waypoint to the two agent, the playground looks like the one in Figure 5.5, for which the coverage score is 79.89. After a certain number of iteration, a pattern in the distribution of the landmark arises and the final playground is the one in Figure 5.6. The final coverage score of the whole team is 118.99.

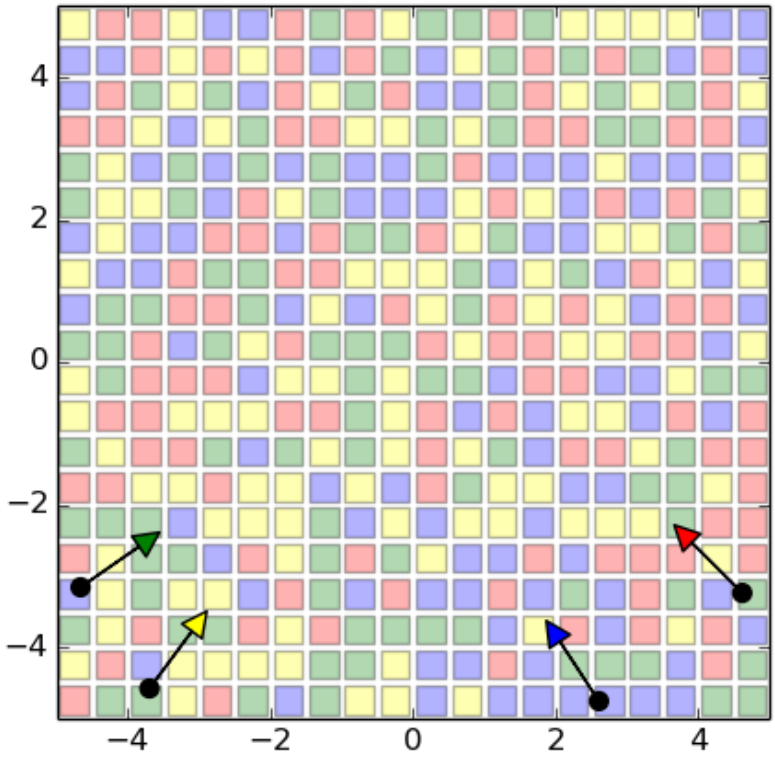


Figure 5.4: Environment after the first pose optimization.
The coverage score is 72.55

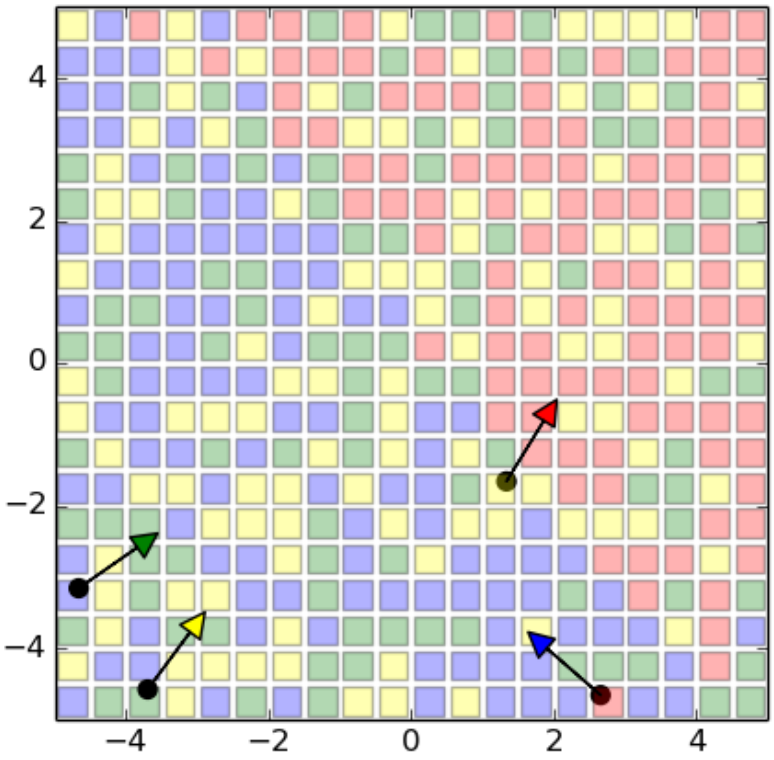


Figure 5.5: Environment after the first landmarks trading.
The coverage score is 79.89

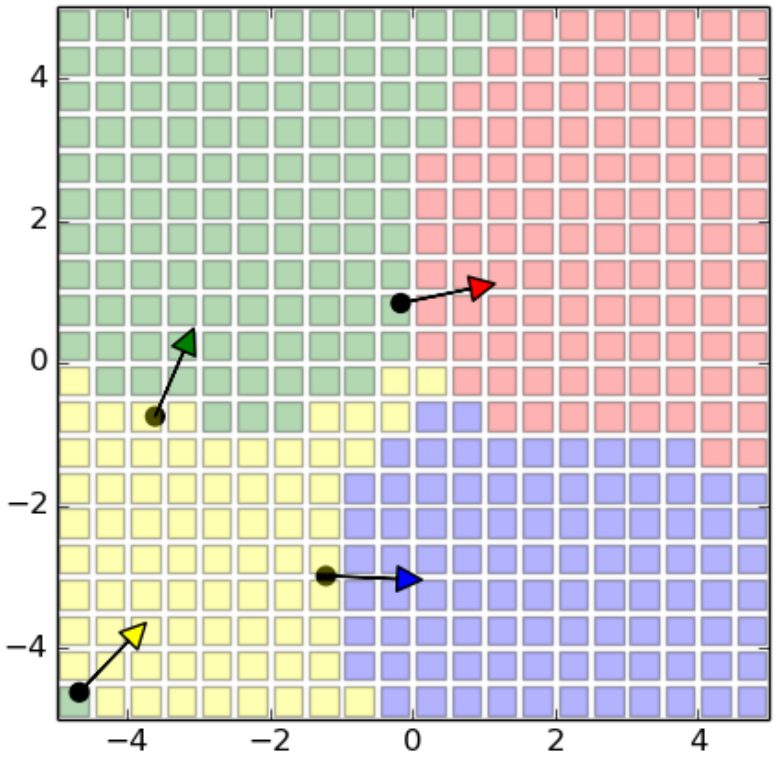


Figure 5.6: Environment repartition at the end of the coverage mission. The coverage score is 118.99

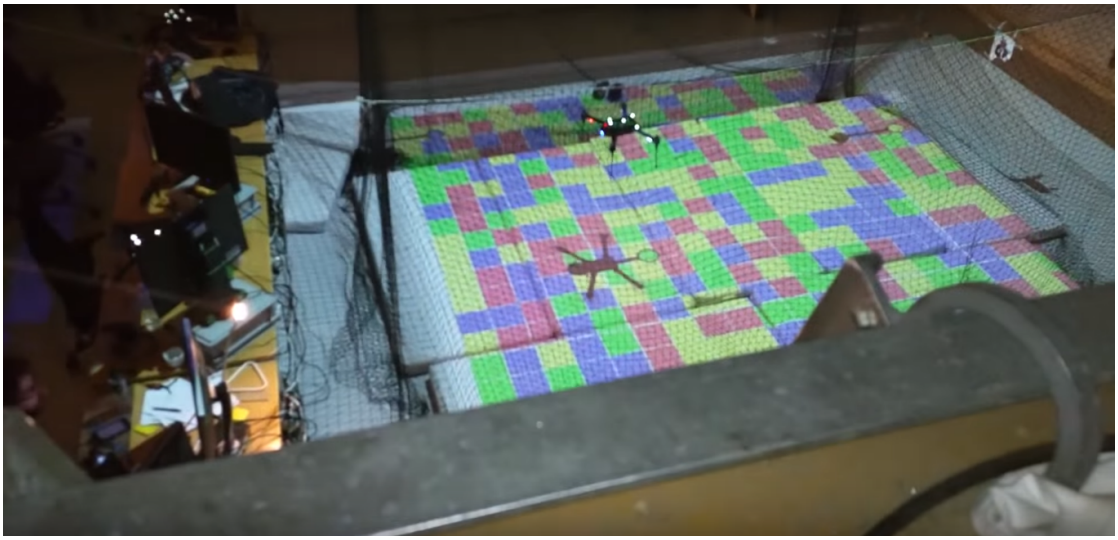


Figure 5.7: Screenshot from the coverage mission experiment with Iris+

5.2 Experiments

We now present the real-world experiment. For space constraints, only one real quadcopter is used to depict an agent involved in the coverage mission, while the other agents are still simulated. In this implementation we dynamically project the changing playground on the floor. For this reason, the environment is abstracted in a set of 500 squared landmark, each 0.20×0.20 meters wide, that span an area of 4×5 meters. The colours give again information on the ownership of the landmarks, while the agent are depicted as double round arrows, with position corresponding to black filled circle, and orientation as the angle between the body of the agent and the direction x. The real quadcopter represent the green agent. Each simulated agent can optimize its pose within the whole area, while the quadcopter is bounded to stay in the flight arena, at a reasonable distance from the net, which leads to an optimization area of 2.2×1.0. In this experiment the initial positions of the agents are

$$\boldsymbol{p}^s_{\text{agent1}} = \begin{bmatrix} 2.5 \\ -2 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{p}^s_{\text{agent2}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{p}^s_{\text{agent3}} = \begin{bmatrix} -1.0 \\ 2 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{p}^s_{\text{agent4}} = \begin{bmatrix} 2.5 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

A video of the experiment can be seen at [62]. Note that, in this implementation, the Playground node can catch *dynamically* the trading of the landmarks, so whenever a certain landmark is projected in white colour, it means that two agents are trading that particular landmark.

6 Conclusion and future work

6.1 Conclusion

In this work, a problem of automatically deploy a team of mobile sensing agents over an environment was addressed. The assignment of the team is to enhance the perception of the area, by means of interaction within the team and local optimization of the pose of each agent. The application developed in this work use a set of UAVs, specifically quadcopters, for acting as sensorized agents. To this aim, we first needed to mathematically model the quadcopter, so to define under which constraints it's capable to fly, and to understand the dynamics of the motion. Then, we implemented a controller based on that model, with a backstepping procedure for the position of the quadcopter, and a non-linear control for its heading angle. With these, we are able to drive the quadcopter to any given referenced position and orientation, as well as follow a pre-planned trajectory. In order to safely fly multiple quadcopters on the same environment, we developed an extra layer of control above the previous one. With this layer we are able to online generate the trajectory towards a given position while taking care of avoiding collision with obstacle present in the environment. In other words, we gave the quadcopter the ability to be aware of the position of the other agents, as well as of any other kind of obstacles, and to dynamically react and change its trajectory to avoid the collision.These steps were preparatory to face the coverage problem as the main objective of this work. Once we fulfilled all the previous tasks, we started dealing with the coverage mission by giving some general definitions, needed to mathematically structure the problem. For that reason, we defined a visibility function to describe the characteristics of the mobile sensor. We gave a formal description of the form of the solution that we were aiming to find, and we presented our approach to solve the problem, based on the abstraction of the environment and on the definitions of the interaction between agents. We presented our algorithm to drive the system in a configuration that is shown to be sub-optimal. Proof of convergence of the algorithm were also given, as well as proof of the optimality of the solution found. Finally, we tested the implemented algorithms in a simulated environment, as well as by using physical quadcopter, and we gave the results of each experiment.

6.2 Future work

Usually, when dealing with widely discussed and relatively recent research topic, every work is far to be comprehensive. Although we presented a self sufficient solution to a coverage problem, that can be applied as it is on a real coverage mission, nonetheless

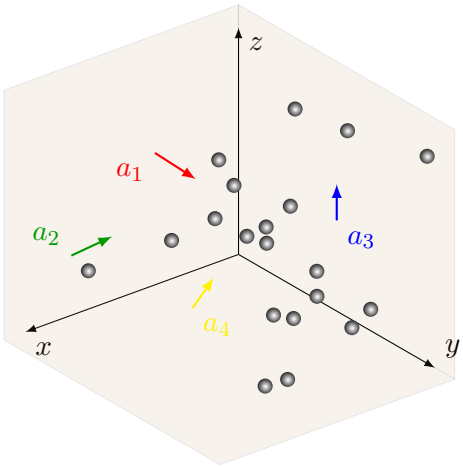


Figure 6.1: Representation of a 3D coverage task, where the landmarks are three-dimensional position.

this work leave an open path to future developments and extensions.

The first future extension to the coverage algorithm presented is the three-dimensional coverage. In this case, the environment will be a subset of $\mathbb{R}^3$, thus the problem is to deploy a team of mobile sensing agents a_i with $i = 1 \dots n$, with pose $A_i \in SE(3)$, in order to maximize the coverage score of the whole team with respect to a set of landmarks $\mathbf{S}$, where each $\boldsymbol{s}_k \in \mathbf{S}$ is a point in $\mathbb{R}^3$. In this scenario, the visibility function will have to be defined to take into account the third dimension, and the optimization process will be bound over a different subset of pose, accordingly to the modified environment. A representation of the three-dimensional coverage is given in Figure 6.1, where we again consider four agents, $a_1 \dots a_4$, and the grey spheres depicts the landmarks.

A second extension, that requires little modification to the approach presented here, is the structural inspection. The structure to inspect must be abstracted into a set of landmarks as well. However, in this case, each landmark will be defined with a position $s_k \in \mathbb{R}^2$, as well as with an orientation $\varphi_k \in [-\pi, \pi)$, hence a pose $Q \in SE(2)$, where in this case the orientation is defined by the curvature of the structure in the neighbourhood of the landmark. Figure 6.2 depicts the environment in this scenario. Hence, the visibility of a landmark from the point of view of a certain agent depends on the orientation of the agent with respect to the landmark, and on the orientation of the landmark with respect to the agent. Namely, the maximum visibility of a landmark is achieved if the agent and the landmark lie one in front of the other, plus of course if they lie at the optimal distance. One mathematical way to capture these characteristics is a visibility function given by $\mathrm{vis}(A, Q) = \mathrm{vis}_{I_3}(A^{-1}Q)$, where

$$\mathrm{vis}_{I_3}(A^{-1}Q) = \begin{cases} 0, & \text{if } s_x \leq 0 \vee \cos(\varphi) \geq 0, \\ s_x \cos(\varphi), & \text{if } \|\boldsymbol{s}\| \leq 1, \\ s_x \cos(\varphi)/\|\boldsymbol{s}\|^2, & \text{if } \|\boldsymbol{s}\| > 1. \end{cases}$$

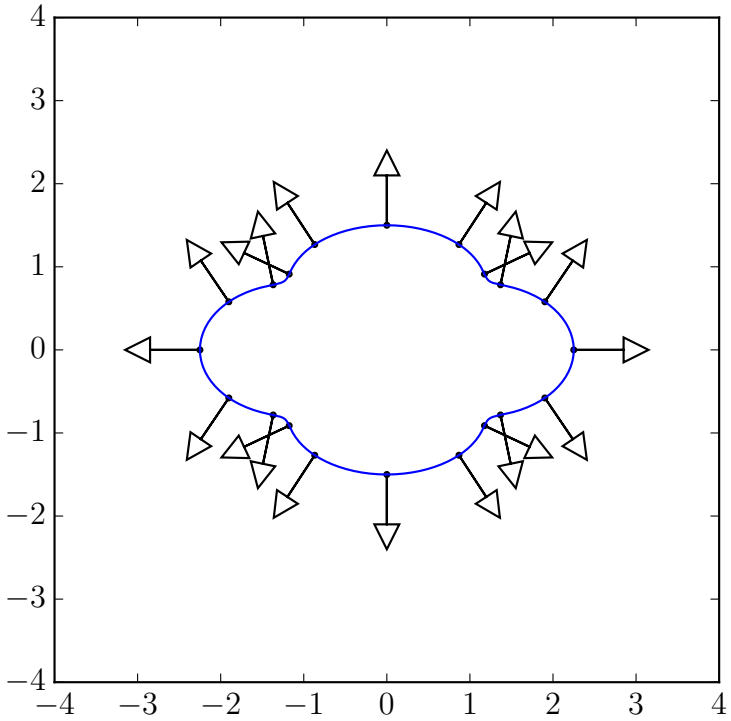
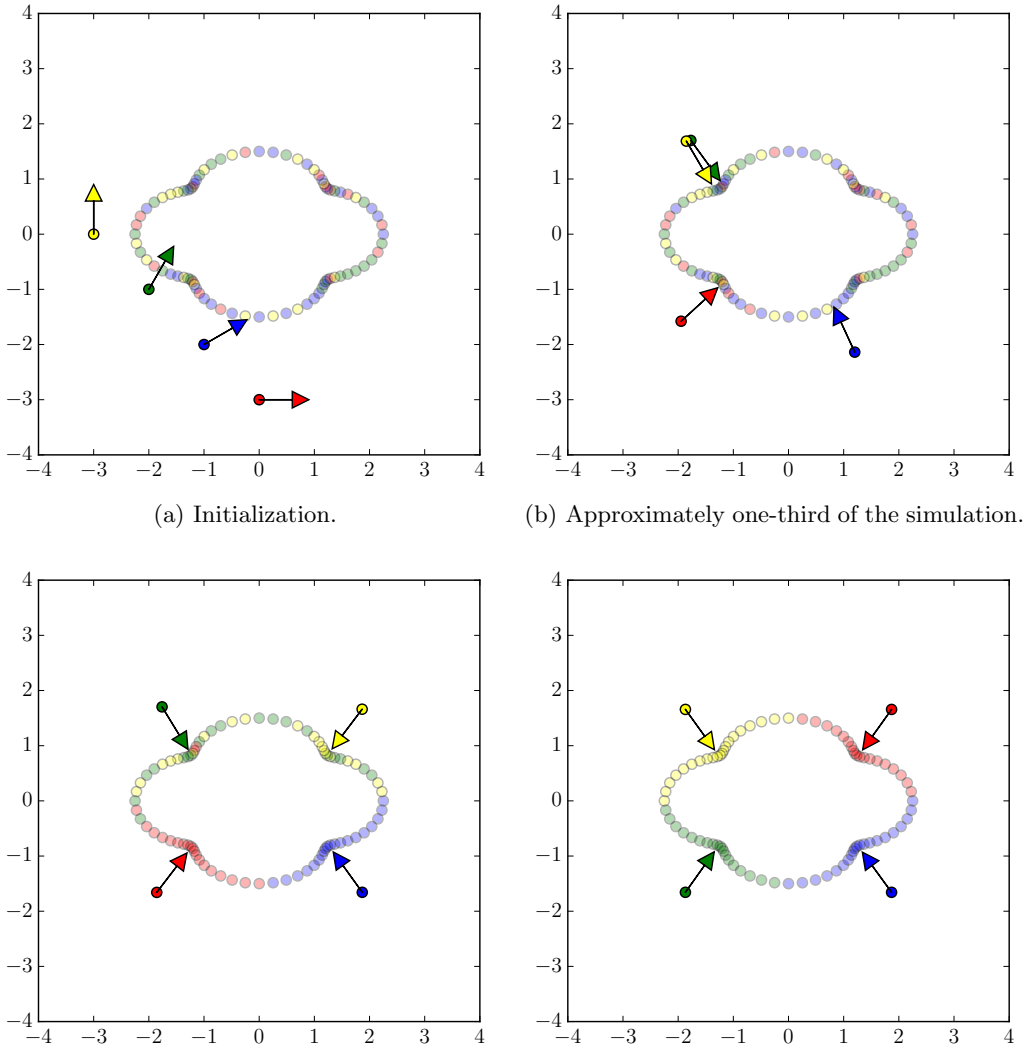


Figure 6.2: Example of a possible abstraction for a structural inspection mission. The structure is abstracted as a curve in $\mathbb{R}^2$, which is further abstracted into a set of landmarks.

A possible execution of the Algorithms 2–6 in Chapter 4, with the visibility function 6.2, on the environment in Figure 6.2, is depicted in Figure 6.3, where it is shown that the four agents $a_1, \dots a_4$, after a random initialization of the landmarks, converge to a sub-optimal configuration with respect of their partition of landmarks. Moreover, the extension of the coverage algorithm for the structural inspection, presents the same convergence properties of the original algorithm, both in theory and in simulation.



(a) Initialization.

(b) Approximately one-third of the simulation.

(c) Approximately two-third of the simulation.

(d) Final configuration.

Figure 6.3: Simulation of an inspection mission

List of Figures

Bibliography

[1] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[2] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed," *IEEE Robotics Automation Magazine*, vol. 17, no. 3, 2010.

[3] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, 2013.

[4] S. Gupte, P. I. T. Mohandas, and J. M. Conrad, "A survey of quadrotor unmanned aerial vehicles," in *IEEE Southeastcon*, 2012.

[5] L. Besnard, Y. B. Shtessel, and B. Landrum, "Quadrotor vehicle control via sliding mode controller driven by sliding mode disturbance observer," *Journal of the Franklin Institute*, vol. 349, no. 2, 2012.

[6] A. Mokhtari, N. K. M'Sirdi, K. Meghriche, and a. Belaidi, "Feedback linearization and linear observer for a quadrotor unmanned aerial vehicle," *Advanced Robotics*, vol. 20, no. 1, 2006.

[7] H. Voos, "Nonlinear control of a quadrotor micro-UAV using feedback-linearization," in *IEEE International Conference on Mechatronics, (ICM)*, 2009.

[8] J. Colorado, A. Barrientos, A. Martinez, B. Lafaverges, and J. Valente, "Mini-quadrotor attitude control based on hybrid backstepping & frenet-serret theory," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[9] M. G. Earl and R. D'Andrea, "Real-time attitude estimation techniques applied to a four rotor helicopter," in *IEEE Conference on Decision and Control (CDC)*, 2004.

[10] O. Magnussen, M. Ottestad, and G. Hovland, "Experimental validation of a quaternion-based attitude estimation with direct input to a quadcopter control system," in *International Conference on Unmanned Aircraft Systems, (ICUAS)*, 2013.

[11] I. Sa and P. Corke, "Estimation and Control for an Open-Source Quadcopter," in *Australian Conference on Robotics and Automation (ACRA)*, 2011.

[12] J. Zhang, F. Meng, Y. Zhou, G. Lu, and Y. Zhong, "Decentralized Formation Control of Multi-UAV Systems under Wind Disturbances," *Chinese Control Conference (CCC)*, 2015.

[13] V. Mistier, a. Benallegue, and N. K. M'Sirdi, "Exact linearization and noninteracting control of a 4 rotors helicopter via dynamic feedback," in *IEEE International Workshop on Robot and Human Interactive Communication (ROMAN)*, 2001.

[14] D. Mellinger and V. Kumar, "Minimum Snap Trajectory Generation and Control for Quadrotors," in *International Conference on Robotics and Automation (ICRA)*, 2011.

[15] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, 2012.

[16] H. Lee, H. Kim, and H. J. Kim, "Path Planning and Control of Multiple Aerial Manipulators for a Cooperative Transportation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[17] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, 1986.

[18] M. Saska, J. Chudoba, L. Precil, J. Thomas, G. Loianno, A. Tresnak, V. Vonasek, and V. Kumar, "Autonomous deployment of swarms of micro-aerial vehicles in cooperative surveillance," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014.

[19] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *American Control Conference (ACC)*, 2002.

[20] A.-L. Barabási, "Taming complexity," *Nature Physics*, vol. 1, no. 2, 2005.

[21] J. Fax and R. Murray, "Information Flow and Cooperative Control of Vehicle Formations," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, 2004.

[22] M. Turpin, N. Michael, and V. Kumar, "Decentralized formation control with variable shapes for aerial robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[23] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, no. 3, 2006.

[24] A. Jadbabaie and A. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, 2003.

[25] T. Lee, "Collision Avoidance for Quadrotor UAVs Transporting a Payload via Voronoi Tessellation," in *American Control Conference (ACC)*, 2015.

[26] K. Sreenath and V. Kumar, "Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots," *Rn*, vol. 1, no. r2, 2013.

[27] F. Augugliaro, "Knot-tying with Flying Machines for Aerial Construction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[28] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, 1982.

[29] J. Cortés, S. Martínez, T. Karatas, F. Bullo, and S. Member, "Coverage Control for Mobile Sensing Networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, 2004.

[30] J. Le Ny and G. J. Pappas, "Adaptive deployment of mobile robotic networks," *IEEE Transactions on Automatic Control*, vol. 58, no. 3, 2013.

[31] M. Zhong and C. G. Cassandras, "Distributed coverage control and data collection with mobile sensor networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, 2011.

[32] R. W. Beard and T. W. McLain, "Multiple UAV cooperative search under collision avoidance and limited range communication constraints," in *IEEE Conference on Decision and Control (CDC)*, 2003.

[33] A. Gusrialdi, S. Hirche, T. Hatanaka, and M. Fujita, "Voronoi based coverage control with anisotropic sensors," in *American Control Conference (ACC)*, 2008.

[34] A. Gusrialdi, T. Hatanaka, and M. Fujita, "Coverage control for mobile networks with limited-range anisotropic sensors," in *IEEE Conference on Decision and Control (CDC)*, 2008.

[35] K. Laventall and J. Cortés, "Coverage control by robotic networks with limited-range anisotropic sensory," in *American Control Conference (ACC)*, 2008.

[36] B. Hexsel, N. Chakraborty, and K. Sycara, "Coverage control for mobile anisotropic sensor networks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[37] Y. Kantaros, M. Thanou, and A. Tzes, "Visibility-oriented coverage control of mobile robotic networks on non-convex regions," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[38] J. W. Durham, R. Carli, P. Frasca, and F. Bullo, "Discrete partitioning and coverage control for gossiping robots," *IEEE Transactions on Robotics*, vol. 28, no. 2, 2012.

[39] "ROS. Robot Operative System," http://wiki.ros.org.

[40] "Python Software Foundation," https://www.python.org/.

[41] "Ubuntu Operative System," http://www.ubuntu.com/.

[42] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Springer International Publishing, 2016.

[43] "3D Robotics Iris+ drone," https://3dr.com/kb/iris/.

[44] "MAVLink extension for ROS," https://github.com/mavlink/mavros.

[45] "Qualisys motion capture system," http://www.qualisys.com/.

[46] "Smart Mobility Lab at KTH," https://www.kth.se/ees/forskning/strategiska-forskningsomraden/intelligenta-transportsystem/smart-mobility-lab.

[47] "Aeroworks 2020, european project," http://www.aeroworks2020.eu/.

[48] H. K. Khalil, *Nonlinear systems, 3rd edition*. Prentice Hall, 2002.

[49] "A Leader-Following task implemented in RotorS," https://youtu.be/qcsL1COQveA.

[50] "A Leader-Following task implemented with an Iris+ quadcopter following a marked stick," https://youtu.be/sRr8XmvFa-s.

[51] A. Richards and J. How, "Decentralized model predictive control of cooperating UAVs," in *IEEE Conference on Decision and Control (CDC)*, 2004.

[52] Y. Kuriki and T. Namerikawa, "Experimental validation of cooperative formation control with collision avoidance for a multi-UAV system," in *International Conference on Automation, Robotics and Applications (ICARA)*, 2015.

[53] "A collision avoidance Swapping task made in RotorS," https://youtu.be/LUY2WIp80HY.

[54] "A collision avoidance task made with Iris+," https://youtu.be/Fwk1q41C9gA.

[55] R. W. J.B. Campbell, *Introduction to Remote Sensing, 5th Edition*. Guilford Publications, 2011.

[56] A. Adaldo, M. Sposato, D. V. Dimarogonas, and K. H. Johansson, "Distributed Position and Orientation Control for Discrete Coverage with Application to Aerial Robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Submitted to review*, 2016.

[57] J. D. Pintér, *Global optimization in action: continuous and Lipschitz optimization: algorithms, implementations and applications*. Springer Science & Business Media, 2013.

[58] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial tessellations: concepts and applications of Voronoi diagrams*. John Wiley & Sons, 2009.

[59] Q. Du, V. Faber, and M. Gunzburger, "Centroidal voronoi tessellations: applications and algorithms," *SIAM review*, vol. 41, no. 4, 1999.

[60] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, 1997.

[61] "A coverage mission simulated in RotorS," https://youtu.be/xiHn8nWkUFs.

[62] "A coverage mission experiment with Iris+," https://youtu.be/dfz1Rn6hnAE.

TRITA TRITA-EE 2016:065