CIMAT Centro de Investigación en Matemáticas, A.C.

DISTRIBUTED MODEL PREDICTIVE CONTROL FOR FORMATION OF QUADROTORS

T E S I S

Que para obtener el grado de

Maestro en Ciencias

con Especialidad en

Computación y Matemáticas Industriales

Presenta

Salim Vargas Hernández

Directores de Tesis:

Dr. Héctor Manuel Becerra Fermín

Dr. Jean Bernard Hayet

[illegible]

Autorización de la versión final
Dr. Héctor Manuel Becerra Fermín

[illegible]

Autorización de la versión final
Dr. Jean Bernard Hayet

Guanajuato, Gto., diciembre de 2020

Acknowledgements

I want to thank CIMAT for being an academically rigorous institution that also provided me with a lot of personal support.

I would like to express my deepest gratitude to my thesis directors Héctor Becerra and Jean-Bernard Hayet, without their wise guidance, remarkable patience and persistent help this thesis would not have been possible.

Special thanks to Antonio Sánchez for all his technical support during the experiments with real quadrotors in the robotics laboratory. Thank you for all the fun and mutually enriching conversations.

I wish to acknowledge the support and great love of my family. They never let me down and always believe in me. Also my friends, including of course the ones that I made at CIMAT, provided me with unwavering support and motivation. Thank you for all the funny experiences and insightful conversations.

I want to thank CONACYT for all the support given in order to successfully complete this journey. Also I would like to thank Intel Labs for the financial help provided in order to finish this thesis and the trust placed in my work.

To all the wonderful people that saved my life

Abstract

This thesis addresses the problem of controlling a group of quadrotors in a distributed way to achieve desired formations with guarantee of collision avoidance and connectivity maintenance. The obstacles can be other agents or fixed obstacles in the environment. The connectivity is modeled through a graph, being neighbor agents those connected by an edge. The maximum distance between neighbor agents is limited by a constant to maintain connectivity. A Model Predictive Control (MPC) strategy has been proposed to solve the formulated problem, which is an effective control strategy with flexibility to consider constraints relative not only to the evasion and connectivity between agents, but also to elements of the environment, such as dimensions of the space where the quadrotors can move. Moreover, the predictive component of this control technique allows the agents to anticipate (in a time window) the realization of the required actions to guarantee the agents not to collide and maintain connectivity.

Formation control is addressed as a consensus problem of virtual agents, which are related to the real ones by constant displacement vectors that define the shape of the formation. Two ways of reaching consensus of the virtual agents have been explored, the first one generating trajectories for consensus and then tracking them using MPC, and the second one introducing directly the local consensus error at each iteration to the MPC cost function. Regarding obstacle avoidance and connectivity maintenance, it was evaluated the use of hard and/or soft constraints in the MPC optimization problem. Furthermore, both decentralized and distributed control architectures were studied; in the first, only the agents position is communicated between neighbors, while in the second, both position and control inputs are shared.

Based on simulation results, the best formation control strategy resulted on using the local consensus error directly in the MPC cost function, and including both hard and soft constraints for obstacle avoidance and connectivity maintenance, all of this inside a distributed control architecture. This final strategy showed advantages in the smoothness of the generated paths, the velocity of convergence to the desired formation, and the effectiveness to avoid obstacles. It was shown in both realistic simulations and real experiments with quadrotors.

Contents

Chapter 1

Introduction

1.1 Motivation

The last decade, the study of Multi-Agent Systems (MASs) has attracted considerable attention due to the wide applications of these kind of systems. From a practical point of view, multiple agents have improved capabilities when their work cooperatively in comparison to the capabilities of a single agent. The control of a MAS may involve many different related research topics and problems, such as consensus, formation control, flocking, coverage control, among others. The general interest is that multiple autonomous agents work together to achieve a global task or a collective group behavior.

In cooperative control of MASs for robotics applications, one of the main areas of interest is formation control. In this task, given any initial configuration of the robots, the objective is to achieve a predefined formation using only information exchanged between agents. Usually the target is to achieve a certain shape in the formation, not necessarily at a particular absolute place in the space but respecting the relative constraints imposed by the specified formation.

The formation control task is closely related with the evasion task, in the sense that, as the robots try to achieve the formation, they also need to avoid collisions with other agents and with any other obstacle present in the environment. This second task increases the difficulty of the control as not many formation control strategies easily allow to consider obstacle avoidance. The realization of both tasks, formation and obstacle avoidance, must be guaranteed by using a distributed strategy considering only information of neighboring agents in the MAS and local information of the obstacles position in the environment.

On the other hand, with the recent improvement of hardware capabilities, Model Predictive Control offers a very flexible strategy for formation control. Its main advantage is the simplicity of its formulation when considering the evasion task; as this control strategy is based on solving an optimization problem, the evasion task can be introduced to the formation task by just adding hard or soft constraints to the optimization problem in a very intuitive way. Also, Model Predictive Control, as its name refers, is based on predicting the future movements of the controlled agent and, in some cases, of other agents, and it uses this information to anticipate the evolution of the system instead of reacting to instantaneous changes.

In this thesis, a Model Predictive Control strategy is presented for the formation control problem, specifically in the case of quadrotors, which are modeled at kinematic level. The objective is that a group of quadrotors reaches a desired formation in tridimensional position. The formulation considers evasion but also connectivity maintenance, which means that a connectivity model of the group of agents is maintained along the missions. Along with the theoretical formulation, some simulations and experimental results on real quadrotors are presented.

1.2 Outline of the work

In Chapter 2, the theoretical background for the full understanding of this thesis is presented, such as the dynamic model of a quadrotor and a possible low level control, the architectures for controlling a MAS, the fundamentals about consensus theory and its application to formation control, and the basics of Model Predictive Control.

In Chapter 3, some of the most recent works regarding both Model Predictive Control and formation control, and their application to quadrotors are reviewed.

Chapter 4 is the main chapter of this thesis, the proposed control strategies for formation control with quadrotors are detailed. Two control architectures, two forms of solving formation tasks and two ways of leading with evasion and connectivity maintenance are presented. Finally, some discussion is done in order to choose the best combination of strategy components based on simulation results.

In Chapter 5, performance results of the final control strategy proposed in Chapter 4 are presented in simulations in the dynamic simulator `Gazebo`, and in real experiments with *Bebop 2.0* quadrotors.

Finally, Chapter 6 corresponds to the conclusions of this thesis.

Chapter 2

Preliminaries

In this chapter, we are going to present some background topics that we are going to use in the rest of the chapters of this work. First, in Section 2.1, we present the dynamic model of a quadrotor and describe some basic movements of this aircraft. Then, in Section 2.2, we introduce a low level velocity control for a quadrotor via feedback linearization. Three main control architectures for multiple agents systems are explained in Section 2.3. The basics of consensus theory and its relation with formation control are presented in Section 2.4. Finally, Section 2.5 is an introduction to Model Predictive Control.

2.1 Dynamic model of a quadrotor

A quadrotor or quadcopter is an aircraft with four rotors, each of them being located on an extremity of the robot. Figure 2.1 shows a schema of a typical quadrotor. The red sphere corresponds to the main body and the four disks to the rotors. To be able to fly in a stable way, two of the propellers rotate counter-clockwise (the front and rear ones in blue), and the other two rotate clockwise (the left and right ones in green), each pair canceling in some way the moment generated by the other. In the following, we are going to present the dynamic model of a quadrotor given in [Sab15]. All the physics details can be consulted in this reference. Also, some considerations on blade flapping and induced drag effects, which are not tackled here, can be consulted in [MKC12].

Using the Newton-Euler-based representation of the orientation of a quadrotor, we can represent the internal state of the system by 12 variables. Six of them refer to the position $x,\ y,\ z$ of the center of mass of the quadrotor in a inertial reference frame (given by $\{X_O, Y_O, Z_O\}$ in Figure 2.1) and their respective velocities $v_x,\ v_y,\ v_z$. Given

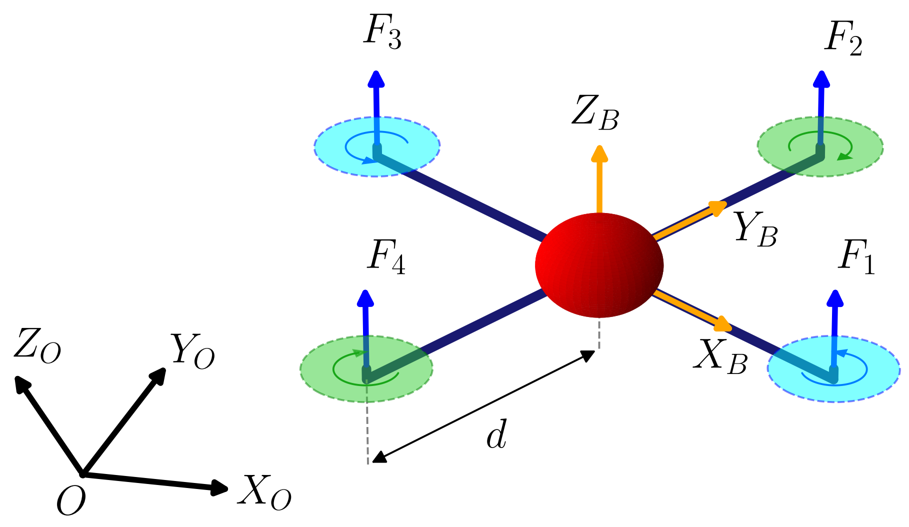


Figure 2.1: Quadrotor model.

the inertial frame of reference and the body frame $\{X_B, Y_B, Z_B\}$, the other six variables refer to the Euler angles ϕ, θ, ψ and their velocities ω_x, ω_y, ω_z. Hence, the state variables are

$$\boldsymbol{x} = \begin{bmatrix} x & y & z & v_x & v_y & v_z & \phi & \theta & \psi & \omega_x & \omega_y & \omega_z \end{bmatrix}^T. \tag{2.1}$$

A quadrotor is a system with four inputs, considering as inputs of the system the total thrust $u_1 = F_1+F_2+F_3+F_4$, where each F_i is the individual trust on propeller i, and the torques for the three Euler angles (pitch, roll and yaw) u_2, u_3 and u_4, respectively. These inputs can be gathered in a vector $\boldsymbol{u} = (u_1, u_2, u_3, u_4)^T$, and the quadrotor dynamics can be expressed as

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}, \tag{2.2}$$

where

$$f(\boldsymbol{x}) = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \\ 0 \\ -g \\ w_x + \sin(\phi)\tan(\theta)w_y + \cos(\phi)\tan(\theta)w_z \\ \cos(\phi)w_y - \sin(\phi)w_z \\ \sin(\phi)\sec(\theta)w_y + \cos(\phi)\sec(\theta)w_z \\ \frac{I_y - I_z}{I_x} w_y w_z \\ \frac{I_z - I_x}{I_y} w_x w_z \\ \frac{I_x - I_y}{I_z} w_x w_y \end{bmatrix} \tag{2.3}$$

and

$$g(\boldsymbol{x}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m}\left(\cos(\phi)\cos(\psi)\sin(\theta) + \sin(\phi)\sin(\psi)\right) & 0 & 0 & 0 \\ \frac{1}{m}\left(\cos(\phi)\sin(\psi)\sin(\theta) - \sin(\phi)\cos(\psi)\right) & 0 & 0 & 0 \\ \frac{1}{m}\left(\cos(\phi)\cos(\theta)\right) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \dfrac{d}{I_x} & 0 & 0 \\ 0 & 0 & \dfrac{d}{I_y} & 0 \\ 0 & 0 & 0 & \dfrac{1}{I_z} \end{bmatrix}, \tag{2.4}$$

where m corresponds to the mass of the quadcopter, d is the distance from any rotor to the center of the quadcopter, g is the value of gravity constant and I_x, I_y and I_z are the diagonal elements of the inertia matrix of the quadcopter. The equations 2.3 and 2.4 are clearly non linear, as the expressions involve, among others, trigonometric functions of some parts of the state.

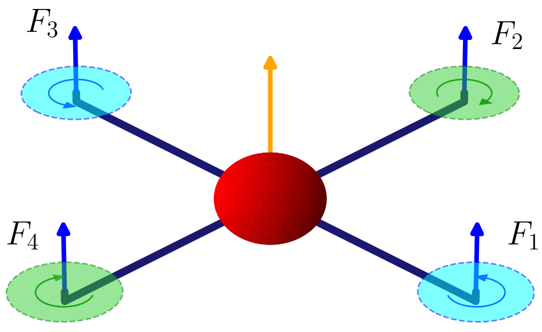
One can show that the vector of inputs $\boldsymbol{u}$ is related to the velocity of the i-th rotor

$\bar{\omega}_i$, $i = 1, \ldots, 4$ by the following expression [Sab15]

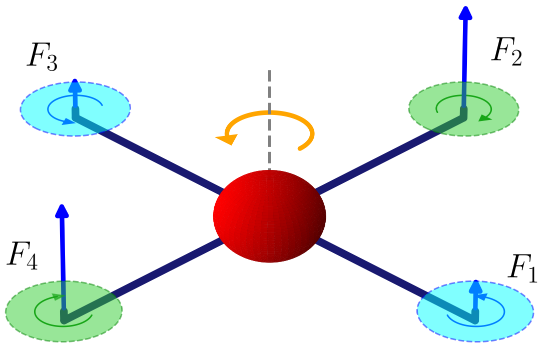
$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ -\frac{\sqrt{2}}{2} d\, c_T & -\frac{\sqrt{2}}{2} d\, c_T & \frac{\sqrt{2}}{2} d\, c_T & \frac{\sqrt{2}}{2} d\, c_T \\ -\frac{\sqrt{2}}{2} d\, c_T & \frac{\sqrt{2}}{2} d\, c_T & \frac{\sqrt{2}}{2} d\, c_T & -\frac{\sqrt{2}}{2} d\, c_T \\ -c_Q & c_Q & -c_Q & c_Q \end{bmatrix} \begin{bmatrix} \bar{\omega}_1^2 \\ \bar{\omega}_2^2 \\ \bar{\omega}_3^2 \\ \bar{\omega}_4^2 \end{bmatrix},$$

where c_T is the thrust factor and c_Q is the drag factor of the quadcopter.

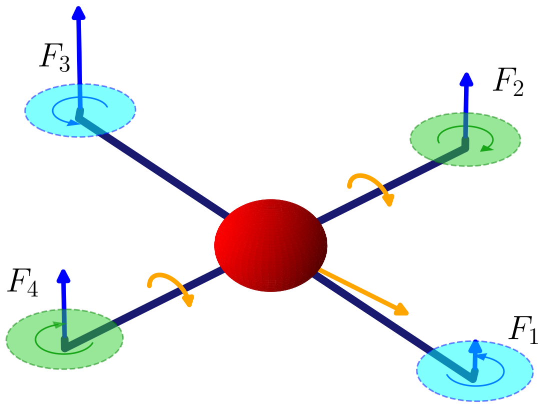
We can identify four basic movements of a quadrotor: vertical displacement due to the total thrust and rotational movements on yaw ψ, pitch ϕ and roll θ. Figure 2.2 shows these four movements and the thrusts needed on each rotor to achieve them.



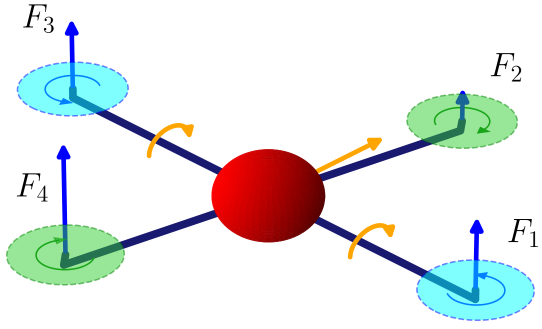
(a) Thrust.



(b) Yaw ψ.



(c) Pitch ϕ.



(d) Roll θ.

Figure 2.2: The four basic movements of a quadrotor.

The vertical movement of a quadrotor corresponds to the total thrust, that is equal to the sum of the individual thrusts of the propellers. The yaw ψ movement is achieved by unbalancing the thrusts of the counter-clockwise rotors and the clockwise ones; a reactive torque is then produced by the rotors with the greater velocities. The pitch ϕ and roll θ motions correspond to unbalanced velocities between opposite rotors. These types of movements also lead to horizontal displacement, i.e. they allow quadrotors to move forward or sideways.

Given that a quadrotor has 3 position and 3 rotational states and only four inputs, one per rotor, a quadrotor is an underactuated system, and its control is not an easy task. Also, we have to remark that this dynamic model is based on ideal situations and that the control of a real quadrotor must overcome some unmodeled physical effects. Due to the underactuated nature of the system, some cascaded control strategies have been conceived such as in [MKC12] and [LLM10]. Having said this, in the next section we are going to present a feedback linearization control for a quadrotor, which is also a studied strategy for controlling a quadrotor in [Sab15] and [MBM01].

2.2 Feedback linearization for low level control

Feedback linearization is a control strategy that transforms a non-linear system into a linear one by applying an appropriate change of variables and control input. To be linearizable by feedback, a system requires to be affine in the control, and that means it should be of the form

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u},$$
$$\boldsymbol{y} = h(\boldsymbol{x}).$$

Let us now consider a change of variables $\boldsymbol{z} = \mathcal{T}(\boldsymbol{x})$ given by a diffeomorphism $\mathcal{T}$ such that $\mathcal{T}(0) = 0$. The proposed control input is searched to be of the form

$$\boldsymbol{u} = \alpha(\boldsymbol{x}) + \beta(\boldsymbol{x})\boldsymbol{\nu}, \tag{2.5}$$

such that the dynamics of the transformed system $\dot{\boldsymbol{z}}$ is a *linear function of the control input*, that is

$$\dot{\boldsymbol{z}} = b(\boldsymbol{x}) + \Delta(\boldsymbol{x})\boldsymbol{u},$$

where $\boldsymbol{v}$ is an auxiliary control input. In that case, the functions $\alpha(\boldsymbol{x})$ and $\beta(\boldsymbol{x})$ will be necessarily

$$\alpha(\boldsymbol{x}) = -\Delta^{-1}(\boldsymbol{x})b(\boldsymbol{x}), \tag{2.6}$$
$$\beta(\boldsymbol{x}) = \Delta^{-1}(\boldsymbol{x}). \tag{2.7}$$

Using the control in (2.5) with the functions (2.6) and (2.7), the dynamics of the transformed system $\dot{\boldsymbol{z}}$ result in

$$\dot{\boldsymbol{z}} = b(\boldsymbol{x}) - \Delta(\boldsymbol{x})\Delta^{-1}(\boldsymbol{x})b(\boldsymbol{x}) + \Delta(\boldsymbol{x})\Delta^{-1}(\boldsymbol{x})\boldsymbol{\nu} = b(\boldsymbol{x}) - b(\boldsymbol{x}) + \boldsymbol{\nu} = \boldsymbol{\nu}. \tag{2.8}$$

Equation (2.8) suggests that if a control input $\boldsymbol{u}$ satisfying (2.5), (2.6) and (2.7) can be found, then the dynamics of the output $\dot{\boldsymbol{z}}$ could be set easily in terms of an auxiliary control input $\boldsymbol{\nu}$.

We are interested in the formulation of a velocity control using feedback linearization, this because we will propose a high-level control for formation tasks that give us linear velocities as outputs, and these velocities will be the input of this lower-level control. Then, considering the quadrotor dynamics described in (2.2) and choosing an output of the same length as the input, the output $\boldsymbol{y}$ for controlling the linear velocities of the quadrotor (v_x, v_y, v_z) and the yaw angle ψ can be established as the following

$$\boldsymbol{y} = h(\boldsymbol{x}) = \begin{bmatrix} v_x & v_y & v_z & \psi \end{bmatrix}^T. \tag{2.9}$$

The *relative degree* is the number of times an output component needs to be differentiated in order for at least one component of the input to appear. That is

$$r_i = (\inf k \mid \exists j,\ 1 \le j \le 4,\ L_{g_j} L_f^{k-1} h_i \neq 0),$$

where L_f and L_{g_j} are the Lie derivatives with respect to f and g_j respectively. Given a scalar function $h : \mathbb{R}^n \longrightarrow \mathbb{R}$ and a vector field $f : \mathbb{R}^n \longrightarrow \mathbb{R}^n$, the Lie derivative or directional derivative of h in the direction of f is defined as the scalar function

$$L_f h = \nabla h \cdot f,$$

where ∇h denotes the gradient vector that gathers all the partial derivatives of h

$$\nabla h = \begin{bmatrix} \frac{\partial h}{\partial x_1} & \frac{\partial h}{\partial x_2} & \cdots & \frac{\partial h}{\partial x_n} \end{bmatrix}^T.$$

The Lie derivatives of superior order can be defined recursively as

$$L_f^0 h = f; \quad L_f^i h = L_f \left(L_f^{i-1} h\right) = \nabla \left(L_f^{i-1} h\right) \cdot f \quad \text{for } i \ge 1.$$

In the same way, if $g(\boldsymbol{x})$ is another vectorial field, the double directional derivative of

h with respect to f and g is given by

$$L_g L_f h = \nabla (L_f h) \cdot g.$$

We search a control input of the form (2.5) such that

$$\begin{bmatrix} y_1^{(r_1)} & y_2^{(r_2)} & y_3^{(r_3)} & y_4^{(r_4)} \end{bmatrix}^T = b(\boldsymbol{x}) + \Delta(\boldsymbol{x})\boldsymbol{u}, \tag{2.10}$$

which is satisfied by taking

$$\Delta(x) = \begin{bmatrix} L_{g_1} L_f^{r_1 - 1} h_1(x) & \cdots & L_{g_4} L_f^{r_1 - 1} h_1(x) \\ \vdots & \ddots & \vdots \\ L_{g_1} L_f^{r_4 - 1} h_4(x) & \cdots & L_{g_4} L_f^{r_4 - 1} h_4(x) \end{bmatrix},$$

and

$$b(x) = \begin{bmatrix} L_f^{r_1} h_1(x) \\ \vdots \\ L_f^{r_4} h_4(x) \end{bmatrix}.$$

Nevertheless, the control law in (2.5) defined by (2.6) and (2.7) can only be established when the matrix $\Delta(x)$ is non-singular.

For the system (2.2) along with output (2.9), the relative degrees are

$$r_1 = r_2 = r_3 = 1, \quad r_4 = 2$$

and

$$\Delta(\boldsymbol{x}) = \begin{bmatrix} \frac{1}{m}\left(\cos(\phi)\cos(\psi)\sin(\theta) + \sin(\phi)\sin(\psi)\right) & 0 & 0 & 0 \\ \frac{1}{m}\left(\cos(\phi)\sin(\psi)\sin(\theta) - \sin(\phi)\cos(\psi)\right) & 0 & 0 & 0 \\ \frac{1}{m}\left(\cos(\phi)\cos(\theta)\right) & 0 & 0 & 0 \\ 0 & 0 & \dfrac{d \ \sin(\phi)}{I_y \ \cos(\theta)} & \dfrac{\cos(\phi)}{I_z \ \cos(\theta)} \end{bmatrix}. \tag{2.11}$$

However, we notice that the matrix $\Delta(\boldsymbol{x})$ in (2.11) is singular for all $\boldsymbol{x}$. Therefore, the control law cannot be computed; this happens because $r_1 + r_2 + r_3 + r_4 = 5 < 12$ that is the order of the system. Hence, there exists zero dynamics (see [Kha02], Chapter 13).

In order to avoid the zero dynamics and the singularity in (2.11), a dynamic extension (see [Sas99], Chapter 9) can be used, such that a double integrator is introduced in u_1

as follows:

$$\begin{aligned} u_1 &= \zeta, \\ \dot{\zeta} &= \epsilon, \\ \dot{\epsilon} &= \bar{u}_1, \end{aligned}$$

and to preserve consistency in the notation, the rest of the inputs are now denoted

$$\begin{aligned} u_2 &= \bar{u}_2, \\ u_3 &= \bar{u}_3, \\ u_4 &= \bar{u}_4. \end{aligned}$$

As we are taking as outputs the linear velocities for this low-level control, and given that the three variables corresponding to the absolute position $(x,\ y,\ z)$ are not involved in the dynamics of the rest of variables, we are going to consider a new system without the three variables relating to position, and with the two new variables relative to the double integrator

$$\begin{aligned} \dot{\bar{\boldsymbol{x}}} &= \bar{f}(\bar{\boldsymbol{x}}) + \bar{g}(\bar{\boldsymbol{x}})\bar{\boldsymbol{u}}, \\ \bar{\boldsymbol{y}} &= \bar{h}(\bar{\boldsymbol{x}}), \end{aligned} \tag{2.12}$$

where

$$\bar{\boldsymbol{x}} = \begin{bmatrix} v_x & v_y & v_z & \phi & \theta & \psi & \zeta & \epsilon & \omega_x & \omega_y & \omega_z \end{bmatrix}^T, \tag{2.13}$$

$$\bar{f}(\bar{\boldsymbol{x}}) = \begin{bmatrix} \frac{1}{m}\left(\cos(\phi)\cos(\psi)\sin(\theta) + \sin(\phi)\sin(\psi)\right) \\ \frac{1}{m}\left(\cos(\phi)\sin(\psi)\sin(\theta) - \sin(\phi)\cos(\psi)\right) \\ -g + \frac{1}{m}\left(\cos(\phi)\cos(\theta)\right) \\ \omega_x + \sin(\phi)\tan(\theta)\omega_y + \cos(\phi)\tan(\theta)\omega_z \\ \cos(\phi)\omega_y - \sin(\phi)\omega_z \\ \sin(\phi)\sec(\theta)\omega_y + \cos(\phi)\sec(\theta)\omega_z \\ \epsilon \\ 0 \\ \frac{I_y - I_z}{I_x}\omega_y\omega_z \\ \frac{I_z - I_x}{I_y}\omega_x\omega_z \\ \frac{I_x - I_y}{I_z}\omega_x\omega_y \end{bmatrix} \tag{2.14}$$

and

$$\bar{g}(\bar{\boldsymbol{x}}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & \dfrac{d}{I_x} & 0 & 0 \\ 0 & 0 & \dfrac{d}{I_y} & 0 \\ 0 & 0 & 0 & \dfrac{1}{I_z} \end{bmatrix} \tag{2.15}$$

and finally,

$$\bar{h}(\bar{\boldsymbol{x}}) = \begin{bmatrix} v_x & v_y & v_z & \psi \end{bmatrix}^T . \tag{2.16}$$

For the new system, the relative degrees $\{r_1,\ r_2,\ r_3,\ r_4\}$ are

$$r_1 = r_2 = r_3 = 3, \quad r_4 = 2.$$

Now we have $r_1 + r_2 + r_3 + r_4 = 11$, which equals the order of the system, so the new system is linearizable by feedback using

$$\Delta(\bar{\boldsymbol{x}}) = \begin{bmatrix} \frac{\cos(\phi)\cos(\psi)\sin(\theta)+\sin(\phi)\sin(\psi)}{m} & -\frac{\zeta(\sin(\phi)\cos(\psi)\sin(\theta)-\cos(\phi)\sin(\psi))d}{mI_x} & \frac{\zeta\cos(\phi)\cos(\theta)d}{mI_y} & 0 \\ \frac{\cos(\phi)\sin(\psi)\sin(\theta)-\sin(\phi)\cos(\psi)}{m} & -\frac{\zeta(\sin(\phi)\sin(\psi)\sin(\theta)+\cos(\phi)\cos(\psi))d}{mI_x} & \frac{\zeta\sin(\psi)\cos(\theta)d}{mI_y} & 0 \\ \frac{\cos(\phi)\cos(\theta)}{m} & -\frac{\zeta\cos(\phi)\cos(\theta)d}{mI_x} & -\frac{\zeta\sin(\theta)d}{mI_y} & 0 \\ 0 & 0 & \frac{\sin(\phi)d}{\cos(\theta)I_y} & \frac{\cos(\phi)}{\cos(\theta)I_z} \end{bmatrix}. \tag{2.17}$$

The matrix $\Delta(\bar{\boldsymbol{x}})$ in (2.17) is non-singular for every $\bar{\boldsymbol{x}}$ where $\zeta \neq 0$ and $-\frac{\pi}{2} < \phi, \theta < \frac{\pi}{2}$. The singularities arise when the propellers stop rotating (i.e. $\zeta = u_1 = 0$), or when the quadrotor tilts up to 90 degrees, reaching a completely vertical position (i.e. $\phi \in \{-\frac{\pi}{2}, \frac{\pi}{2}\}$ or $\theta \in \{-\frac{\pi}{2}, \frac{\pi}{2}\}$). In practice, these situations appear only when the quadrotor needs to do acrobatic maneuvers but this is outside the scope of this work. Given this, the system (2.12) can be transformed by a change of coordinates $\boldsymbol{\lambda} = \Phi(\bar{\boldsymbol{x}})$ given by

$$\begin{array}{llll} \lambda_1 = \bar{h}_1(\bar{\boldsymbol{x}}) = v_x & \lambda_4 = \bar{h}_2(\bar{\boldsymbol{x}}) = v_y & \lambda_7 = \bar{h}_3(\bar{\boldsymbol{x}}) = v_z & \lambda_{10} = \bar{h}_4(\bar{\boldsymbol{x}}) = \psi \\ \lambda_2 = L_{\bar{f}}\bar{h}_1(\bar{\boldsymbol{x}}) = \dot{v}_x & \lambda_5 = L_{\bar{f}}\bar{h}_2(\bar{\boldsymbol{x}}) = \dot{v}_y & \lambda_8 = L_{\bar{f}}\bar{h}_3(\bar{\boldsymbol{x}}) = \dot{v}_z & \lambda_{11} = L_{\bar{f}}\bar{h}_4(\bar{\boldsymbol{x}}) = \dot{\psi} \\ \lambda_3 = L^2_{\bar{f}}\bar{h}_1(\bar{\boldsymbol{x}}) = \ddot{v}_x & \lambda_6 = L^2_{\bar{f}}\bar{h}_2(\bar{\boldsymbol{x}}) = \ddot{v}_y & \lambda_9 = L^2_{\bar{f}}\bar{h}_3(\bar{\boldsymbol{x}}) = \ddot{v}_z & \end{array} . \tag{2.18}$$

In these new coordinates, the system can be rewritten as

$$\begin{aligned} \dot{\boldsymbol{\lambda}} &= A\boldsymbol{\lambda} + B\boldsymbol{\nu} \\ \bar{\boldsymbol{y}} &= C\boldsymbol{\lambda}, \end{aligned} \tag{2.19}$$

in which

$$A = \begin{bmatrix} 0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}, \quad B = \begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 1&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&1&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&1&0 \\ 0&0&0&0 \\ 0&0&0&1 \end{bmatrix},$$

$$C = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0 \end{bmatrix},$$

i.e. the system is now written in a linearized form.

2.3 Control architectures of Multi-agent systems (MAS)

There exists three main strategies for controlling multi-agent systems in general, and multiple quadrotor systems in particular [WG18]. The factors that differentiate each one depend on *where* the controllers computation takes place, and *what* type of information is communicated between agents. Hereafter, we briefly present these strategies.

2.3.1 Centralized control

In the *centralized* control strategy, the computation work by all the controllers takes place in the same station. The states information $\boldsymbol{x}_i(t)$ (for agent i, at time t) is gathered together in one computation station, then the control law is calculated for each agent and communicated back to each of them. The computation station could be any one of the agents or a remote computer. The advantage of this strategy is that all the information is available before computing the control laws.

The Figure 2.3 shows a schema of a centralized control architecture.

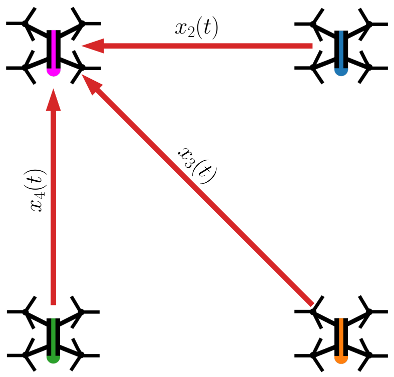


Figure 2.3: Centralized control architecture. The agents share their states $\boldsymbol{x}_i(t)$ with a single agent who is going to compute all the control inputs.

2.3.2 Decentralized control

In the *decentralized* control strategy, the computation of the controllers takes place in a different station for each agent. Only the states information $\boldsymbol{x}_i(t)$ is shared between the agents and, with that data, each agent computes its own control law. Since each agent i only knows the states $\boldsymbol{x}_j(t)$ of its partners j but does not know the control law they have computed, the agents do not know what the others are going to do next. The agents could collect the states information of all their partners or well the states information could be gathered together in one place and then be communicated to each agent.

The Figure 2.4 shows a schema of a decentralized control architecture.

2.3.3 Distributed control

As in the decentralized control strategy, in the distributed control strategy the computation of the control laws takes place in different stations, for each agent. Nevertheless,

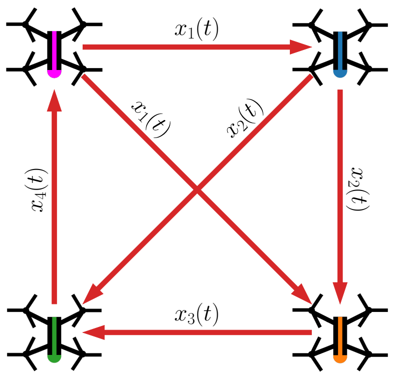


Figure 2.4: Decentralized control architecture. The agents share only their states $\boldsymbol{x}_i(t)$ with their neighbors.

the information shared between the agents consists not only on the states $\boldsymbol{x}_i(t)$ but also on the control inputs $\boldsymbol{u}_i(t)$ computed by the neighbors. In this strategy, the agents need a mean of communication and the information can not be just gathered together in one place and communicated to each agent, as it is done in decentralized control.

The Figure 2.5 shows a schema of a distributed control architecture.

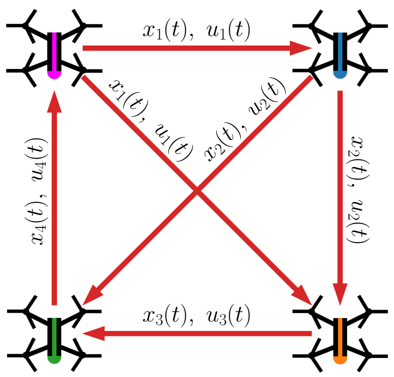


Figure 2.5: Distributed control architecture. The agents share their states $\boldsymbol{x}_i(t)$ and also their computed control inputs $\boldsymbol{u}_i(t)$ with their neighbors.

2.4 Consensus

Given a networked multi-agent system, *consensus* means to reach an agreement state for all the agents. This agreement should be achieved only with the sharing of local information of connected agents. In this section, we are going to describe the main results in consensus theory, and more detailed information can be found in [OSFM07].

Let us consider a system of n agents with state variables $\boldsymbol{x}_i(t)$ at time $t \in \mathbb{R}$ for

$i = 1, \dots, n$ with every agent having m state-variables, that is $\boldsymbol{x}_i \in \mathbb{R}^m$. The aim of a consensus protocol is to define control inputs $\boldsymbol{u}_i(t)$ such that the agreement in the state variables of all agents in the network can be asymptotically accomplished.

2.4.1 Multi-agent networks and graph theory

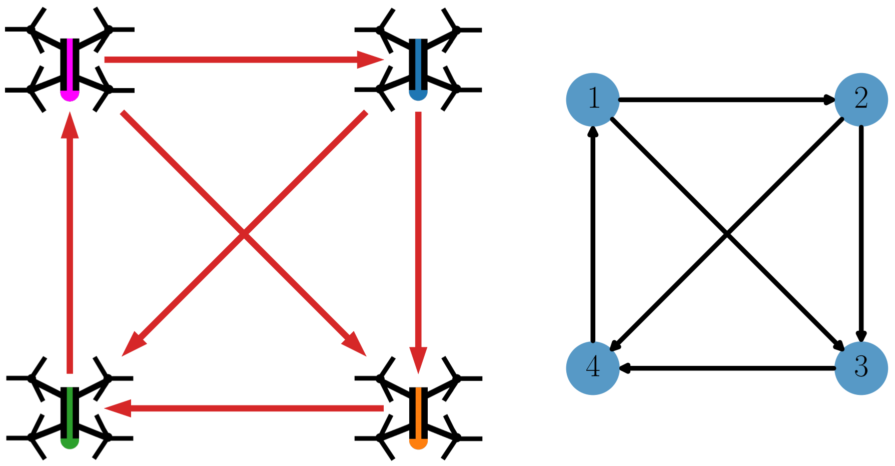


Figure 2.6: Graph induced by communication topology of a 4-agent network.

The interaction topology of a multi-agent network can be mathematically represented by a directed graph $G = (V, E)$ where $V = \{1, 2, \dots, n\}$ is the set of nodes or agents, and where $E \subset V \times V$ is the set of edges of the graph.

An edge (i, j) belongs to E if the agent i is communicated through the network with agent j. The set of neighbors of agent i is denoted by $N_i = \{j \in V | (i, j) \in E\}$.

A directed graph G has a directed *spanning tree* if there is a vertex i such that there is a (directed) path to any other vertex in the graph.

The *Laplacian matrix* of a graph G is defined by

$$L = \left[l_{ij}\right] = \begin{cases} -1 & j \in N_i, \\ |N_i| & j = i. \end{cases}$$

This matrix has the property that all its rows sum 0.

2.4.2 Consensus protocol

Let us consider a multi-agent system where the dynamics of every state-variable of each agent is a simple integrator, resulting on a decoupled dynamics for each agent of the form

$$\dot{\boldsymbol{x}}_i = I_m \boldsymbol{u}_i = \boldsymbol{u}_i \in \mathbb{R}^m \qquad \forall i \in V. \tag{2.20}$$

The aim is to asymptotically reach a configuration of states for the system such that

$$\boldsymbol{x}_1 = \boldsymbol{x}_2 = \cdots = \boldsymbol{x}_n. \tag{2.21}$$

Given the initial conditions of the system $\boldsymbol{x}_i(0)$ for $i \in V$, the consensus protocol is given by the control law

$$\dot{\boldsymbol{x}}_i(t) = \sum_{j \in N_i} (\boldsymbol{x}_j(t) - \boldsymbol{x}_i(t)). \tag{2.22}$$

Equation (2.22) can be expressed in terms of the Laplacian matrix of the graph, giving the following expression

$$\dot{\boldsymbol{X}} = -L\boldsymbol{X}, \tag{2.23}$$

where $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T & \boldsymbol{x}_2^T & \cdots & \boldsymbol{x}_n^T \end{bmatrix}^T$.

2.4.3 Spectral properties

The Laplacian matrix has always a zero eigenvalue $\lambda_1 = 0$ that corresponds to the eigenvector $\mathbb{1} = \begin{bmatrix} 1, 1, \cdots, 1 \end{bmatrix}^T$. This is because all the rows of this matrix sums to 0, that is $\sum_j L_{ij} = 0$ for any i.

This means that one equilibrium of the system is of the form

$$\boldsymbol{X}^* = \begin{bmatrix} \alpha \\ \alpha \\ \vdots \\ \alpha \end{bmatrix} = \alpha \mathbb{1} \tag{2.24}$$

where all the agents states *agree* on the same value α.

In [OSFM07], it is shown that $\boldsymbol{X}^*$ is the only equilibrium (up to a multiplicative factor) for *connected* graphs.

2.4.4 Convergence analysis

In order to achieve consensus, the graph G must have a directed spanning tree. In [OSFM07], it is shown that the final consensus value depends on whether the connectivity graph is directed or undirected. If the graph G is undirected, it can be shown that the final consensus value is equal to the mean of the initial values, that is

$$\boldsymbol{\alpha} = \frac{1}{n} \sum_{i \in V} \boldsymbol{x}_i(0). \tag{2.25}$$

The rate of convergence will depend on the second smallest eigenvalue λ_2 of L, known as *algebraic connectivity.* For dense graphs, λ_2 is larger than for sparse graphs, resulting on a slower consensus for the latter ones [OM04].

If the graph G is directed, the consensus value depends on the eigenvector $\boldsymbol{v}$ of L such that $\boldsymbol{v}^T L = 0$ and the consensus value is

$$\boldsymbol{\alpha} = \frac{\boldsymbol{v}^T \boldsymbol{X}(0)}{\sum\limits_j v_j}, \tag{2.26}$$

which is again an average value of the initial states, but weighted by coefficients $\frac{v_i}{\sum\limits_j v_j}$. In [OM04] the authors extend the notion of *algebraic connectivity* to directed graphs, by the application of a mirror operation that gives as result an undirected graph from the original directed one, obtaining similar conclusions as for the undirected case.

2.4.5 Formation control

Consensus can be used for formation control. Assume that we are given a set of desired configurations of the agents, defined by n displacement vectors $\boldsymbol{d}_i$ with respect to an arbitrary origin, then we will see that *virtual agents* can be defined to achieve consensus between them and, at the same time, to ensure that the formation is reached.

The displacement vector $\boldsymbol{d}_i$ starts from the origin to the desired position of the agent i. Inspired by [JG17], let us consider a virtual agent $\boldsymbol{z}_i$ as

$$\boldsymbol{z}_i = \boldsymbol{x}_i - \boldsymbol{d}_i. \tag{2.27}$$

Because vectors $\boldsymbol{d}_i$ are constant, virtual agents defined by (2.27) have the same dynamics as the real agents as expressed in (2.28), so the results presented along this section apply also to virtual agents. We have

$$\dot{\boldsymbol{z}}_i = \dot{\boldsymbol{x}}_i = \boldsymbol{u}_i. \tag{2.28}$$

Then, in order to achieve consensus between virtual agents, the control law should be

$$\boldsymbol{u}_i(t) = \sum_{j \in N_i} (\boldsymbol{z}_j(t) - \boldsymbol{z}_i(t)). \tag{2.29}$$

Considering $\boldsymbol{\alpha}_z$ as the consensus value reached by the virtual agents, then the final state of real agents will satisfy

$$\boldsymbol{x}_i - \boldsymbol{d}_i = \boldsymbol{\alpha}_z.$$

Moreover $\boldsymbol{x}_i = \boldsymbol{d}_i + \boldsymbol{\alpha}_z$, and this means that when consensus is achieved for the virtual agents $\boldsymbol{z}_i$, the configuration between the real agents $\boldsymbol{x}_i$ would be the same as the one defined by vectors $\boldsymbol{d}_i$, up to an overall displacement given by the consensus final value $\boldsymbol{\alpha}_z$.

2.4.6 Discretization

Up to this moment we have considered continuous-time expressions, nevertheless in practice the signals received and sent to robots are digital signals, i.e. discrete signals sampled every T seconds. Given an initial time $t_0 \in \mathbb{R}$, the time can be discretized as

$$t_k := t_0 + kT,$$

where $k \in \mathbb{N}$, resulting on the sequence $\{t_0,\ t_1,\ t_2, \ldots, t_k, \ldots\}$ that corresponds to the instants of time where we receive and send signals to the robot.

From now on, abusing of notation, we will refer as $\cdot(k)$ instead of $\cdot(t_k)$ for the discrete time expressions.

2.5 Model Predictive Control

Model Predictive Control (MPC) is a control strategy that takes advantage of the fact that there may exist a reasonably good enough predictive model of the system that we want to control. Given that predictive model, this control strategy aims to predict the behavior of the system within a time window and evaluate the effect of this behavior through a user-defined cost function.

In this section, we are going to describe briefly the fundamentals of Model Predictive Control, but the reader looking for more details can consult sources such as [CSdlPL12], [Lee11] and [CBA99].

Figure 2.7 shows a block diagram of a general MPC when a reference $\boldsymbol{r}$ is given for the states $\boldsymbol{x}$. In order to choose the control input, a cost function $\Phi(k)$ is minimized at each iteration k; this cost function is typically expressed in terms of the control input and of some other parameters that the specific problem needs to take into account. Using the model of the system, the states of the system can be predicted ($\hat{\boldsymbol{x}}$) along a predefined horizon. It is important to remark that the cost function $\Phi(k)$ considers these predictions. Then, once $\Phi(k)$ is optimized, we will obtain inputs for all the iterations along the prediction horizon, but only the control input of the present iteration $\boldsymbol{u}(k)$ will be applied to the system.

One of the main advantages of Model Predictive Control is that, given the main optimization problem, it also permits to integrate constraints according to the system specific characteristics such as bounds on the inputs or the states, and obstacle avoidance.

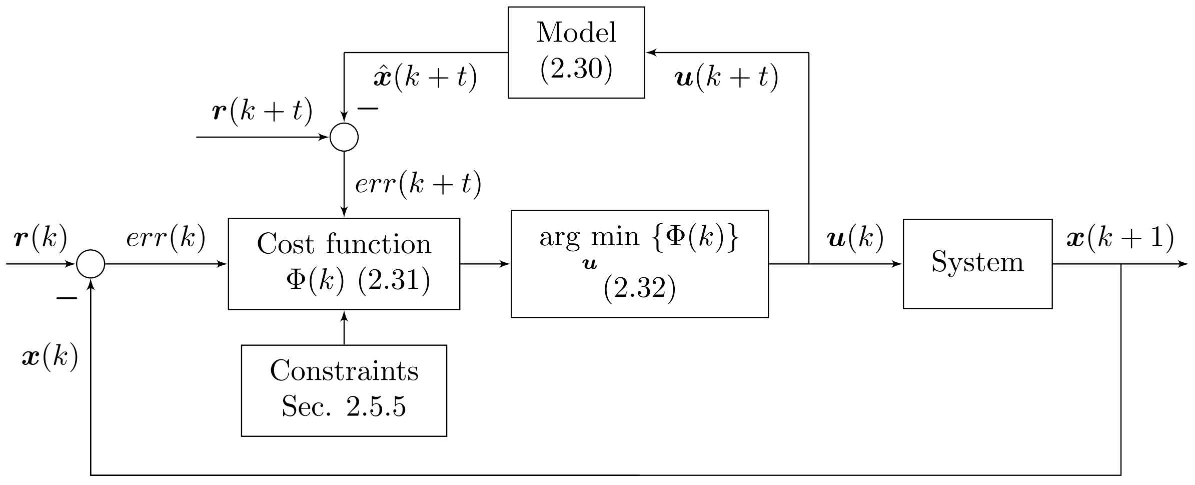


Figure 2.7: Model Predictive Control block diagram.

2.5.1 Model of the system

We are going to consider a discrete time linear model of the system. This model is defined in (2.30) and is represented by the "Model" block in Figure 2.7.

$$\boldsymbol{x}(k+1) = A\boldsymbol{x}(k) + B\boldsymbol{u}(k), \tag{2.30}$$

where $\boldsymbol{x}(k) = \begin{bmatrix} x_1(k) & x_2(k) & \cdots & x_m(k) \end{bmatrix}^T$ is the vector of m state variables at iteration k and $\boldsymbol{u}(k) = \begin{bmatrix} u_1(k) & u_2(k) & \cdots & u_l(k) \end{bmatrix}^T$ is the vector of l control inputs applied to the system at iteration k. The matrices $A \in \mathbb{R}^{m\times m}$ and $B \in \mathbb{R}^{m\times l}$ completely describe the model.

2.5.2 Cost function

We are going to consider the generic cost function for Model Predictive Control presented in [CFZ15] that considers a given reference $\mathbf{r}$ for the states and a term for the waste of energy from the inputs. This cost function penalizes the error between the actual and predicted states from the reference, and the amount of the inputs. The equation (2.31) shows this cost function and it corresponds to the "Cost function" block in Figure 2.7.

$$\Phi(k) = \beta \sum_{t=H_w}^{H_p} ||\boldsymbol{x}(k+t|k) - \boldsymbol{r}(k+t|k)||^2 + \gamma \sum_{t=1}^{H_u} ||\boldsymbol{u}(k+t-1|k)||^2. \tag{2.31}$$

The notation $a(m|n)$ in use in the equation above refers to the predicted value of a at the iteration m such that the value of a is known at iteration $n < m$. This prediction is obtained from the model (2.30) of the system.

The value H_p corresponds to the prediction horizon, which is the length of the window of predicted values for the state variables. The value H_u is called the control horizon and corresponds to the number of steps of control to be taken into account. The value of H_u should be less or equal to the prediction horizon H_p.

Finally, the number H_w is the window parameter, and corresponds to the first iteration in the prediction horizon to be penalized. This is used because we usually do not want to start penalizing the error from the reference since the actual state. H_w should be superior or equal to 1, and inferior to the prediction horizon H_p.

2.5.3 Optimization problem

At each iteration of Model Predictive Control, the following optimization problem is solved

$$\boldsymbol{u}^* = \arg\min_{\boldsymbol{u}} \Phi(k). \tag{2.32}$$

Solving the optimization problem (2.32) gives as a result an optimal input sequence of future controls

$$\{\boldsymbol{u}^*(k|k), \boldsymbol{u}^*(k+1|k), \ldots, \boldsymbol{u}^*(k+H_u-1|k)\} \tag{2.33}$$

associated to a minimum cost $\Phi^*(k)$.

The input actually applied to the controlled system, at each iteration, corresponds to the *first* term of the sequence (2.33), that is $\boldsymbol{u}^*(k|k)$, discarding the rest of the sequence. Some authors, such as [HT12], use the information of the rest of the sequence as a warm start for the optimization problem at the next iteration.

2.5.4 Formulation as a quadratic programming problem

For general dynamic systems, the optimization problem (2.32) is hard to solve and it is not evident that a solution exists. In order to rewrite (2.32) as a quadratic programming problem, that guarantees that a global minimum exists, the following vectors are defined

$$\boldsymbol{X}(k+1) := \begin{bmatrix} \boldsymbol{x}(k+1|k) \\ \boldsymbol{x}(k+2|k) \\ \vdots \\ \boldsymbol{x}(k+H_p|k) \end{bmatrix} \quad \boldsymbol{R}(k+1) := \begin{bmatrix} \boldsymbol{r}(k+1|k) \\ \boldsymbol{r}(k+2|k) \\ \vdots \\ \boldsymbol{r}(k+H_p|k) \end{bmatrix} \quad \boldsymbol{U}(k) := \begin{bmatrix} \boldsymbol{u}(k|k) \\ \boldsymbol{u}(k+1|k) \\ \vdots \\ \boldsymbol{u}(k+H_u-1|k) \end{bmatrix}, \tag{2.34}$$

where the different numerical values over the horizon are stacked together. Then, having that $H_w = 1$, the objective function (2.31) can be rewritten as

$$\Phi(k) = (\boldsymbol{X}(k+1) - \boldsymbol{R}(k+1))^T Q(\boldsymbol{X}(k+1) - \boldsymbol{R}(k+1)) + \boldsymbol{U}(k)^T C \boldsymbol{U}(k), \tag{2.35}$$

where the matrices Q and C hold weights for the different terms of the objective function,

$$Q := \begin{bmatrix} \beta & 0 & \cdots & 0 \\ 0 & \beta & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta \end{bmatrix} \quad C := \begin{bmatrix} \gamma & 0 & \cdots & 0 \\ 0 & \gamma & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \gamma \end{bmatrix}.$$

As seen above, we have used uniform weights along the horizon window, but the weights could be defined in a different manner (penalizing differently according to the position in the horizon window).

Given the model of the system defined in (2.30), it can be shown that the model can be extended along the prediction horizon as

$$\boldsymbol{X}(k+1) = P_X \boldsymbol{x}(k) + P_U \boldsymbol{U}(k), \tag{2.36}$$

where

$$P_X := \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^{H_p} \end{bmatrix}, \tag{2.37}$$

$$P_U := \begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{H_u-1}B & A^{H_u-2}B & \dots & B \\ \vdots & \vdots & & \vdots \\ A^{H_p-1}B & A^{H_p-2}B & \cdots & \sum\limits_{i=0}^{H_p-H_u} A^i B \end{bmatrix}. \tag{2.38}$$

Given those expressions, the objective cost function (2.35) can be rewritten as

$$\Phi(k) = \frac{1}{2}\boldsymbol{U}(k)^T H(k)\boldsymbol{U}(k) + \boldsymbol{f}(k)^T\boldsymbol{U}(k) + \boldsymbol{g}(k), \tag{2.39}$$

where

$$H(k) := 2\big(P_U^T Q P_U + C\big), \tag{2.40}$$

$$\boldsymbol{f}(k) := 2P_U^T Q\big(P_X\boldsymbol{x}(k) - \boldsymbol{R}(k+1)\big), \tag{2.41}$$

$$\boldsymbol{g}(k) := \big(P_X\boldsymbol{x}(k) - \boldsymbol{R}(k+1)\big)^T Q\big(P_X\boldsymbol{x}(k) - \boldsymbol{R}(k+1)\big). \tag{2.42}$$

The matrix (2.40) is positive definite and describes the quadratic part of the objective function (2.39), and the vector (2.41) describes the linear part. The last term (2.42) is independent of $\boldsymbol{U}$ so it has no influence in the determination of the optimum $\boldsymbol{U}^*$.

The cost function (2.39) is written as a quadratic form in $\boldsymbol{U}$, and this ensures that solving optimization problem (2.32) is possible and that the optimum is unique, because $H(k)$ is positive definite.

There exists a lot of solvers for this type of optimization problems, that are fast and reliable. This is important given the fact that the system we are using, a quadrotor, requires a fast computation of the control inputs.

Reference as a linear function of $\boldsymbol{U}$

If the reference $\boldsymbol{r}(k+t|k)$ for the state variables depends linearly on the predicted states $\boldsymbol{x}(k+t|k)$, a vector $\boldsymbol{A}(k) \in \mathbb{R}^{mH_p}$ and a matrix $B(k) \in \mathbb{R}^{(mH_p)\times(mH_u)}$ can be found in order to rewrite the reference vector $\boldsymbol{R}(k+1)$ along the prediction horizon as a linear function of the input vector $\boldsymbol{U}(k)$ along the control horizon as follows

$$\boldsymbol{R}(k+1) = \boldsymbol{A}(k) + B(k)\boldsymbol{U}(k). \tag{2.43}$$

Given this and the extended model (2.36), the matrix $H(k)$ and vectors $\boldsymbol{f}(k)$ and $\boldsymbol{g}(k)$ of the quadratic form of the cost function (2.39) will be defined as

$$H(k) := 2\left[\big(P_U - B(k)\big)^T Q \big(P_U - B(k)\big) + C\right], \tag{2.44}$$

$$\boldsymbol{f}(k) := 2\big(P_U - B(k)\big)^T Q \big(P_X \boldsymbol{x}(k) - \boldsymbol{A}(k)\big), \tag{2.45}$$

$$\boldsymbol{g}(k) := \big(P_X \boldsymbol{x}(k) - \boldsymbol{A}(k)\big)^T Q \big(P_X \boldsymbol{x}(k) - \boldsymbol{A}(k)\big). \tag{2.46}$$

2.5.5 Common constraints

Model Predictive Control is very flexible when it is necessary to take constraints into account as they just need to be attached to the optimization problem, as shown in the “Constraints” block of Figure 2.7. Some of the most used ones are the ones related to the physical limitations in the control input signals, such as bounds in the values that can be applied to the system. These constraints can be expressed as

$$\boldsymbol{u}_{\min}(k) \leq \boldsymbol{u}(k) \leq \boldsymbol{u}_{\max}(k), \tag{2.47}$$

then the optimization problem (2.32) becomes

$$\begin{aligned} \boldsymbol{u}^* = \quad & \arg\min_{\boldsymbol{u}} \Phi(k), \\ & \text{subject to} \quad \mathcal{L}\boldsymbol{u} \leq \ell \end{aligned} \tag{2.48}$$

where

$$\mathcal{L} = \begin{bmatrix} I_m \\ -I_m \end{bmatrix} \quad \ell = \begin{bmatrix} \boldsymbol{u}_{\max}(k) \\ -\boldsymbol{u}_{\min}(k) \end{bmatrix}. \tag{2.49}$$

Extending this to the quadratic form in (2.39), the previous constraints can take the form

$$\boldsymbol{U}(k) \leq \boldsymbol{U}_{\max}(k) \tag{2.50}$$

$$-\boldsymbol{U}(k) \leq -\boldsymbol{U}_{\min}(k) \tag{2.51}$$

where

$$\boldsymbol{U}_{\min}(k) := \begin{bmatrix} \boldsymbol{u}_{\min}(k) \\ \boldsymbol{u}_{\min}(k+1) \\ \vdots \\ \boldsymbol{u}_{\min}(k+H_u-1) \end{bmatrix}, \quad \boldsymbol{U}_{\max}(k) := \begin{bmatrix} \boldsymbol{u}_{\max}(k) \\ \boldsymbol{u}_{\max}(k+1) \\ \vdots \\ \boldsymbol{u}_{\max}(k+H_u-1) \end{bmatrix}.$$

Chapter 3

State of the art

In this chapter, we briefly review some state-of-the-art works in the topics explored in this thesis.

3.1 Solving the MPC optimization problem

In [CBA99] Model Predictive Control is fully reviewed, from the main elements of every MPC strategy, to several industrial applications examples. This book explains in detail the MPC formulation, its analytic solution when constraints are not considered, along with some numerical procedures that are good when we need to consider restrictions in the output or predicted values. This reference also includes a full chapter regarding robust MPC when uncertainties and inaccuracies arise.

Being computation time a common disadvantage of the implementations of Model Predictive Control, in [HT12] a method based on constraint reduction with exact penalization is proposed for reducing the computation time of the optimization problem solved at each iteration of Model Predictive Control. The article proposes a primal-dual interior point method that takes advantage of the redundancy of some of the constraints of the optimization problem. Also, the constraints are separated into hard and soft, and latter violations can be traded off, permitting the use of infeasible warm starts for the optimization problem.

3.2 MPC and quadrotors

In [CW13], a Model Predictive Control for quadrotors is presented for position control problems. The non-linear dynamics of the quadrotor is linearized around hover state to obtain a linear model to be used in the MPC; this model is also decoupled into a rotational and a positional system. Due to the fact that the linear model performs well only in a vicinity of the hover conditions, the authors propose a cascade structure where first, having as input the desired position, an MPC control generates the total thrust and pitch and roll angles, constrained to be near the hover configuration, then a second MPC uses the angles as inputs to generate the rotational torques, determining the 4 inputs needed to achieve the desired position. The paper presents simulations that support the effectiveness of the proposed cascaded control.

A comparison between two state-of-the-art approaches to Linear and Nonlinear Model Predictive Control for quadrotors is held in [KBS16]. The two methods performance is evaluated in three scenarios: hovering state, step response and aggressive trajectory tracking. They adopt a cascade control scheme for the two inner controllers: an attitude controller and a model based trajectory tracking controller; this scheme permits to run control algorithms on separate and adequate hardware. The two approaches show comparable behavior, with the nonlinear formulation performing slightly better on disturbance rejection.

3.3 MPC and consensus

A review on the recent results in the design of distributed Model Predictive Control is presented in [CSdlPL12]. The article presents several schemes, among which the cooperative distributed Model Predictive Control, where the same global cost function is optimized in each of the local controllers, and where each of them takes into account the effects of its inputs on the entire plant. Distributed Model Predictive Control is also reviewed, where the controllers do not know the dynamics of their neighbors. They propose a solution in order to reach an agreement.

As we have seen it in Chapter 2, consensus theory is very useful for formation control. In [CFZ15] and [DLL17], we can see two MPC-based approaches for consensus of first-order multi-agent systems. The first one presents a decentralized MPC-based consensus protocol for the case where the topology of communication between the agents switches every T seconds. They give some conditions over the connectivity graphs to ensure convergence of the agents. Depending on the characteristics of the connectivity graphs, they propose a control protocol that can ensure convergence, in some cases, even for an arbitrary value of the sampling period T. Also, in this paper, a *window* parameter

H_w is introduced to consider the time-delay between applying a control and viewing its effect into the system. The reference for the MPC considered in this work consists on the local consensus error at each iteration.

In [DLL17], a distributed, MPC-based control for consensus among the same type of agents is proposed. The control protocol depends on the actual consensus error and on the predicted quantities along the prediction horizon. This protocol permits to use a predefined consensus value, rather than the one given by the average of the initial states of the agents. Also, the rate of convergence is shown to be greater than the one of the classical consensus protocol. Finally, a particular formulation of the exact solution of the optimization problem is given, permitting a low computational cost at each iteration.

A distributed consensus and Model Predictive Control for formation of quadrotors is formulated in [CS18]. The formation control problem is decoupled into horizontal and vertical motions and the optimization problem considered in the Model Predictive control is different for each motion. The leader of the formation tracks the desired rate for each motion and generates the desired formation trajectories for the followers which follow these by solving each one a Model Predictive Control optimization problem. The studied movements where straight flight, turning around and following an helicoidal path. The results where presented only as simulations.

A formation control of a fleet of quadrotors with Model Predictive Control is presented in [DS18]. The article explores centralized and decentralized control architectures and studies the performance of formation control in two scenarios: a static reference and a dynamic one for tracking. All of this is conceived in a motion capture environment so the position of every quadrotor is known through all the experiment. The C/GMRES method to solve the Model Predictive Control optimization problem is proved to be fast enough to assemble and maintain a formation following a static reference for both architectures.

3.4 Formation control and collision avoidance

In the terms of formation control of quadrotors, [YSN17] propose a method that considers collision avoidance using Model Predictive Control, including a term of consensus. Collision avoidance between the agents is considered by adding a term of repulsive force to the cost function of Model Predictive Control. The difficulty arises from the nonlinearity of the repulsive function and makes that it should be approximated as a linear function, in order to achieve real-time control. Also, the model is assumed as a linear system. The method is verified via experiments with three quadcopters.

On the other hand, a nonlinear Model Predictive formation Control is presented in [RSCSC15]. The controllers are distributed to each quadcopter and coupling is done through its objective function. The kinematic model of the quadrotor is formulated relative to a the Serret-Frenet frame moving along a reference curved path. The Serret-Frenet frame is specially useful for curved trajectories, and some details about this frame and an application to path following of quadrotors can be found in [KG17]. In [RSCSC15], the coordination problem is solved through adjustments in the robot's speeds, combined with the path update rate for each robot. Since the Model Predictive Control optimization problem is not always feasible due to measurement errors, some of the constraints are relaxed in order to obtain an optimum. The results are presented in a realistic simulation scenario.

A cooperative formation control strategy is proposed in [KN15], with collision-avoidance capability, for a quadrotor system using decentralized Model Predictive Control and consensus-based control. The dynamic model of a quadrotor is linearized and a multi quadrotor system modeled as a graph. For each quadcopter, a Model Predictive Control optimization problem involving formation and collision avoidance is solved, but the coupled constraints related with collision avoidance permit consistency for the whole quadrotor system.

A collision avoidance control scheme between multiple quadrotor systems is formulated in [IN17]. It uses distributed Model Predictive Control. Two quadrotor teams must follow a trajectory while keeping a defined formation, but at some point the trajectories cross. The collision risk is considered in the cost function as a penalty term and a Kalman filter is used to estimate the positions and the predicted future positions of an obstacle. A proof of the stability of the formation control is presented. However, the results are only shown in a simulation environment.

As we can see, a lot of theoretical work has been done in both consensus with MPC and formation control of quadrotors with MPC, but there are almost none that includes realistic simulations nor experiments with real quadrotors. In this work, we aim to formulate a MPC based formation control algorithm but also test its performance in both realistic simulations and real experiments. Also, we want to introduce the maintenance of connectivity, along obstacle avoidance, as part of the formation control task.

Chapter 4

Model Predictive Control for formation control of quadrotors

Assumptions

We are going to consider a network of n agents described by a directed graph G which has a spanning tree (i.e., the graph is connected). Each agent i has m state-variables described as a vector $\boldsymbol{x}_i \in \mathbb{R}^m$. A desired formation configuration for the whole set of agents is given by a set of n displacement vectors $\boldsymbol{d}_i \in \mathbb{R}^m$ (see Chapter 2.4.5).

All the agents are assumed to have the same dynamics, where the model of the dynamics for each state-variable is the one of a simple integrator (2.20). Considering a constant sampling period T, the discrete model of the dynamics for each agent i is given by

$$\boldsymbol{x}_i(k+1) = \boldsymbol{x}_i(k) + T\boldsymbol{u}_i(k). \tag{4.1}$$

where $\boldsymbol{u}_i(k) \in \mathbb{R}^m$ is the vector of control inputs. In this chapter, we will present two main strategies for the formation control of n quadrotors, modeled at kinematic level as in (4.1), given a desired configuration. In section 4.1, a control strategy in two steps is presented. The first step defines the reference trajectories for each quadrotor and the second one corresponds to a decentralized MPC-based tracking control for these reference trajectories. In section 4.2, we present a single-step, decentralized MPC-based control, using as a reference the consensus error at each iteration. In section 4.3, corresponding distributed architecture considerations for the previous strategies are presented. Finally, in section 4.4, we discuss the main advantages and disadvantages of each strategy and present some simulations to support the selection of the best strategy to implement.

Some basic `Python` simulations associated with obstacle avoidance and connectivity maintenance are introduced in this chapter, while more detailed simulations made within the dynamic simulator `Gazebo` are presented in Chapter 5, along with experiments with real quadrotors.

4.1 Decentralized architecture with a predefined consensus path

Given the classical consensus protocol defined in (2.22), the trajectories for all the agents can be pre-defined in order to achieve consensus. Even though these trajectories are not necessarily collision-free, they can be used as a reference for the Model Predictive Control algorithm presented in (2.31). As we will see, some constraints need to be added in order to avoid collisions.

Following this idea, in this section, we are going to introduce a two-step control: firstly, consensus trajectories for virtual agents are generated; secondly, the generated trajectories are followed using decentralized Model Predictive Control, while adding some constraints to avoid collisions.

4.1.1 Consensus-driven trajectories

In order to reach the desired configuration defined by the vectors $\boldsymbol{d}_i$, virtual agents $\boldsymbol{z}_i$ are defined as in (2.27). Using Euler integration with sampling period T and the consensus protocol defined in (2.29), discrete reference trajectories $\boldsymbol{r}_i(k)$ for each virtual agent $\boldsymbol{z}_i$ can be defined as follows.

To characterize the initial conditions of the system, we define $\boldsymbol{r}_i(0) = \boldsymbol{z}_i(0) = \boldsymbol{x}_i(0) - \boldsymbol{d}_i$ for $i = 1, \dots, n$, and the reference path for each virtual agent is described by

$$\boldsymbol{r}_i(k+1) = \boldsymbol{r}_i(k) + T \sum_{j \in N_i} (\boldsymbol{r}_j(k) - \boldsymbol{r}_i(k)). \tag{4.2}$$

These reference paths $\boldsymbol{r}_i$ are updated at least until consensus is achieved and, consequently, until the real agents reach a formation. If we would like to find the reference trajectories for a real agent $\boldsymbol{x}_i$, we should add $\boldsymbol{d}_i$ to $\boldsymbol{r}_i(k)$ for all the iterations k:

$$\boldsymbol{x}_i(k) = \boldsymbol{r}_i(k) + \boldsymbol{d}_i.$$

4.1.2 Cost function for the Model Predictive Control

We are considering a decentralized control architecture (see Section 2.3), so the control inputs for each agent are computed separately and are not sent to the other agents. Given (2.28), the optimization problem can be expressed in terms of the virtual agents $\boldsymbol{z}_i$ and the control inputs. Then, the optimal control inputs $\boldsymbol{u}_i^*$ can be directly applied to real agents $\boldsymbol{x}_i$.

Each agent will minimize an individual cost function of the form (2.31) taking as reference the path generated by (4.2). With the parameters defined in Section 2.5.2, the cost function for the agent i is defined by

$$\Phi_i(k) = \beta_i \sum_{t=H_w}^{H_p} ||\boldsymbol{z}_i(k+t|k) - \boldsymbol{r}_i(k+t)||^2 + \gamma_i \sum_{t=1}^{H_u} ||\boldsymbol{u}_i(k+t-1|k)||^2. \tag{4.3}$$

The first term penalizes deviations from the reference trajectory induced by the consensus control (Eq. (4.2)), while the second term penalizes high values of the control inputs.

4.1.3 Obstacle avoidance

The mere minimization of (4.3) will lead to achieve a formation with the same characteristics of the one defined by vectors $\boldsymbol{d}_i$, but there is no guarantee about the absence of collisions between real agents. Given this, it is mandatory to consider constraints in the optimization problem to ensure that the real agents do not collide. Even more, we will ensure that they do not approach each other more than a security distance $D > 0$.

We can consider two types of constraints: *hard constraints*, in the form of equalities or inequalities in the optimization problem, and *soft constraints*, in the form of a penalty term to be added to the cost function (4.3).

Given that we are considering a decentralized control architecture, an agent will only receive the information about the *positions* of its neighbors, so it would be able to avoid only the agents it is communicated with. We will also apply the following evasion strategies to the collision avoidance with respect to fixed obstacles that can be in the environment; we suppose either that each agent knows in advance the position of the fixed obstacles or that it can measure its relative distance to the obstacles.

Hard constraints

Given an agent i, hard constraints for collision avoidance along the prediction horizon are expressed in the form

$$||\boldsymbol{x}_i(k+t|k) - \boldsymbol{x}_j(k)||^2 \geq D^2, \tag{4.4}$$

with $t = 1, \ldots, H_p$ and $j \in N_i$. As the agent i only knows the position of its neighbors at iteration k and does not know in what direction they are going to move, the neighboring agents are considered as *static* obstacles along the prediction horizon.

There will be H_p constraints of the type of (4.4) for each neighbor j of agent i, adding in total $H_p|N_i|$ inequality constraints to the optimization problem. These constraints ensure that, along the prediction horizon, the distance between the agent i and the actual position of its neighbors, is at least D for all the predicted positions of agent i.

Considering that all the predicted positions along the prediction horizon H_p are actually close from each other, it is reasonable to suggest that most of the $H_p|N_i|$ constraints of the form (4.4) are very similar. Depending on the connectivity of the graph G and on the size of the prediction horizon H_p, several constraints could be needed, making the optimization problem computationally harder to solve. On the other hand, some configurations of the system, along with the perturbations associated with the system, can lead to inconsistent constraints, making the optimization problem impossible to solve.

Hence, as a lighter and more flexible alternative, let us consider the average distance between the agent i and a neighbor $j \in N_i$ along the prediction horizon. This average value is given by

$$d_{ij}^2 = \frac{||\boldsymbol{X}_i(k+1) - \boldsymbol{Y}_j(k)||^2}{H_p}, \tag{4.5}$$

where

$$\boldsymbol{X}_i(k+1) := \begin{bmatrix} \boldsymbol{x}_i(k+1|k) \\ \boldsymbol{x}_i(k+2|k) \\ \vdots \\ \boldsymbol{x}_i(k+H_p|k) \end{bmatrix}, \quad \boldsymbol{Y}_j(k) := \begin{bmatrix} \boldsymbol{x}_j(k) \\ \boldsymbol{x}_j(k) \\ \vdots \\ \boldsymbol{x}_j(k) \end{bmatrix}.$$

Then, we consider just one constraint for each neighbor, given by

$$d_{ij}^2 \geq D^2. \tag{4.6}$$

There will be $|N_i|$ constraints of type (4.6) added to the optimization problem. This option to manage the collision avoidance is computationally cheaper; however, this type of constraint does not ensure that all the predicted positions are away from the neighbors by more than D, as only the average is.

We can rewrite the average distance (4.5) explicitly in terms of the vector $\boldsymbol{U}_i(k)$, that stacks the future controls, as defined in (2.34), using (2.36)

$$\begin{aligned} d_{ij}^2\left(\boldsymbol{U}_i(k)\right) &= \frac{\left|\left|P_x\boldsymbol{x}_i(k) + P_U\boldsymbol{U}_i(k) - \mathbf{Y}_j(k)\right|\right|^2}{H_p} \qquad (4.7) \\ &= \frac{1}{H_p}\Big[\boldsymbol{U}_i(k)^T P_U^T P_U \boldsymbol{U}_i(k) \\ &\qquad + 2\left(P_X\boldsymbol{x}_i(k) - \mathbf{Y}_j(k)\right)^T P_U\boldsymbol{U}_i(k) \qquad (4.8) \\ &\qquad + \left|\left|P_X\boldsymbol{x}_i(k) - \mathbf{Y}_j(k)\right|\right|^2\Big]. \end{aligned}$$

We solve the optimization problem as a quadratic programming problem for the vector $\boldsymbol{U}_i(k)$, and constraints in this type of formulation should be linear in $\boldsymbol{U}_i(k)$. Nevertheless, as we can see in (4.8), $d_{ij}^2(\boldsymbol{U}_i(k))$ is non linear in $\boldsymbol{U}_i(k)$ (it is quadratic instead). Hence, we can linearize $d_{ij}^2(\boldsymbol{U}_i)$ around $\boldsymbol{U}_i = \mathbf{0}$ in order to get a linear approximation that we can use in the formulation of the constraints. Let $\bar{d}_{ij}^2$ be this linear approximation defined as

$$\begin{aligned} \bar{d}_{ij}^2\left(\boldsymbol{U}_i(k)\right) &= d_{ij}^2(\mathbf{0}) + \nabla_{\boldsymbol{U}_i} d_{ij}^2(\mathbf{0}) \cdot \left(\boldsymbol{U}_i(k) - \mathbf{0}\right) \\ &= \frac{\left|\left|P_X\boldsymbol{x}_i(k) - \mathbf{Y}_j(k)\right|\right|^2}{H_p} + \frac{2}{H_p}\left(P_X\boldsymbol{x}_i(k) - \mathbf{Y}_j(k)\right)^T P_U\boldsymbol{U}_i(k). \qquad (4.9) \end{aligned}$$

Then, constraint (4.6) is substituted by

$$\bar{d}_{ij}^2 \geq D^2. \qquad (4.10)$$

Figure 4.1 shows the levels of the squared distance linear approximation $\bar{d}_{ij}^2$ for an agent $\boldsymbol{x}_i$ and an obstacle $\boldsymbol{x}_j$. The red line corresponds to the level $\bar{d}_{ij}^2 = D^2$, when the constraint (4.10) becomes active; therefore, the control is forced to stay at the left side of the red line. As we can see, the levels are perpendicular to the line between the agent $\boldsymbol{x}_i$ and the obstacle $\boldsymbol{x}_j$; in particular, the level $\bar{d}_{ij}^2 = D^2$ should be tangent to the security circumference in green dashed line, but the linearization error does not make this possible. Nevertheless, the red line and the security circumference stay close to each other.

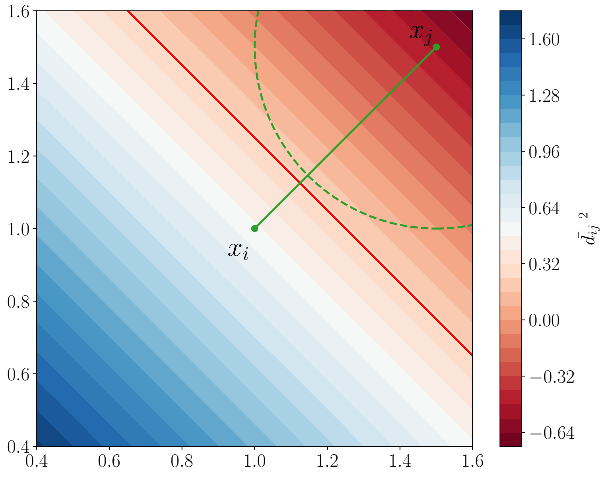


Figure 4.1: Shape of the linearized distance $\bar{d}_{ij}^2$ for an agent $\boldsymbol{x}_i$ and an obstacle $\boldsymbol{x}_j$.

Soft constraints

In [KN15], the authors propose to add a penalty term to the cost function to be optimized in order to avoid collisions; they claim that, even though adding this term does not guarantee the absence of collisions, it is enough in most of practical situations.

This penalty term is of the form of a potential field and is given by

$$\rho_{ij} = \frac{k_d}{(d_{ij} - D)^2}, \tag{4.11}$$

where $k_d > 0$ is a constant associated with the smoothness of evasion and d_{ij} is the average distance defined in (4.5). This penalty term in (4.11) becomes smaller as the agents are farther, and increases as they approach one to the other.

This penalty term is clearly non linear nor quadratic in $\boldsymbol{U}_i(k)$, hence, as we did before, we propose an approximation of ρ_{ij} that we can use for the quadratic programming formulation of the optimization problem. Let $\bar{\rho}_{ij}$ be a linear approximation around $\boldsymbol{U}_i = \boldsymbol{0}$ of the penalty term defined as

$$\begin{aligned}\bar{\rho}_{ij}\left(\boldsymbol{U}_i(k)\right) &= \rho_{ij}(\boldsymbol{0}) + \nabla_{\boldsymbol{U}_i}\rho_{ij}(\boldsymbol{0}) \cdot (\boldsymbol{U}_i(k) - \boldsymbol{0}) \\ &= \frac{k_d}{(d_{ij}(\boldsymbol{0}) - D)^2} - \frac{k_d}{(d_{ij}(\boldsymbol{0}) - D)^3}\frac{1}{d_{ij}(\boldsymbol{0})}\nabla_{\boldsymbol{U}_i} d_{ij}^2(\boldsymbol{0}) \cdot \boldsymbol{U}_i(k) \\ &= \frac{k_d}{(d_{ij}(\boldsymbol{0}) - D)^2} - \frac{2k_d}{H_p d_{ij}(\boldsymbol{0})\left(d_{ij}(\boldsymbol{0}) - D\right)^3}\left(P_X\boldsymbol{x}_i(k) - \boldsymbol{Y}_j(k)\right)^T P_U \boldsymbol{U}_i(k).\end{aligned} \tag{4.12}$$

The linear approximation $\bar{\rho}_{ij}$ has an independent term and adding it to the cost function will not affect the optimum $\boldsymbol{U}_i^*(k)$, so we only need to add to the cost function the linear penalty term

$$-\,k_\rho \left(P_X \boldsymbol{x}_i(k) - \mathbf{Y}_j(k)\right)^T P_U \boldsymbol{U}_i(k), \tag{4.13}$$

where $k_\rho = \dfrac{2k_d}{H_p d_{ij}(\mathbf{0}) \left(d_{ij}(\mathbf{0}) - D\right)^3}$.

We can also consider the quadratic term of the Taylor approximation of $\rho_{ij}(\boldsymbol{U}_i(k))$ around $\boldsymbol{U}_i = \mathbf{0}$ by adding to the matrix $H(k)$ of the quadratic form, the following matrix

$$\begin{split} k_\rho \Bigg[\frac{1}{H_p d_{ij}(\mathbf{0})} \left(\frac{1}{d_{ij}(\mathbf{0})} + \frac{3}{d_{ij}(\mathbf{0}) - D} \right) \\ P_U^T \left(P_X \boldsymbol{x}_i(k) - \mathbf{Y}_j(k)\right) \left(P_X \boldsymbol{x}_i(k) - \mathbf{Y}_j(k)\right)^T P_U - P_U^T P_U \Bigg]. \end{split} \tag{4.14}$$

Basic simulations

To evaluate the performance of the previous obstacle avoidance constraints, some very simple simulations made in `Python` are presented hereafter. The experiment consists of two agents lying on the two opposite corners of a square, at the same elevation level. Then, each agent tries to reach the opposite corner, making the agents going one towards the other. Some Gaussian noise is added to the position of each agent neighbor, which gives certain randomness to the chosen evasion path.

Figure 4.2 shows a decentralized evasion maneuver using only average hard constraints of the form (4.10) between a green agent, who goes from the red dot $(0, 0, 1)$ to the black dot $(2, 2, 1)$, and a blue agent, who goes from the black dot to the red one. As we can see, the evasion maneuver occurs mainly vertically, along the Z axis, causing that one agent passes on top of the other one.

Nevertheless, in practice, when a quadrotor is at the same (x, y) coordinates but at a higher altitude than a neighboring one, a lot of disturbance is created over the quadrotor who is at the bottom, from air turbulence. This is why we propose to handle the evasion between agents only in the XY plane, considering the other agents as obstacles modeled as infinite cylinders.

Figure 4.3 shows a top view of the same experiment as before but with the difference that we force the evasion to occur only horizontally, using the cylindrical constraints. On the left is the evasion using only average hard constraints of the form (4.10), adapted to consider only horizontal evasion; the evasion maneuver is performed around $(0.8, 0.8)$

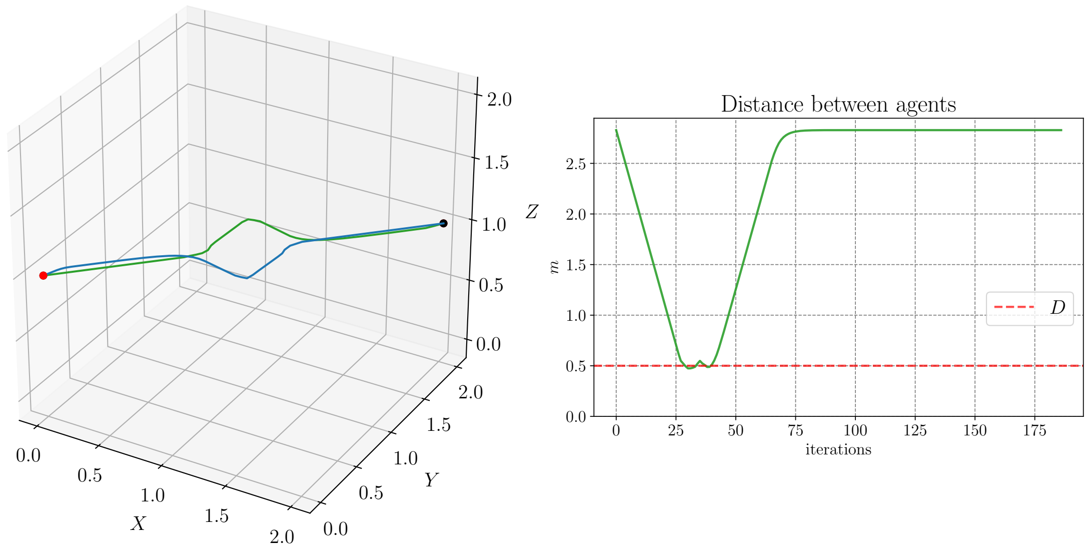


Figure 4.2: Decentralized evasion between two agents using only average hard constraints of the form (4.10). Evasion is performed mainly along the Z axis.

for the green agent and around $(1.2, 1.2)$ for the blue one. On the other hand, the figure in the right shows the same evasion experiment but using *also* soft constraints of the form (4.13), adapted for horizontal evasion; the evasion maneuver occurs earlier than the scenario on the left, taking place around $(0.75, 0.75)$ for the green agent and around $(1.25, 1.25)$ for the blue one. Also, the final trajectory presents a lot of oscillations. These oscillations that we observe with the use of soft constraints in the form of a penalty term in the cost function may appear because, at some iterations, the penalty term has increased a lot and leads the agents to evade themselves, affecting the consensus, and then, when the penalty term has decreased, the agents become closer again in order to achieve their target positions.

Along with Figure 4.3, Figure 4.4 shows the distance between the two agents during the simulation. For the scenario using only average hard constraints, sometimes the distance reaches the security distance D, even trespassing this bound at some iterations. In the case of the right figure, where we *also* use soft constraints, the distance between the agents never reaches the security distance D, but it oscillates a lot in its vicinity.

4.1.4 Connectivity maintenance

The connectivity between agents, that is defined through the graph G modeling the communication between agents, usually depends on the distance between them. Hence,

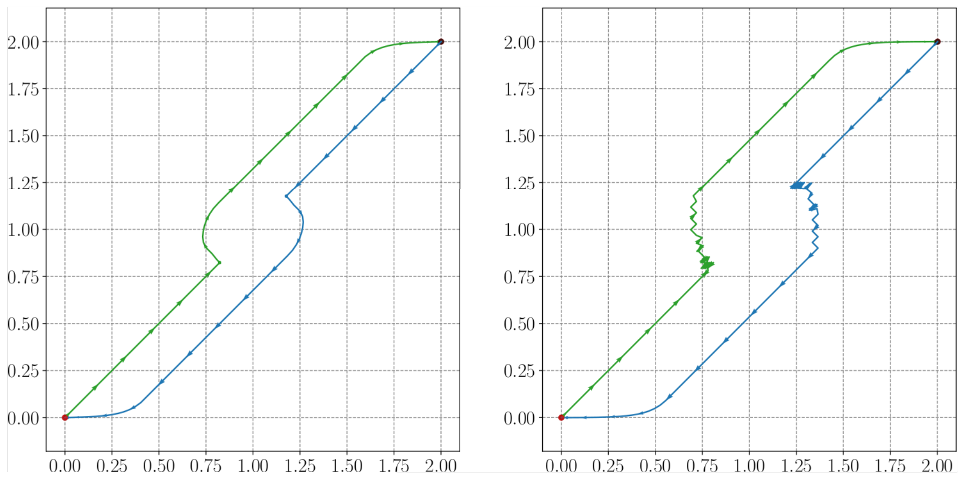


Figure 4.3: Decentralized evasion between two agents. On the left: Using only the average hard constraints of the form (4.10). On the right: Using *also* the linear soft constraints of the form (4.13).

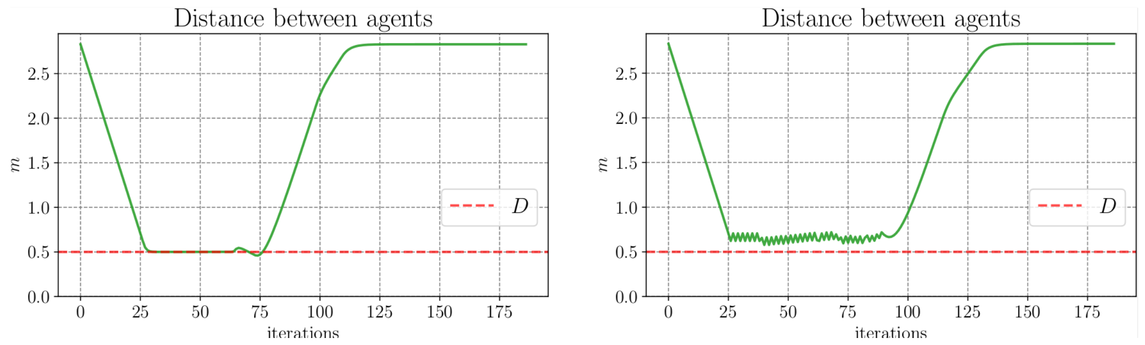


Figure 4.4: Distance between two agents for the decentralized evasion maneuver of Figure 4.3. On the left: Using only the average hard constraints of the form (4.10). On the right: Using *also* the linear soft constraints of the form (4.13).

in order to preserve the connectivity between two neighbor agents, the distance between them should be top-limited by a constant E, which corresponds to the maximum distance that two neighbor agents can have without losing communication. We can rewrite the constraints for obstacle avoidance in order to satisfy connectivity maintenance.

Hard constraints

Considering the average distance between an agent i and its neighbor $j \in N_i$ along the prediction horizon d_{ij}, given by (4.5), and following the idea of avoidance constraints in (4.6), we can write connectivity maintenance constraints as

$$d_{ij}^2 \leq E^2. \tag{4.15}$$

As we did for the obstacle avoidance constraints, we need to consider linear constraints in terms of $\boldsymbol{U}_i(k)$. Hence, considering the linear approximation of the average distance $\bar{d}_{ij}^2$ given by (4.9), the constraints of the form (4.15) are substituted by

$$\bar{d}_{ij}^2 \leq E^2. \tag{4.16}$$

Soft constraints

Following the idea used for obstacle avoidance, of a penalty term of the form of a potential field, the term (4.11) can be adapted for connectivity maintenance as

$$\tau_{ij} = \frac{k_c}{\left(d_{ij} - E\right)^2}, \tag{4.17}$$

where $k_c > 0$ and d_{ij} is the average distance defined in (4.5). This penalty term becomes larger as the agents are farther and near the maximum distance E.

Following the same idea of Taylor approximation in (4.12), we can consider the linear penalty term to be added to the cost function as

$$-k_\tau \left(P_X \boldsymbol{x}_i(k) - \mathbf{Y}_j(k)\right)^T P_U \boldsymbol{U}_i(k), \tag{4.18}$$

where $k_\tau = \dfrac{2k_c}{H_p d_{ij}(\mathbf{0}) \left(d_{ij}(\mathbf{0}) - E\right)^3}$.

We can also consider the quadratic term of the Taylor approximation by adding to the matrix $H(k)$ of the quadratic form, the matrix

$$\begin{aligned} k_\tau \Bigg[& \frac{1}{H_p d_{ij}(\mathbf{0})} \left(\frac{1}{d_{ij}(\mathbf{0})} + \frac{3}{d_{ij}(\mathbf{0}) - E} \right) \\ & P_U^T \left(P_X \boldsymbol{x}_i(k) - \mathbf{Y}_j(k)\right) \left(P_X \boldsymbol{x}_i(k) - \mathbf{Y}_j(k)\right)^T P_U - P_U^T P_U \Bigg]. \end{aligned} \tag{4.19}$$

Using both types of soft constraints, the ones for obstacle avoidance and the ones for connectivity maintenance, results on the sum of both penalty terms, giving a potential field of two asymptotes for the average distance d_{ij}. Figure 4.5 illustrates the penalty terms when they are used only for obstacle avoidance or connectivity maintenance, or for both tasks simultaneously.

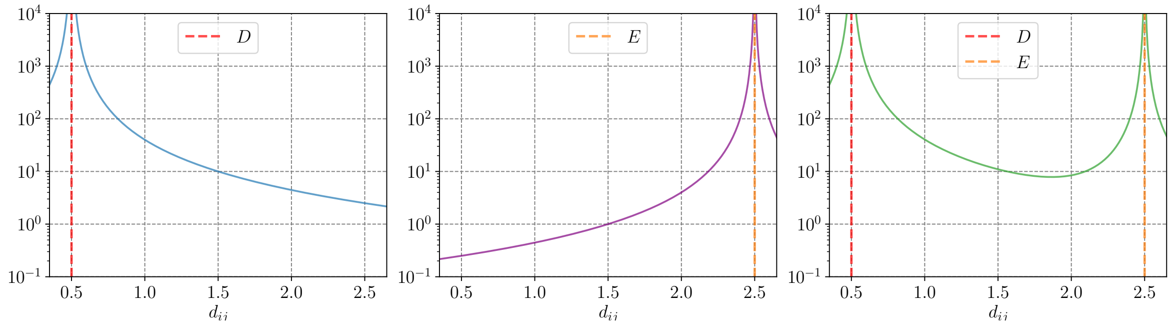


Figure 4.5: Shape of the soft constraints. Left: Obstacle avoidance. Center: Connectivity maintenance. Left: Both.

4.2 Decentralized architecture with local consensus error

Instead of generating reference trajectories for each agent, we can also use the theoretical consensus values between neighbors at each iteration as a reference for the Model Predictive Control cost function, i.e., not separating the two processes. This approach is inspired by the results of [CFZ15].

As in the previous section, we consider virtual agents defined as in (2.27) and we aim at controlling them to achieve consensus between them and reach the desired formation for the corresponding real agents. In order to achieve this, a communication graph with a spanning tree is enough [CFZ15].

4.2.1 Cost function for the Model Predictive Control

The cost function for each agent i is defined as follows

$$\Phi_i(k) = \beta_i \sum_{t=H_w}^{H_p} ||\boldsymbol{z}_i(k+t|k) - \boldsymbol{q}_i(k)||^2 + \gamma_i \sum_{t=1}^{H_u} ||\boldsymbol{u}_i(k+t-1|k)||^2, \tag{4.20}$$

where

$$\boldsymbol{q}_i(k) = \frac{1}{|N_i(k)|+1} \sum_{j \in N_i(k) \cup \{i\}} \boldsymbol{z}_j(k). \tag{4.21}$$

As mentioned above, one can see that the objective function is now expressed as the sum of predicted consensus errors in the near future (instead of deviations to the reference trajectory arising from solving the consensus problem separately) and control errors (similar to Eq. 4.3). At each iteration, the reference value $\boldsymbol{q}_i$ is the same for all the prediction horizon, because we do not have more information about what will occur to the neighbors. The term $\boldsymbol{q}_i(k)$ corresponds to the theoretical consensus of the actual positions of virtual agent i and its neighbors, as presented in (2.25).

As we are still considering a decentralized control architecture, in order to avoid collisions, the same constraints defined in (4.4), (4.6) and (4.11) can be used.

4.3 Distributed architecture

The main advantage of using a distributed control architecture (see Section 2.3) relies in the fact that each agent can take into account the movements that are going to be performed by its neighbors, since they also communicate their control inputs.

Again, we are going to look for the consensus of virtual agents as defined in (2.27) to consequently achieve the agents formation.

4.3.1 Cost function for the Model Predictive Control

It is worth noting that the two options of decentralized schemes described in the previous sections can be extended to the distributed form. On the one hand, the cost function using the reference trajectories for each agent defined in (4.3) keeps the same form by using a distributed control architecture, as the reference path is predefined and cannot be changed.

On the other hand, the cost function that considers the local consensus error defined in (4.20) presents an opportunity to take advantage of distributed control. The reference value $\boldsymbol{q}_i$ can change along the prediction horizon by considering a different consensus theoretical value for each term in the prediction horizon. In this case, the cost function of agent i is defined as

$$\Phi_i(k) = \beta_i \sum_{t=H_w}^{H_p} ||\boldsymbol{z}_i(k+t|k) - \boldsymbol{q}_i(k+t|k)||^2 + \gamma_i \sum_{t=1}^{H_u} ||\boldsymbol{u}_i(k+t-1|k)||^2, \tag{4.22}$$

where

$$\boldsymbol{q}_i(k+t|k) = \frac{1}{|N_i(k)|+1} \sum_{j \in N_i(k) \cup \{i\}} \boldsymbol{z}_j(k+t|k). \tag{4.23}$$

As it appears in (4.23), agent i considers that its neighbor $j \in N_i$ will apply the same control input as the one that it has computed through its own MPC and that it has communicated to him along all the prediction horizon, that is

$$\begin{bmatrix} \boldsymbol{z}_j(k+1|k) \\ \boldsymbol{z}_j(k+2|k) \\ \vdots \\ \boldsymbol{z}_j(k+H_p|k) \end{bmatrix} = P_X \boldsymbol{z}_j(k) + P_U \begin{bmatrix} \tilde{\boldsymbol{u}}_j(k) \\ \tilde{\boldsymbol{u}}_j(k) \\ \vdots \\ \tilde{\boldsymbol{u}}_j(k) \end{bmatrix}, \tag{4.24}$$

where $\tilde{\boldsymbol{u}}_j(k)$ is the communicated control input from j to i and P_X, P_U are defined as in (2.37), (2.38) according to the dynamics in (4.1).

The reference $\boldsymbol{q}_i(k+t|k)$ for the distributed architecture depends not only on the predicted states of the neighbors of agent i, but also on the predicted states of the agent itself, that involves a linear dependency of $\boldsymbol{q}_i(k+t|k)$ on the inputs $\boldsymbol{u}_i$ that are going to be computed.

An extended reference vector $\boldsymbol{Q}_i(k+1)$ along the prediction horizon can be defined as follows

$$\boldsymbol{Q}_i(k+1) = \frac{1}{|N_i(k)|+1} \begin{bmatrix} \sum_{j\in N_i(k)} \boldsymbol{z}_j(k+1|k) \\ \sum_{j\in N_i(k)} \boldsymbol{z}_j(k+2|k) \\ \vdots \\ \sum_{j\in N_i(k)} \boldsymbol{z}_j(k+H_p|k) \end{bmatrix} + \frac{1}{|N_i(k)|+1} \begin{bmatrix} \boldsymbol{z}_i(k+1|k) \\ \boldsymbol{z}_i(k+2|k) \\ \vdots \\ \boldsymbol{z}_i(k+H_p|k) \end{bmatrix}. \tag{4.25}$$

As stated in (2.43), vector $\boldsymbol{Q}_i(k+1)$ can be rewritten as $\boldsymbol{Q}_i(k+1) = \boldsymbol{A}_i(k) + B_i(k)\boldsymbol{U}_i(k)$ where

$$\boldsymbol{A}_i(k) := \frac{1}{|N_i(k)|+1} \begin{bmatrix} \sum_{j\in N_i(k)} \boldsymbol{z}_j(k+1|k) \\ \sum_{j\in N_i(k)} \boldsymbol{z}_j(k+2|k) \\ \vdots \\ \sum_{j\in N_i(k)} \boldsymbol{z}_j(k+H_p|k) \end{bmatrix} + \frac{1}{|N_i(k)|+1} P_X \boldsymbol{z}_i(k),$$

$$B_i(k) := \frac{1}{|N_i(k)|+1} P_U.$$

4.3.2 Obstacle avoidance

Considering the communicated control inputs in the constraints for obstacle avoidance may lead to an anticipated or smoother evasion maneuver.

The average distance along the prediction horizon between the agent i and a neighbor $j \in N_i$ will now be defined as

$$h_{ij}^2 := \frac{||\boldsymbol{X}_i(k+1) - \tilde{\boldsymbol{X}}_j(k+1)||^2}{H_p}, \tag{4.26}$$

where $\boldsymbol{X}_i(k+1)$ is defined as in (4.5) and $\tilde{\boldsymbol{X}}_j(k+1)$ gathers all the predicted states for the neighbor j given $\tilde{\boldsymbol{u}}_j(k)$, the communicated control input from j to i:

$$\tilde{\boldsymbol{X}}_j(k+1) := \begin{bmatrix} \tilde{\boldsymbol{x}}_j(k+1|k) \\ \tilde{\boldsymbol{x}}_j(k+2|k) \\ \vdots \\ \tilde{\boldsymbol{x}}_j(k+H_p|k) \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}_j(k+1|k) \\ \boldsymbol{z}_j(k+2|k) \\ \vdots \\ \boldsymbol{z}_j(k+H_p|k) \end{bmatrix} + \begin{bmatrix} \boldsymbol{d}_j \\ \boldsymbol{d}_j \\ \vdots \\ \boldsymbol{d}_j \end{bmatrix}. \tag{4.27}$$

Hard constraints

The constraint presented in (4.4) can be modified for the distributed architecture as follows:

$$||\boldsymbol{x}_i(k+t|k) - \tilde{\boldsymbol{x}}_j(k+t|k)||^2 \geq D^2. \tag{4.28}$$

Using just the average distance, the constraint (4.6) becomes

$$h_{ij}^2 \geq D^2. \tag{4.29}$$

Following the linearization presented in (4.9) and given that $\tilde{\boldsymbol{X}}_j(k+1)$ is independent of $\boldsymbol{U}_i(k)$, the linear model for $h_{ij}^2\left(\boldsymbol{U}_i(k)\right)$ is defined as

$$\bar{h}_{ij}^2\left(\boldsymbol{U}_i(k)\right) = \frac{\left|\left|P_X \boldsymbol{x}_i(k) - \tilde{\boldsymbol{X}}_j(k+1)\right|\right|^2}{H_p} + \frac{2}{H_p}\left(P_X \boldsymbol{x}_i(k) - \tilde{\boldsymbol{X}}_j(k+1)\right)^T P_U \boldsymbol{U}_i(k) \tag{4.30}$$

then the constraint (4.29) can be substituted by

$$\bar{h}_{ij}^2 \geq D^2. \tag{4.31}$$

Soft constraints

The decentralized penalty term in (4.11) becomes

$$\sigma_{ij} = \frac{k_d}{\left(h_{ij} - D\right)^2}. \tag{4.32}$$

In order to solve the optimization problem as a quadratic programming problem, as we did in (4.13), we can consider the linear penalty term to be added as

$$-k_\sigma \left(P_X \boldsymbol{x}_i(k) - \tilde{\boldsymbol{X}}_j(k+1) \right)^T P_U \boldsymbol{U}_i(k) \tag{4.33}$$

where $k_\sigma = \dfrac{2k_d}{H_p h_{ij}(\mathbf{0}) \left(h_{ij}(\mathbf{0}) - D\right)^3}$.

As for the decentralized case, we can also consider the quadratic term of the Taylor approximation of $\sigma_{ij}(\boldsymbol{U}_i(k))$ around $\boldsymbol{U}_i = \mathbf{0}$ by adding to the matrix $H(k)$ of the quadratic form, the matrix

$$\begin{aligned} k_\sigma \Bigg[& \frac{1}{H_p h_{ij}(\mathbf{0})} \left(\frac{1}{h_{ij}(\mathbf{0})} + \frac{3}{h_{ij}(\mathbf{0}) - D} \right) \\ & P_U^T \left(P_X \boldsymbol{x}_i(k) - \tilde{\boldsymbol{X}}_j(k) \right) \left(P_X \boldsymbol{x}_i(k) - \tilde{\boldsymbol{X}}_j(k) \right)^T P_U - P_U^T P_U \Bigg]. \end{aligned} \tag{4.34}$$

Basic simulations

Figure 4.6 shows the same experiment as in the previous section in Figure 4.3, but considering now a distributed architecture. On the left is the evasion maneuver using only the average hard constraints of the form (4.31), adapted to consider only horizontal evasion; the evasion maneuver is performed around $(0.62, 0.62)$ for the green agent and around $(1.37, 1.37)$ for the blue one. On the other hand, the figure on the right shows the same evasion experiment but using *also* the soft constraints of the form (4.33) adapted for horizontal evasion; the evasion maneuver occurs around the same coordinates as for the left scenario, but in a different direction. This is probably caused by the randomness introduced by the Gaussian noise added to the perception of neighbors.

The whole evasion maneuver starts earlier than the one that we observed in the decentralized case, because each agent knows in advance what its neighbor is planning to do. Also, the final trajectory for the right scenario presents much less oscillations than the equivalent one in the decentralized architecture.

Along with Figure 4.6, Figure 4.7 shows the distance between the two agents along the simulation. For the scenario using only the average hard constraints, the distance only approaches and almost reaches the security distance D. In the case of the right figure, using *also* the soft constraints, the distance between the agents never reaches the security distance D, and the oscillations are drastically decreased from what we observed with the decentralized architecture.

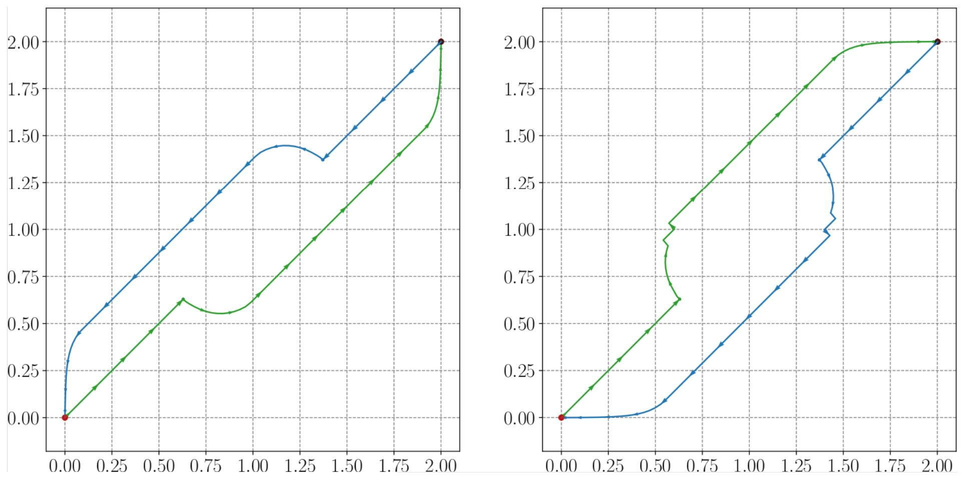


Figure 4.6: Distributed evasion maneuver between two agents. On the left: Using only the average hard constraints of the form (4.31). On the right: Using *also* the linear soft constraints of the form (4.33).

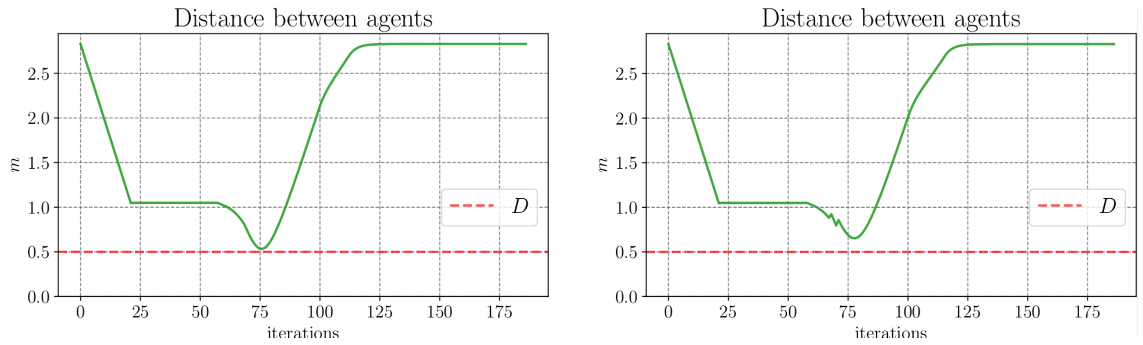


Figure 4.7: Distance between two agents for the distributed evasion maneuver of Figure 4.6. On the left: Using only the average hard constraints of the form (4.31). On the right: Using *also* the linear soft constraints of the form (4.33).

4.3.3 Connectivity maintenance

Regarding connectivity maintenance, we will see that considering the communicated controls may also result in smoother trajectories.

Hard constraints

Considering the average distance h_{ij} between an agent i and its neighbor $j \in N_i$ along the prediction horizon, given by (4.26), following the idea of avoidance constraints in (4.29), we can write connectivity maintenance constraints as

$$h_{ij}^2 \leq E^2. \tag{4.35}$$

As we did for obstacle avoidance constraints, we need to consider linear constraints in terms of $\boldsymbol{U}_i(k)$. Hence, considering the linear approximation of the average distance $\bar{h}_{ij}^2$ given by (4.30), the constraints of the form (4.15) are substituted by

$$\bar{h}_{ij}^2 \leq E^2. \tag{4.36}$$

Soft constraints

The decentralized penalty term in (4.17) can be adapted for a distributed architecture as

$$\upsilon_{ij} = \frac{k_c}{(h_{ij} - E)^2}, \tag{4.37}$$

where $k_c > 0$ and h_{ij} is the average distance defined in (4.26). This penalty term becomes larger as the agents are farther and near the maximum distance E. Then, we can consider the linear penalty term to be added to the cost function as

$$-k_v \left(P_X \boldsymbol{x}_i(k) - \tilde{\boldsymbol{X}}_j(k) \right)^T P_U \boldsymbol{U}_i(k), \tag{4.38}$$

where $k_v = \dfrac{2k_c}{H_p h_{ij}(\mathbf{0}) \left(h_{ij}(\mathbf{0}) - E\right)^3}$.

We can also consider the quadratic term of the Taylor approximation by adding to the matrix $H(k)$ of the quadratic form, the matrix

$$\begin{aligned} k_v \Bigg[& \frac{1}{H_p h_{ij}(\mathbf{0})} \left(\frac{1}{h_{ij}(\mathbf{0})} + \frac{3}{h_{ij}(\mathbf{0}) - E} \right) \\ & P_U^T \left(P_X \boldsymbol{x}_i(k) - \tilde{\boldsymbol{X}}_j(k) \right) \left(P_X \boldsymbol{x}_i(k) - \tilde{\boldsymbol{X}}_j(k) \right)^T P_U - P_U^T P_U \Bigg]. \end{aligned} \tag{4.39}$$

4.4 Discussion and strategy selection

Now we are going to discuss the main advantages and disadvantages of the different strategies presented along this chapter. Based on this discussion, we are going to select the best strategy in three main areas of the formation control: the type of the reference to be used in the cost function of the Model Predictive Control, i.e., predefined reference or local consensus error; the types of the constraints to be considered for both obstacle evasion and connectivity maintenance, i.e., average hard and/or soft constraints; and finally the control architecture (decentralized or distributed).

All the experiments along this section consider a fully connected communication graph for the agents, which means that every agent knows the positions, and the computed controls for the distributed architecture, of all the other agents.

Reference for the Model Predictive Control

Regarding the reference for the Model Predictive Control, using a predefined consensus path offers some advantages over using local consensus error; the first concerns computation time, since having a predefined consensus path decreases the amount of computations that are needed at each iteration; another advantage relates with the evasion as, when an agent faces an obstacle, it just needs to deviate from its reference and then return to its predefined path, without affecting the reference for the other agents.

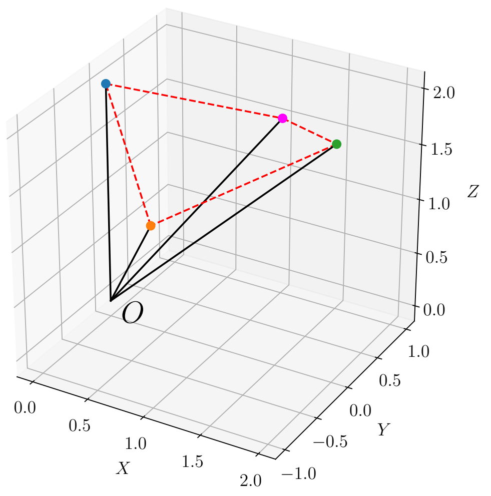
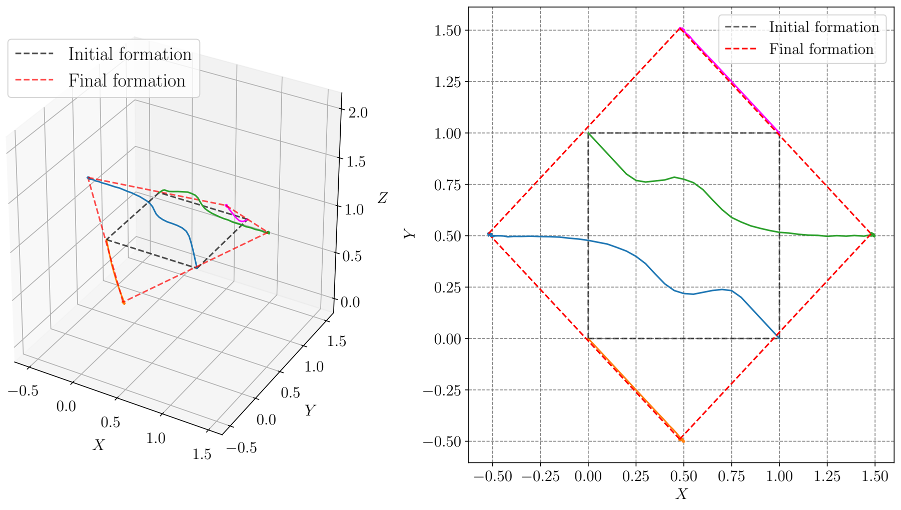


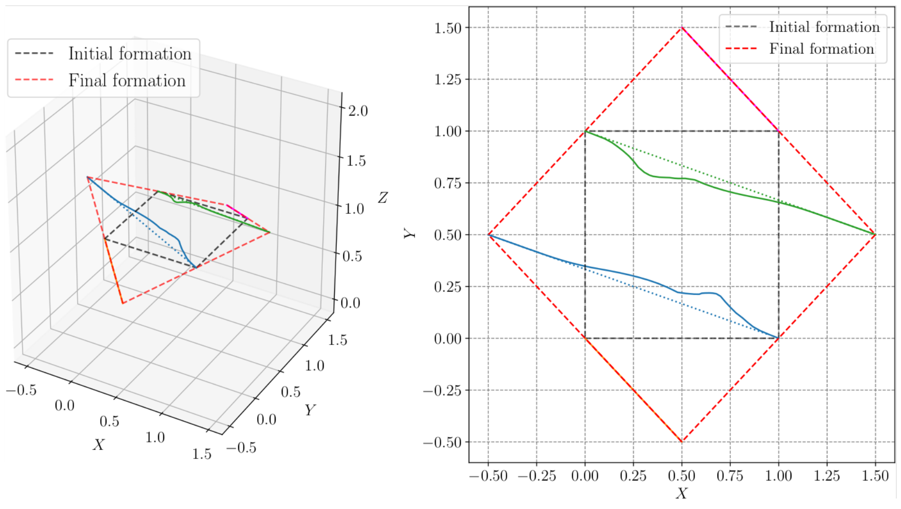
Figure 4.8: Target formation for 4 quadrotors.

We first compared the performance of the approaches in the way in which the reference trajectory is generated. The experiment aims to achieve a formation as the one in Figure 4.8, which shows the displacement vectors that defines the desired formation for 4 agents. Figure 4.9 shows a comparison between using as a reference the local consensus error or the predefined consensus trajectories. The experiment is designed in such a way that two agents must avoid each other. For evasion between agents, average hard constraints of the form (4.10) and a decentralized control architecture are considered.

The reference trajectories consist on straight lines from the initial formation to the target one. However, when evasion constraints are added, the final trajectories differ from the reference ones, but the evasion between the blue and green agents does not affect the trajectories of the two other agents. In the case of the use of local consensus



(a) Local consensus error



(b) Predefined reference

Figure 4.9: Decentralized formation experiment using evasion constraints of the form (4.10). On the top: Using local consensus error as reference. On the bottom: Using predefined reference paths (dotted lines).

error as reference for the MPC, the trajectories are longer and perform as curves, even for the agents that do not need to evade themselves. This can be appreciated in Figure 4.10, that shows the trajectories and final consensus of the virtual agents. Using the local consensus error as the reference, the trajectories of the orange and purple agents are more curvy, even though these agents do not need to evade themselves; in contrast, using predefined paths as the reference leads to straighter trajectories that are strictly followed if the agents do no need to evade an obstacle.

It is important to remark that the predefined consensus trajectories should be feasible in the sense of the bounds of the input controls that can be applied to the system. If the

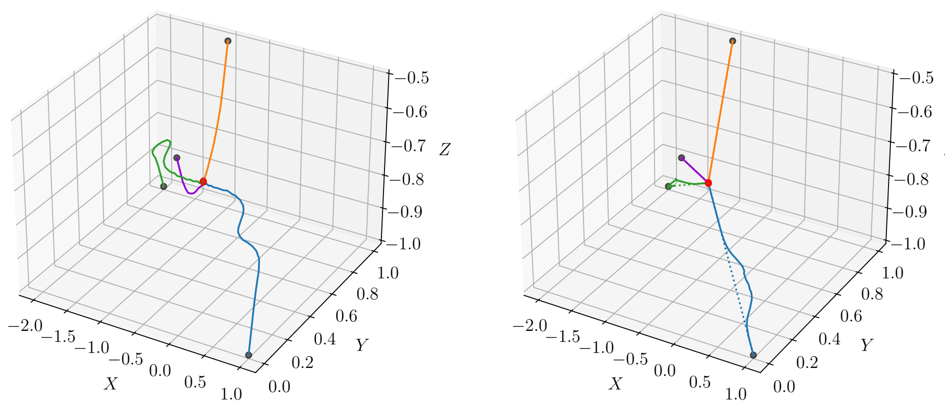


Figure 4.10: Trajectories of the virtual agents for the formation experiment of Figure 4.9. Left: Using local consensus error as reference. Right: Using predefined paths (dotted lines).

predefined consensus trajectory advances too fast in the time, then the controls generated with the MPC can saturate and make impossible to reach the reference trajectory. Figure 4.11 shows the controls generated for the formation experiment in Figure 4.9 using predefined consensus trajectories as reference for the MPC. We can see that the controls saturate a little at the beginning, but then the reference trajectories advance slower and this makes possible for the MPC to reach them; the controls saturate again when there is a need of evasion between the agents. If the predefined reference trajectories advance really fast, the behavior of the generated trajectories of the MPC will be very similar to the ones generated using as a reference the local consensus error.

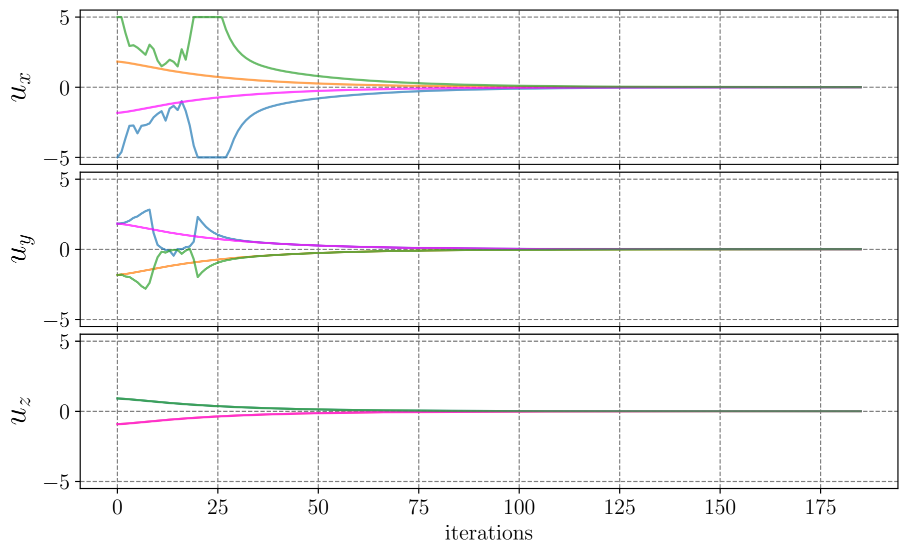


Figure 4.11: Input controls for the Figure 4.9 (b).

As a side note, even though we are considering a complete communication graph, a graph with a spanning tree is enough to achieve consensus between the virtual agents and therefore the desired formation. For obstacle avoidance, only the communication between the blue and green agents is needed. Then, the communication could be relaxed with a lighter graph, depending on the formation requirements for each experiment.

Nevertheless, the main disadvantage of using a predefined consensus path as reference, arises with the presence of fixed obstacles in the environment. In this case, we cannot guarantee the achievement of the formation because, if the final position of the formation given the predefined consensus path is obstructed by any fixed obstacle, then the agents will not be able to reach the desired configuration. That is why we choose as a reference for the MPC the local consensus error, because of its capability to adapt to different environments, even with the presence of fixed obstacles and even when the evasion maneuvers of one agent affect the consensus reference.

Type of constraints

Regarding the obstacle avoidance strategies, we have three choices for implementation: use only average hard constraints of the form (4.10) or (4.31), depending on the architecture; use only soft constraints on the form of a penalty term such as (4.13) or (4.33); or finally, use a combination of the previous two, i.e. both hard and soft constraints.

Beyond the advantages and disadvantages of the different strategies presented through the previous figures of this chapter, such as approaching and occasionally trespassing the security distance D, or the presence of a lot of oscillations, here we are going to present some cases of failures corresponding to the different strategies for obstacle avoidance.

As stated in [KN15], using only soft constraints of the form (4.13) or (4.33) has shown to be enough to avoid collisions between agents. Nevertheless, in some cases, the use of only this type of constraints makes impossible to reach the position reference. As we can see in Figure 4.12, facing the same experiment as in Figure 4.3, the use of only soft constraints of the form (4.13) makes impossible in some cases to reach the reference for each agent, as the distance between the agents keeps oscillating forever and none of the agents knows what to do next.

However, when we add average hard constraints of the form (4.10) to the same experiment, the agents are now able to avoid themselves and reach successfully theirs references. This can be appreciated in Figure 4.13.

In the cases where the use of only soft constraints of the form (4.13) is enough to reach the reference, adding average hard constraints of the form (4.10) helps to reduce

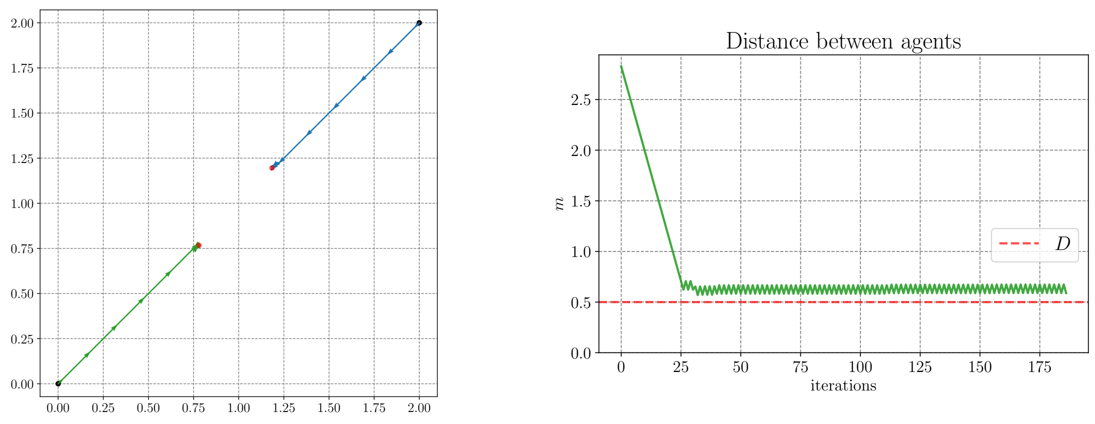


Figure 4.12: Case of failure for decentralized evasion between two agents using only the soft constraints of the form (4.13).

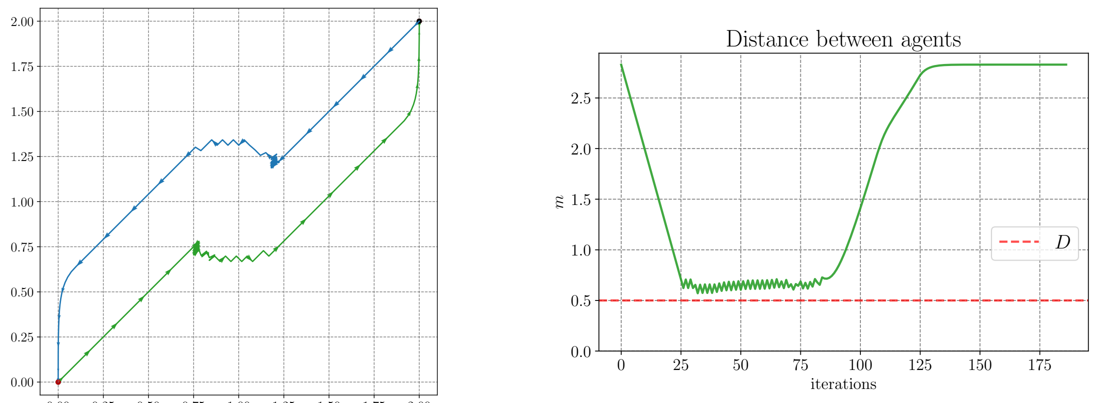


Figure 4.13: The same evasion experiment as in Figure 4.12 using both hard and soft constraints.

the oscillations. In Figure 4.14, we present a case of success for an evasion maneuver between two agents using only the soft constraints. The oscillations appear along 85 iterations, while in Figure 4.15, using both soft and hard constraints, the oscillations appear only along 50 iterations.

Apparently, for the decentralized architecture, the use of only average hard constraints is not good enough to make the agents reach the reference for some avoidance cases. Figure 4.16 shows an experiment of an evasion maneuver between three agents, where the use of only average hard constraints is not sufficient to make the agents reach the formation, as the distance between the agents stays at the vicinity of the security distance D. Again, the use of soft constraints of the form (4.13) makes possible for the agents to reach the target reference, as it can be seen in Figure 4.17.

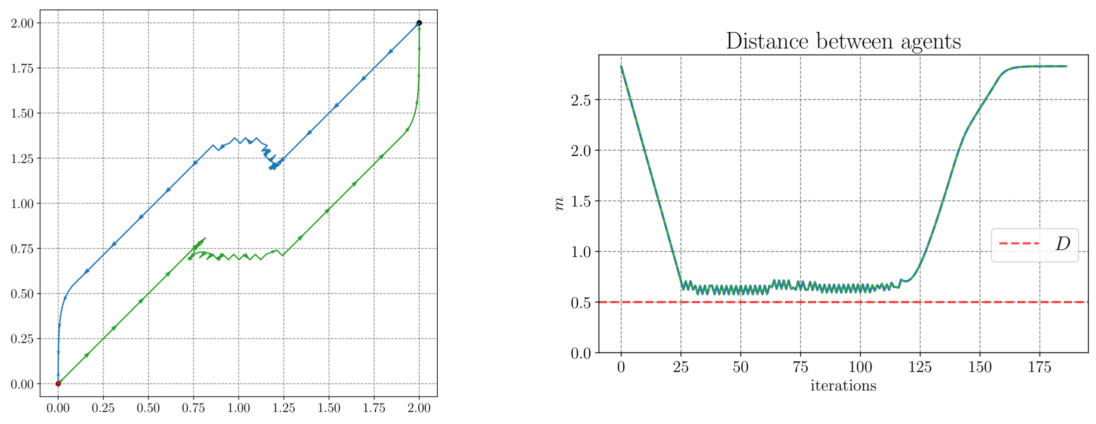


Figure 4.14: Case of success for a decentralized evasion maneuver between two agents using only the soft constraints of the form (4.13).

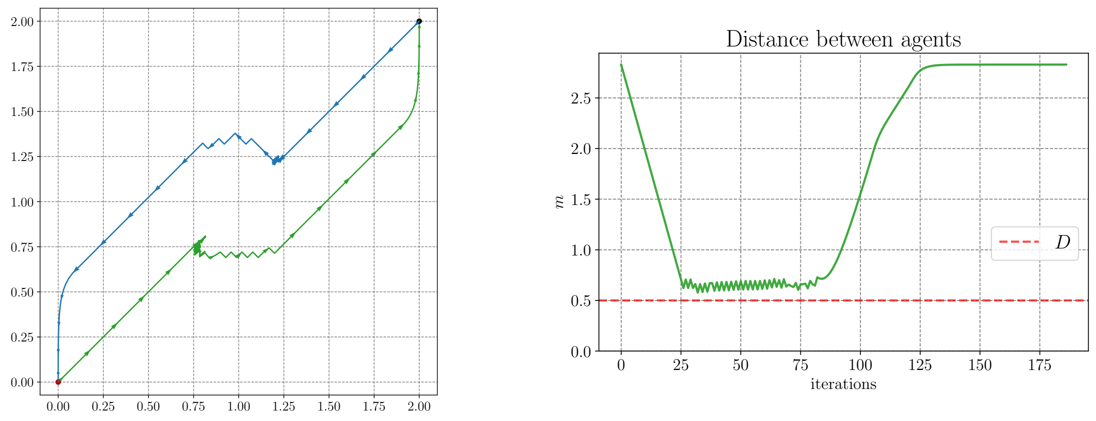


Figure 4.15: The same evasion experiment as in Figure 4.14 but using both the hard and soft constraints. The oscillations are considerably reduced.

To evaluate the ratio of success of each evasion strategy, we performed 3000 evasion experiments: 1000 between two agents, 1000 between three agents and 1000 between four agents, where each agent is on the corner of a square and attempts to reach the opposite corner. Each experiment was performed using either only average hard constraints, only soft constraints or the combination of both, for the decentralized and distributed architectures. Table 4.1 shows the results. It is important to remark that a failure case is considered when the agents could not reach their reference and that we have observed no case where the agents collide, i.e., the distance between the agents never trespass the security distance D.

It is interesting to notice that neither the average hard constraints alone, nor the soft constraints alone have a good performance for all the evasion cases. For example, in

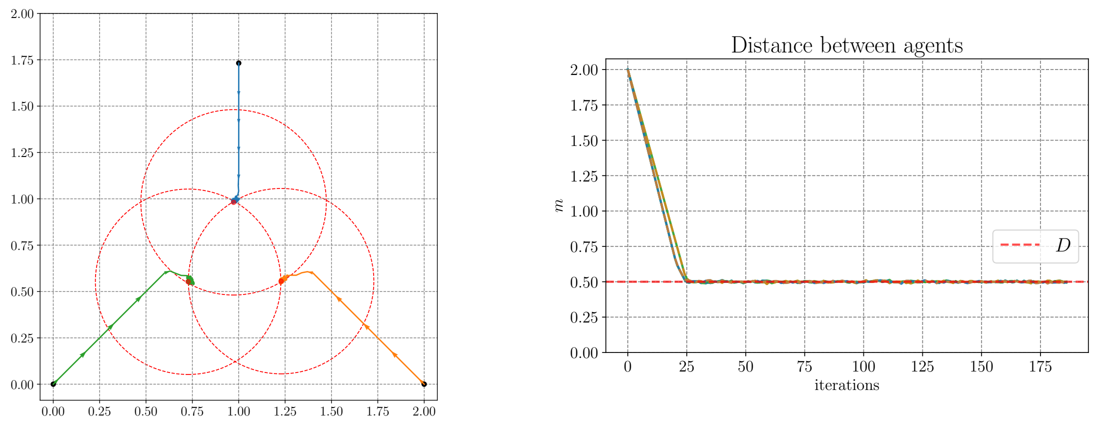


Figure 4.16: Case of failure for an evasion maneuver between three agents using only average hard constraints of the form (4.10). The red circles correspond to the security radius.

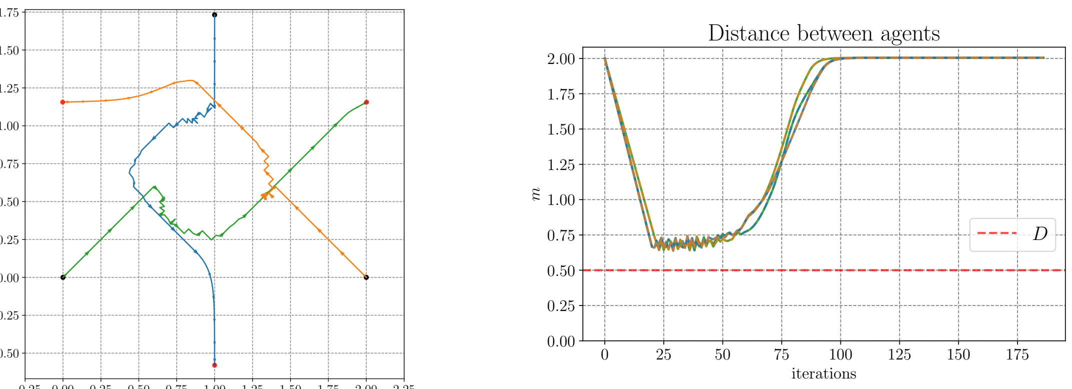


Figure 4.17: The same evasion experiment as in Figure 4.16 but using both hard and soft constraints.

N. agents	Decentralized			Distributed		
	Avg. hard	Soft	Both	Avg. hard	Soft	Both
2	100%	0.6%	100%	100%	0%	100%
3	0%	100%	100%	90.6%	71.4%	100%
4	0%	99.8%	100%	25.6%	12.9%	99.2%

Table 4.1: Percentage of success for evasion maneuvers between different number of agents using the three proposed evasion strategies, in both decentralized and distributed architectures.

the decentralized architecture, the use of only average hard constraints is effective only for the two agents case of evasion, while the use of only soft constraints is only effective for the evasion of three and four agents. However, the use of both types of constraints is effective for the three evasion cases, in both decentralized and distributed control architectures.

These results lead us to conclude that the best avoidance strategy consists on using both types of constraints, soft and average hard. They allow to reach the reference for the MPC and to reduce oscillations when the distance between the agents is near the security distance D.

We have run simulations for evasion experiments up to 30 agents, and the use of both types of constraints, soft and average hard, has shown to be effective to avoid collisions. The generated paths can be seen in Figure 4.18, for both decentralized and distributed architectures. The evasion experiments have as an initial configuration a regular polygon and each agent has to reach the diametrically opposite point of the circumference that circumscribes the polygon. We observed that the length of each side of the polygon should be four times the security distance D in order to give enough space for the evasion. Also we have not observed any increment on the computation time of the optimization problem, just an increment on the time needed to construct this problem.

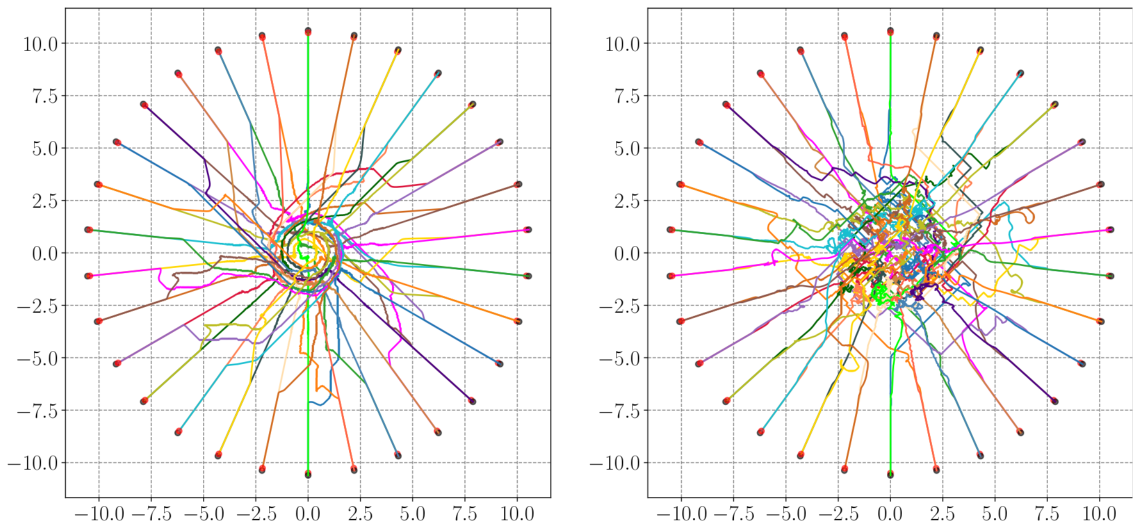


Figure 4.18: Generated paths for evasion between 30 agents using both hard and soft constraints. Left: Decentralized architecture. Right: Distributed architecture.

In the case of connectivity maintenance, as the proposed constraints are of the same form as the ones for evasion, based on the previous results, we choose also to use both types of constraints, soft and average hard constraints.

Control architecture

Regarding the choice of the control architecture, decentralized or distributed, we refer to the quality of the solution provided by each method. Figure 4.19 shows the same success case of Figure 4.15 but using a distributed control architecture. The oscillations in the generated trajectory and in the distance between agents have almost vanished compared to the decentralized case.

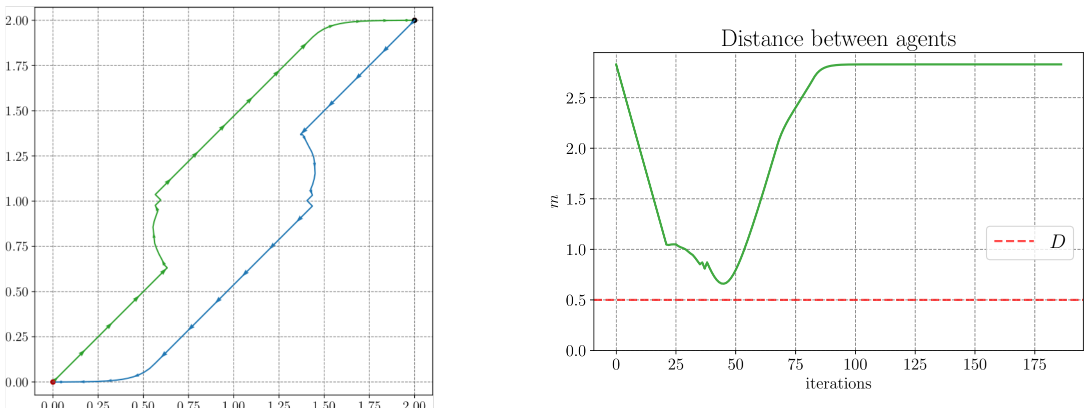
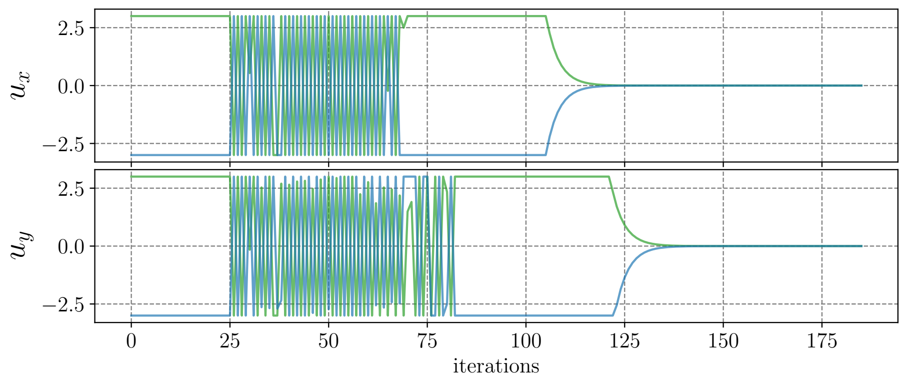


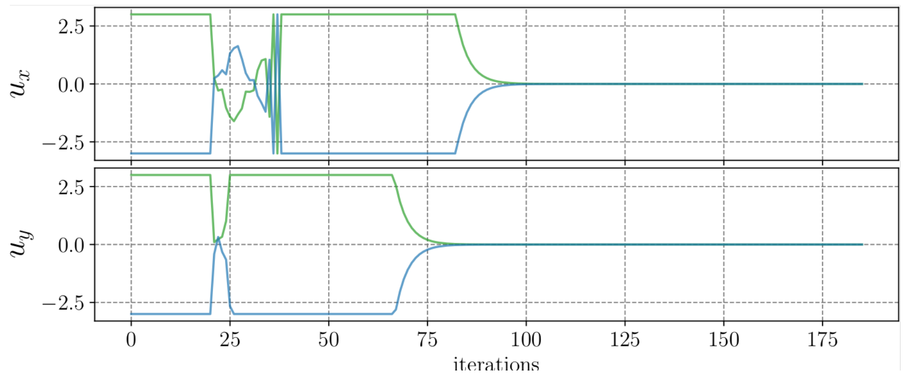
Figure 4.19: The same evasion experiment as in Figure 4.15 but using a distributed control architecture, resulting on a smoother trajectory.

Regarding the computed control inputs, Figure 4.20 shows, in the top, the computed controls for the decentralized architecture, and, in the bottom, the controls for the distributed architecture. As we can see, the computed controls oscillate drastically less in the distributed architecture. Another thing that we can notice is that the evasion effect in the controls starts earlier for the distributed case, where the controls change their behavior around 5 iterations before the corresponding time-step in the decentralized architecture; this is caused by the information that possess each agent in advance regarding the actions to be taken by its neighbors.

Given all these observations, we choose as our baseline formation control strategy an MPC using as reference the local consensus error between virtual agents, both average hard constraints of the form (4.31) and soft constraints of the form (4.33) adapted for horizontal evasion, and both average hard constraints of the form (4.36) and soft constraints of the form (4.38) for connectivity maintenance, all of this using a distributed control architecture.



(a) Decentralized architecture



(b) Distributed architecture

Figure 4.20: Computed controls for the evasion experiments of Figures 4.15 and 4.19.

Complete performance example

Here we illustrate the performance of a distributed architecture for formation control in the case of four quadrotors in an environment with a fixed post obstacle, using as reference the local consensus error, and both average hard constraints of the form (4.31) and linear soft constraints of the form (4.33) adapted for horizontal evasion with $D = 0.5$. The Figure 4.8 shows the target formation we desire to achieve. Figures 4.21 and 4.22 show the control results.

As we can see in Figure 4.21, the green and blue agents avoid the post and the purple and orange agents respectively. Also, the generated trajectories present few oscillations, due to the combination of the two types of constraints for evasion, i.e. average hard and soft ones.

The distance between the agents, as presented in Figure 4.22, never reaches the security distance D, but some oscillations can be observed near this limit. The oscillations in the distance between the blue and green agents are caused by the evasion maneuver with respect to the post, as we can see on the left.

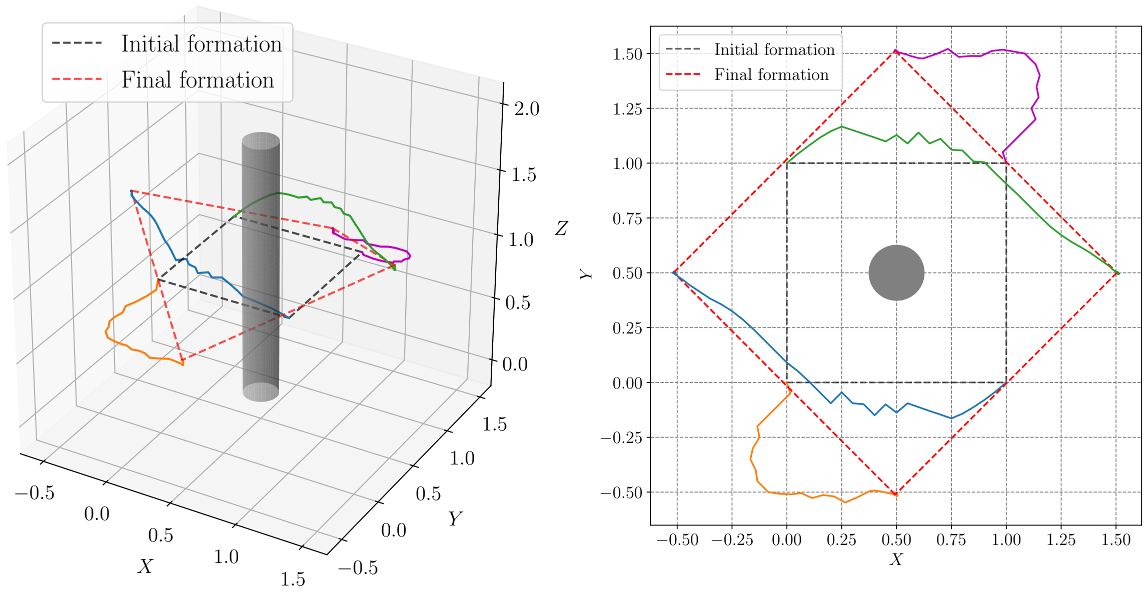


Figure 4.21: Distributed formation control of 4 agents, using both average hard constraints of the form (4.31) and linear soft constraints of the form (4.33).

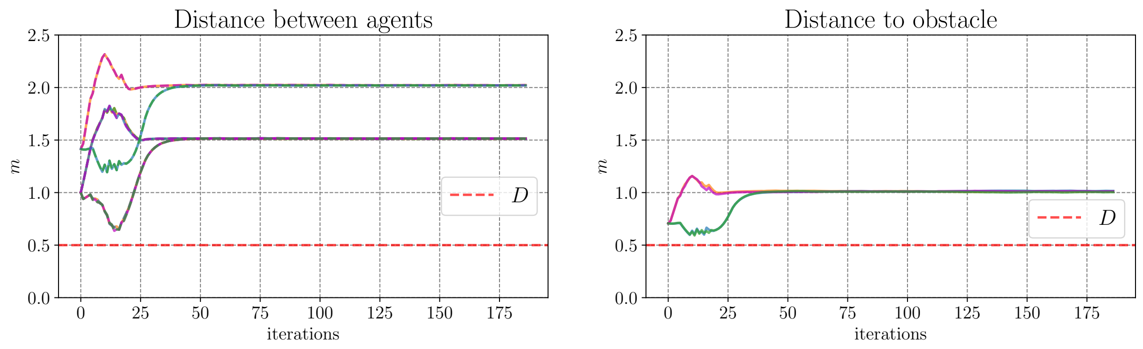


Figure 4.22: Distance between agents and to the obstacle for the distributed formation control of Figure 4.21.

Regarding the controls showed in Figure 4.23, some oscillations are observed, but the inputs are always restricted by the controls bounds. We can also see some disturbances caused by the added noise that the controls attempt to overcome.

If we consider also connectivity maintenance constraints of the form (4.36) and (4.18) with $E = 2.25$, we can see results as the ones in Figures 4.24, 4.25 and 4.26. Adding the connectivity constraints results in more complicated trajectories, as we can see in Figure 4.24. When the orange and purple agents try to stay within the connectivity maximum distance E, they have an impact in the trajectory of the other agents as well.

As we can see in Figure 4.25, the distance between the agents stays within the limits

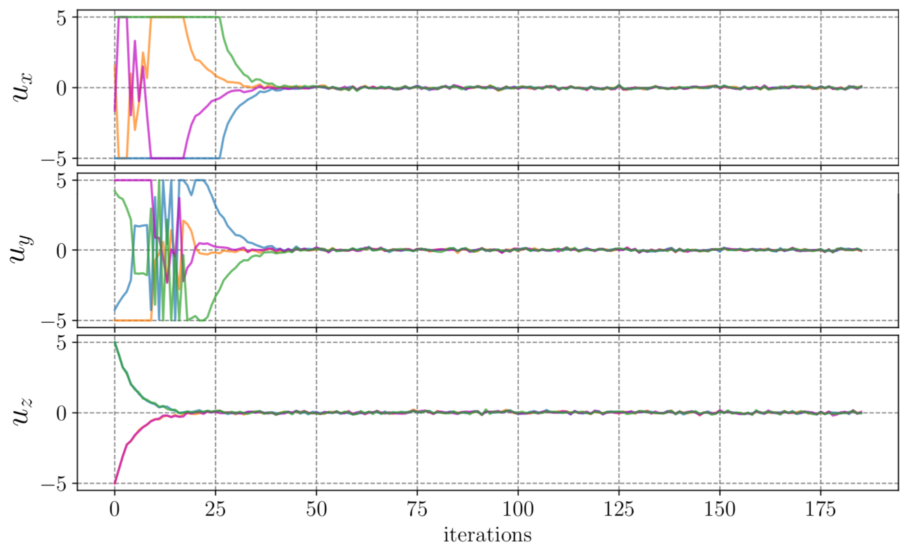


Figure 4.23: Computed controls for the formation experiment of Figure 4.21.

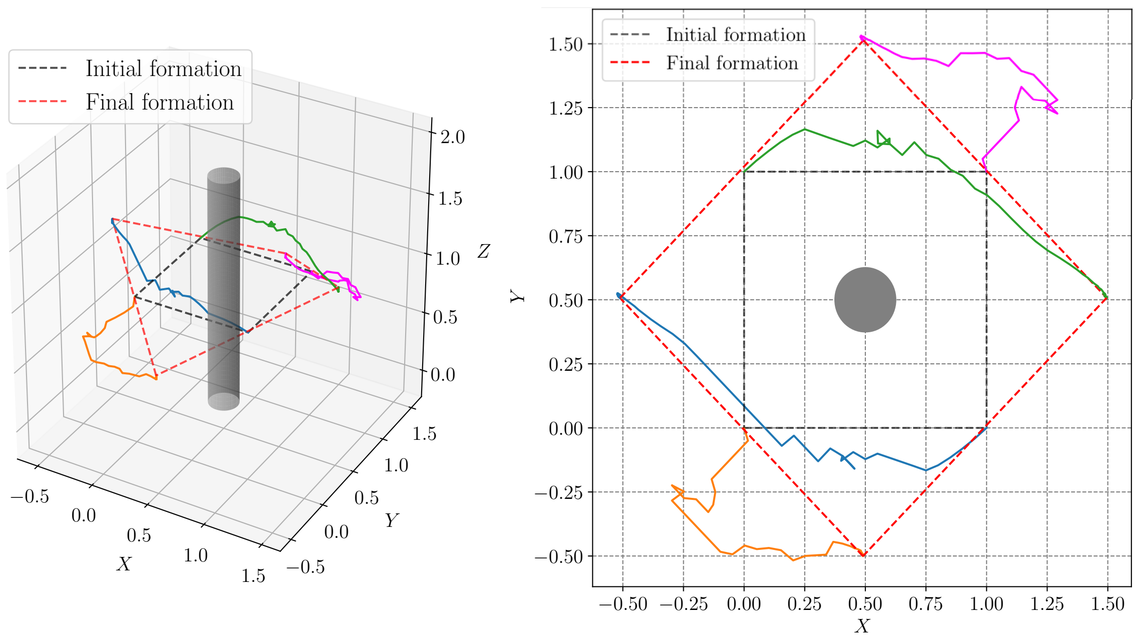


Figure 4.24: Distributed formation control of 4 agents, using both average hard constraints of the form (4.31) and (4.36), and linear soft constraints of the form (4.33) and (4.38), for both evasion avoidance and connectivity maintenance respectively.

of the security distance D and the connectivity maximum distance E. The distance between the orange and purple agents approaches to the distance E but does not trespass the limit. We can see some oscillations near the limits because of the use of the soft constraints both for obstacle avoidance and connectivity maintenance. Nevertheless, the use of average hard constraints makes these oscillations not to extend for a lot of iterations.

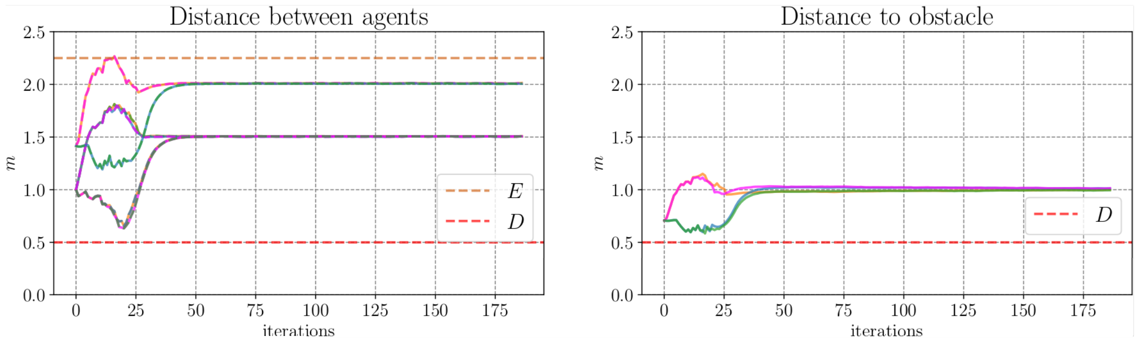


Figure 4.25: Distance between agents and to the obstacle for the distributed formation control of Figure 4.24.

Finally, in Figure 4.26, we can see the magnitude of the computed controls for this formation experiment. The controls oscillate more than in the previous case without connectivity maintenance constraints. Also, as the connectivity maintenance consider also the distance in the Z-axis, some changes in u_z can be seen for the orange and purple agents. The final formation is achieved later than for the previous case, but the difference is only of about 5 iterations.

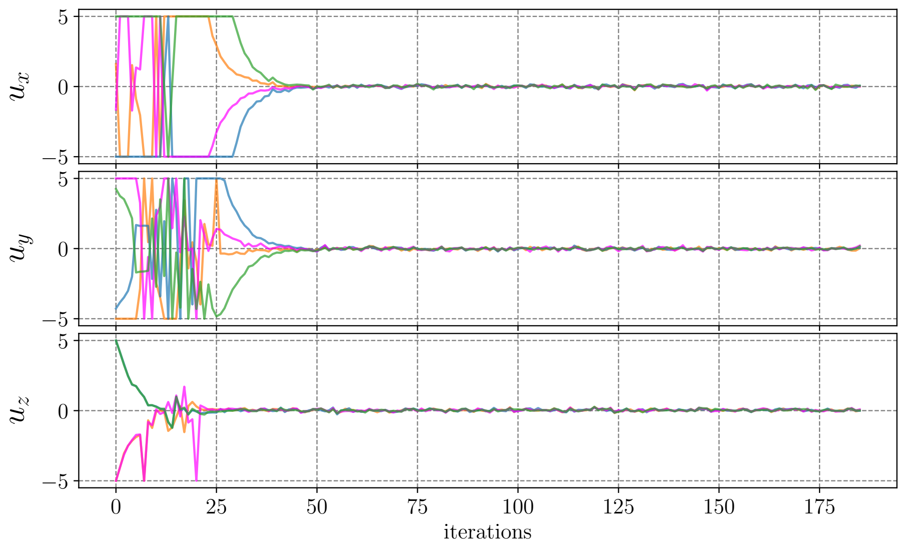


Figure 4.26: Computed controls for the formation experiment of Figure 4.24.

Chapter 5

Simulations and experiments

In this chapter, we present the results obtained both in simulations and in real experiments for the implementation of the control strategies presented in the previous chapter for the formation problem of quadrotors. First, in Section 5.1, the results for simulations are presented, and then some real-life experiments appear in Section 5.2.

5.1 Simulations in `Gazebo`

The control strategies presented in the previous chapter have been implemented in the `Gazebo` simulator [KH04] for the same formation problem of quadrotors given by Figure 4.8. For this purpose, we have used the `rotorS` package [FBAS16].

The control strategy presented in this work has as outputs linear velocities for the quadrotor, assuming that there exists a low level control that transforms these linear velocity commands into velocities for each rotor. This low level control can be of the form of the one presented in Section 2.2, but we have decided to use the low level control provided by the `rotorS` package, which we will describe hereafter. Nevertheless, the `rotorS` package includes only a position control. Hence, in order to achieve simulation, the velocities generated by our control are integrated at each time step, resulting on a position that is passed as a reference to the low-level position control implemented on the `rotorS` package.

Low level control in `rotorS`

The position control included in the `rotorS` package is an implementation of [LLM10]. This is a geometric non-linear tracking control that achieves almost global asymptotic stability of the tracking error.

This control considers a model for the quadrotor based on the special Euclidean group $SE(3)$. The main advantage of this is that every configuration has only one representation, and also that it avoids the singularities that can arise when using Euler angles. The control inputs considered by the control are the total thrust f and the total moment M of the quadrotor. These can be directly mapped to the velocities of each rotor.

As shown in Figure 5.1, given a desired trajectory and a desired angle yaw ψ, the controller computes in cascade: first, the total thrust f to achieve asymptotic trajectory tracking, and, then, the total moment M in order to achieve zero error on attitude tracking.

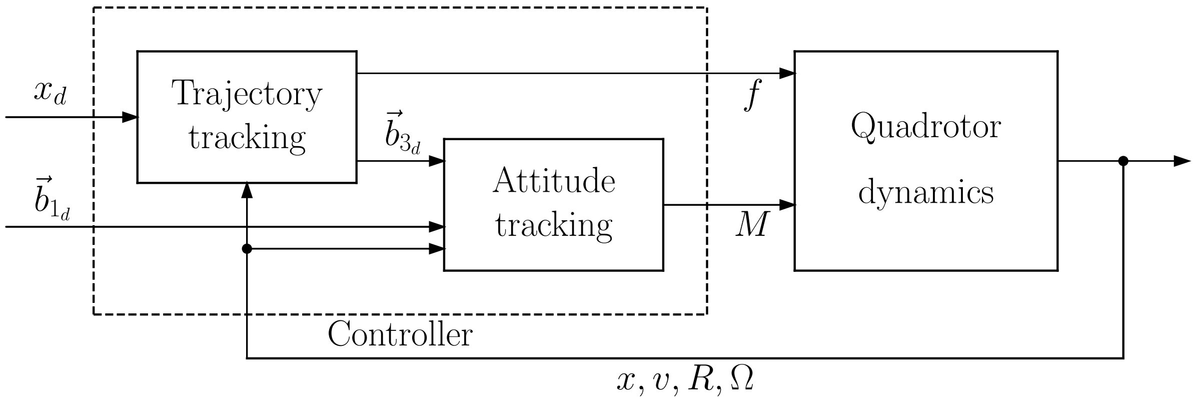


Figure 5.1: Position controller structure from [LLM10].

The control is tuned by four gain vectors: $\boldsymbol{k}_x$, $\boldsymbol{k}_v$, $\boldsymbol{k}_R$, $\boldsymbol{k}_\Omega \in \mathbb{R}^3$. $\boldsymbol{k}_x$ and $\boldsymbol{k}_v$ are associated with the position and velocity errors, respectively, and $\boldsymbol{k}_R$, $\boldsymbol{k}_\Omega$ to the attitude and angular velocity errors, respectively.

The `rotorS` package contains preset values of the four gains for different models of quadrotors. Nevertheless, in our simulations we got to adjust the gains corresponding to the z axis in order to achieve good performance, since we observed an undesired motion in the altitude of the drones when using the preset gains.

5.1.1 Simulation performance

Here we present the simulation results for a formation control experiment driven in `Gazebo`. We use the control scheme proposed in Chapter 4, i.e., a Model Predictive Control scheme using as reference the local consensus error at each iteration; both average hard and soft constraints are used for obstacle avoidance, all of this along with a distributed control architecture. The target formation is given by Figure 4.8 and a fixed column is located at $(0.5, 0.5)$. As in `Python` simulations of Chapter 4.4, a fully connected communication graph is considered. Since this is the same experimental setup as in Chapter 4, it is expected to get similar results to those of the `Python` implementation.

At each iteration k, the controller computes the linear velocities $v_x(k)$, $v_y(k)$ and $v_z(k)$, that are then integrated into positions by

$$\begin{aligned} x(k+1) &= x(k) + Tv_x(k), \\ y(k+1) &= y(k) + Tv_y(k), \\ z(k+1) &= z(k) + Tv_z(k), \end{aligned}$$

where $T = 0.01s$ is the sampling period. Finally, the position $(x(k+1), y(k+1), z(k+1))$ is passed as a reference to the position controller from the `rotorS` package.

Formation experiment without connectivity constraints

Simulation parameters: The prediction horizon H_p is set to 15 and the control horizon H_u is equal to 10. The gain for the penalty term k_d is equal to 10.0 and the control inputs must lie within the $(-5.0, 5.0)$ interval. These parameters are the same as the ones used for the `Python` simulations.

Figure 5.2 shows some snapshots of the evolution of the quadrotors during the experiment. Initially, the quadrotors form a unitary square and they reach the desired formation of Figure 4.8. The complete simulation can be seen in the video `https://youtu.be/PTsa7ThsWSs`.

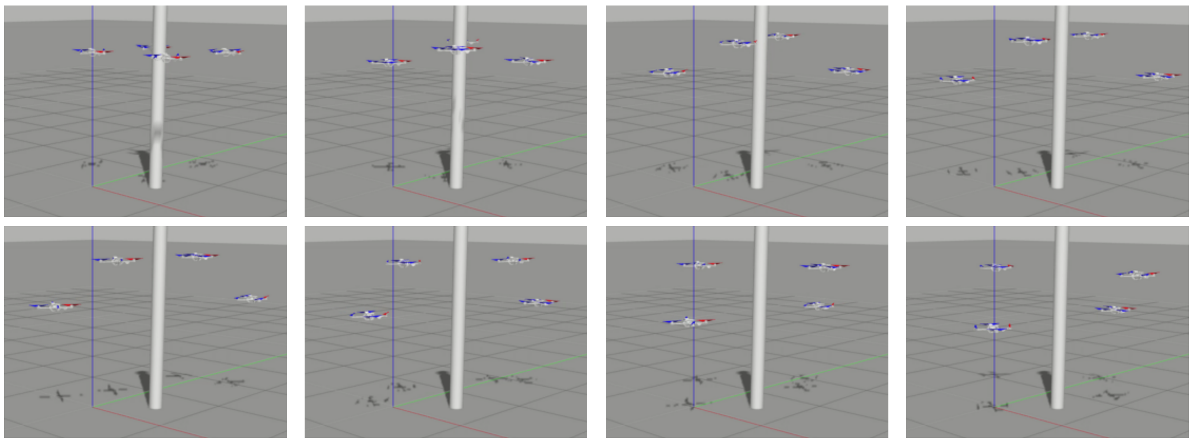
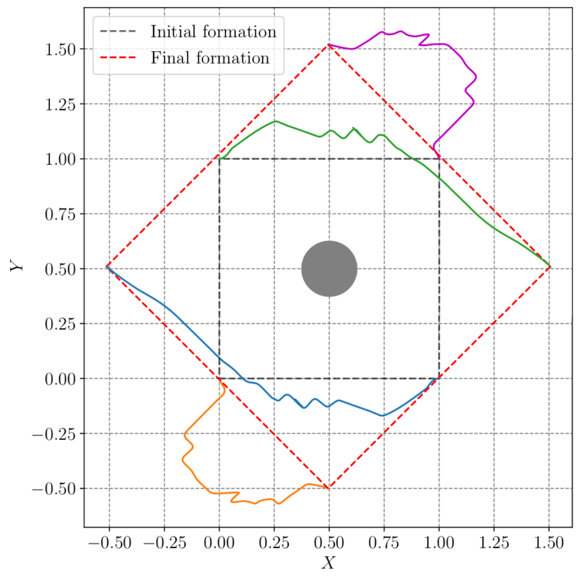


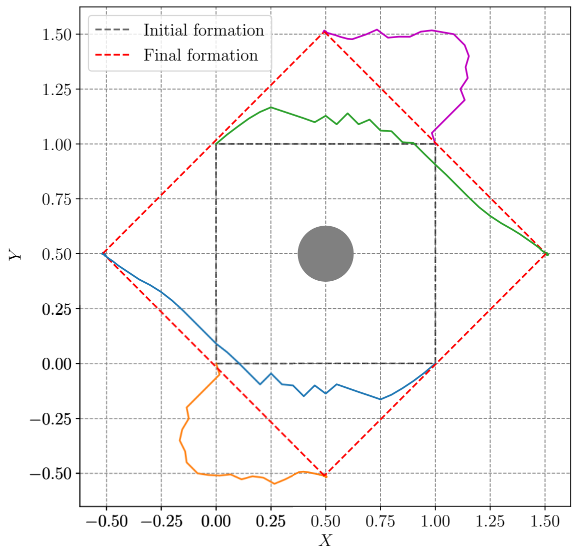
Figure 5.2: Evolution of the quadrotors for a formation experiment in `Gazebo`.

The generated paths can be seen in Figure 5.3, along with the paths generated by the

Python simulations of Chapter 4, for comparison. We can see that the paths in both simulations are very similar.



(a) Gazebo



(b) Python

Figure 5.3: Comparison between the paths generated by Gazebo and Python simulations for the formation experiment in Figure 5.2.

The main difference between the Gazebo and Python simulations is the time required to reach the target formation. The simulations in Python required about 50 iterations to achieve consensus between the virtual agents and, therefore, the desired formation, while in Gazebo the same task took around 300 iterations. This is reasonable because, while in Python the model is assumed to be perfect and the computed velocities are perfectly executed, in Gazebo more physics factors are considered. Also, the low level control can be the cause of the slower performance.

Figure 5.4 shows the distance between the agents and the distance to the fixed column during the Gazebo simulation. We can see that two pairs of agents get close to the security distance but hardly violate the limit. We also observe some oscillations as the agents approach the security distance between them or with respect to the fixed obstacle.

Finally, Figure 5.5 shows the control computed by the MPC. Saturation can be observed at the predefined limits, altogether with some oscillations due to the evasion maneuvers.

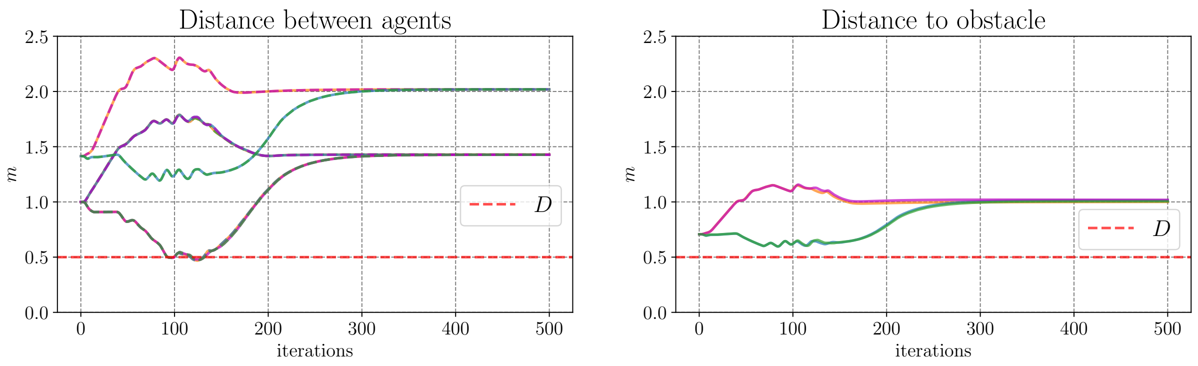


Figure 5.4: Distance between agents and to the obstacle for the distributed formation control of Figure 5.2.

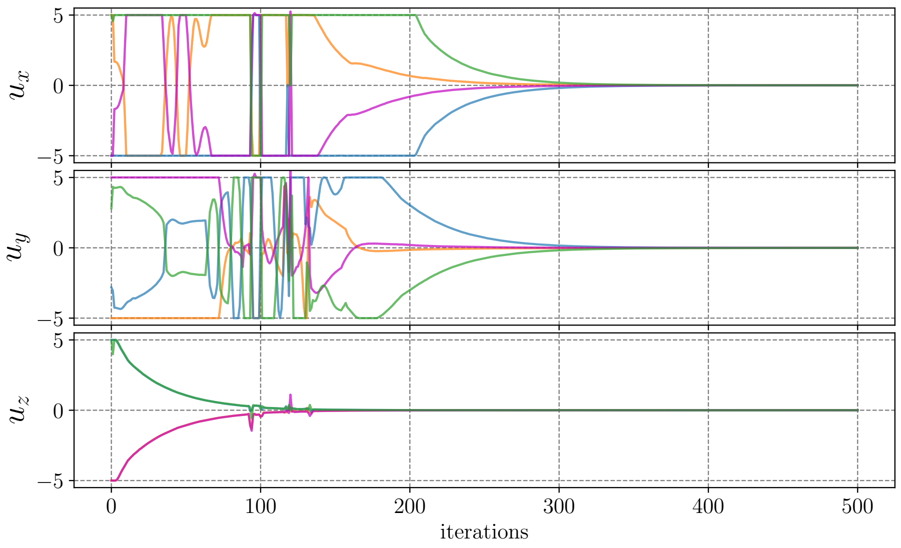


Figure 5.5: Computed controls for the formation experiment of Figure 5.2.

Formation experiment with connectivity constraints

Simulation parameters: In this simulation case, the parameters are the same as in the last experiment without connectivity constraints, adding the parameter $k_c = 1.0$ for the connectivity soft constraints.

Some snapshots of the evolution of the quadrotors during the formation experiment with connectivity constraints can be seen in Figure 5.6. In contrast with the previous experiment without connectivity constraints, where the whole maneuver occurs in a very symmetric way, here we observe that two agents have to wait for the others to move, and then they complete the formation. This can be seen in more detail in the video `https://youtu.be/GozRDSlKMX0`.

The two agents at the bottom cannot separate as much as they did in the exper-

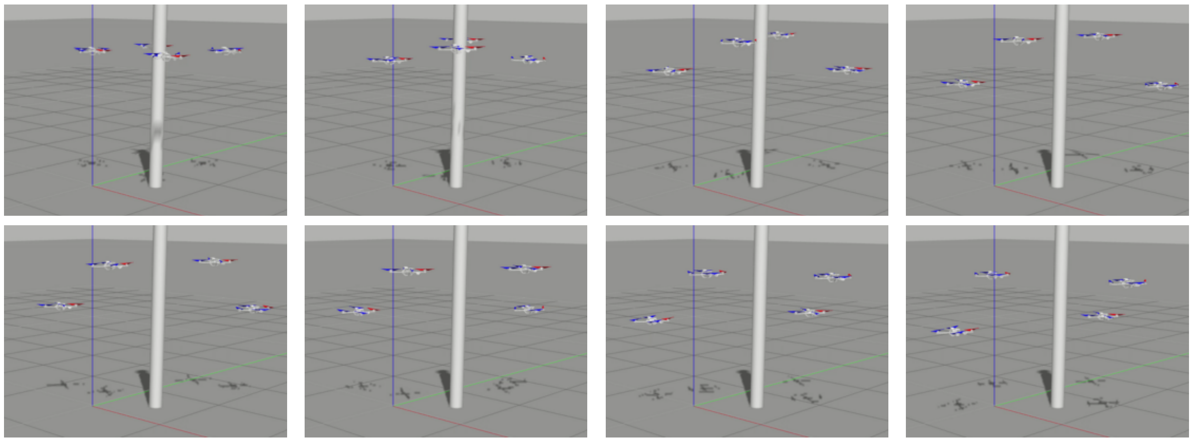


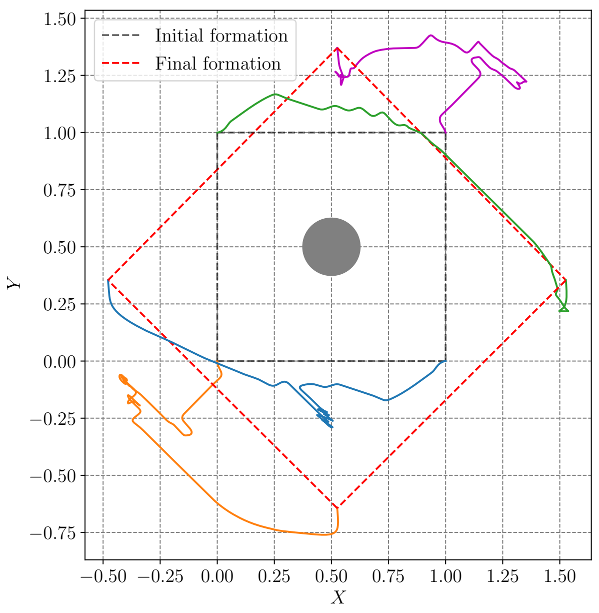
Figure 5.6: Evolution of the quadrotors for a formation experiment in `Gazebo` with connectivity constraints.

iment without connectivity constraints. But in contrast with the `Python` simulation, where the agents behave symmetrically, the simulations in `Gazebo` result in two of the agents waiting for the others to move, avoiding this way to violate the connectivity constraints. A comparison between the paths generated by `Gazebo` and `Python` can be seen in Figure 5.7. In this case, the blue and orange agents wait for the others to move. The generated paths for the other two agents, green and purple, are similar in both simulations.

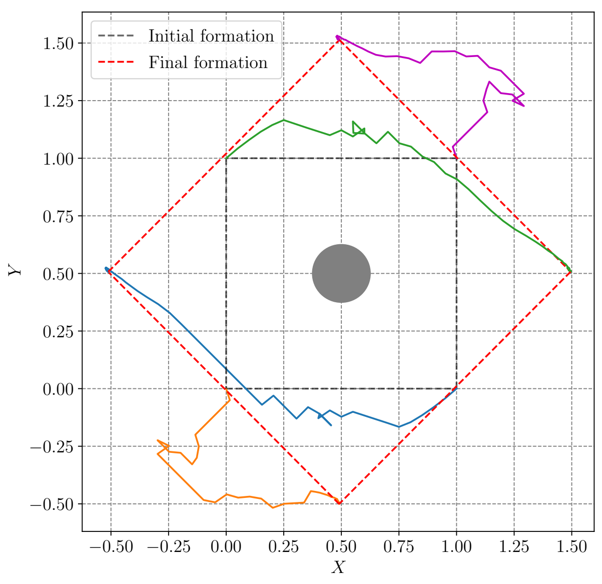
Due to the connectivity constraints and given that two agents have to wait for the others to move, the achievement of the desired formation took about 100 more iterations than for the previous experiment. Figure 5.8 shows the distance between the agents and to the fixed obstacle. We observe that the distance between the orange and purple agents approaches but do not trespass the connectivity limit E. Nevertheless, at some point the distance between the green and purple agents violates the evasion limit D.

Figure 5.9 shows the computed controls for this formation experiment with connectivity constraints. Due to these constraints, more oscillations are observed but the controls stay within the predefined limits.

As we saw in Figure 5.8, the distance between the agents trespassed the security distance D, but this can be avoided by increasing the gain k_d of the penalty evasion term. Figure 5.10 shows the distance between the agents and to the obstacle when we increase the evasion penalty parameter to $k_d = 19$. The inferior limit D is no more trespassed but the connectivity limit E is violated at some iterations. Also, we observe that the consensus takes about 500 iterations, 100 more than for the experiment with $k_d = 10$.

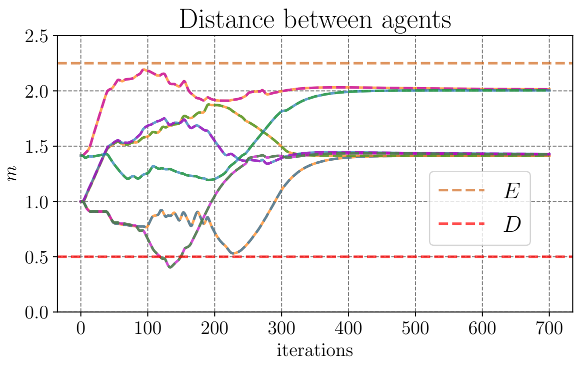


(a) Gazebo



(b) Python

Figure 5.7: Comparison between the paths generated by `Gazebo` and `Python` simulations for the formation experiment in Figure 5.6.



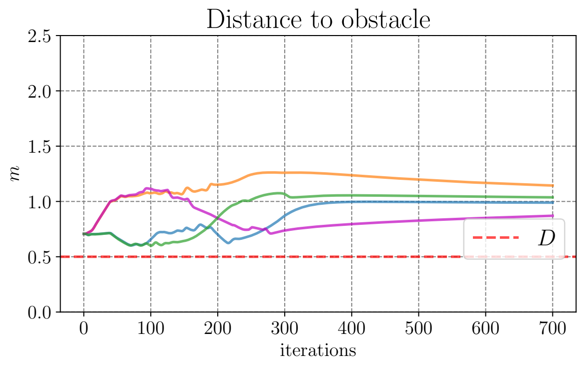


Figure 5.8: Distance between agents and to the obstacle for the distributed formation control of Figure 5.6.

The paths generated with this increased penalty parameter can be seen in Figure 5.11. In this case, the orange and blue agents wait for the green and purple ones to move, and due to the increased penalty parameter k_d, the generated trajectories are more complicated than for the previous cases. This is also reflected in the computed controls in Figure 5.12.

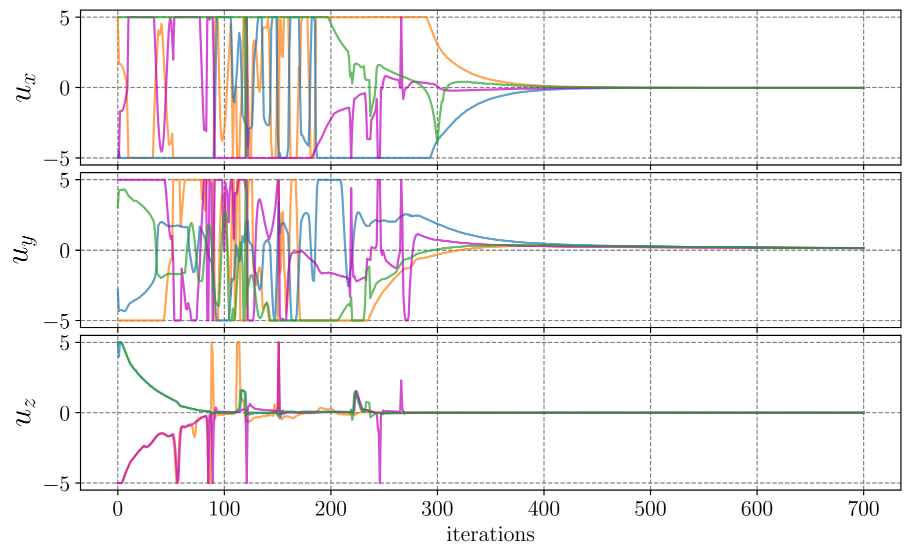


Figure 5.9: Computed controls for the formation experiment of Figure 5.6.

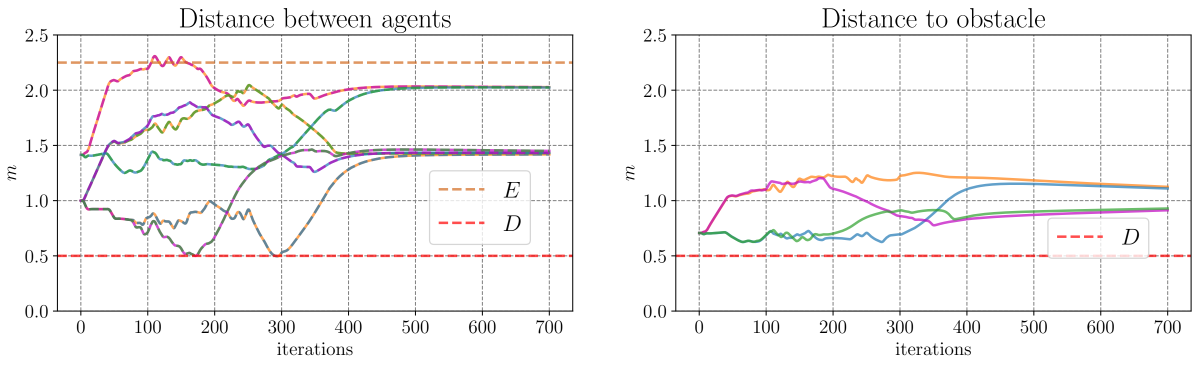


Figure 5.10: Distance between agents and to the obstacle for the distributed formation control with increased penalty parameter $k_d = 19$.

5.2 Experiments with real Bebop 2.0 quadrotors

We ran some formation experiments with *Bebop 2.0* quadrotors from *Parrot*® as the one shown in Figure 5.13.

Experiments setup

All the experiments have been run in a motion capture environment, so that the position of every quadrotor is known along all the experiment. Each quadrotor has some reflecting markers that are tracked by the set of cameras of the motion capture system. These positions are read by each quadrotor, at each iteration, in order to know where its neighbors are.

The control velocities are calculated in a different computer for each quadrotor and

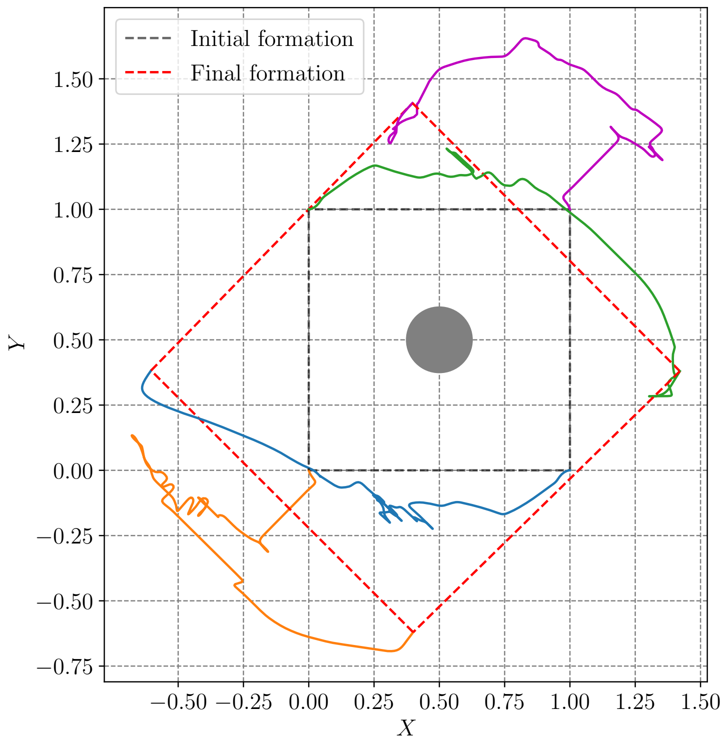


Figure 5.11: Paths generated by a `Gazebo` simulation with an increased penalty parameter $k_d = 19$.

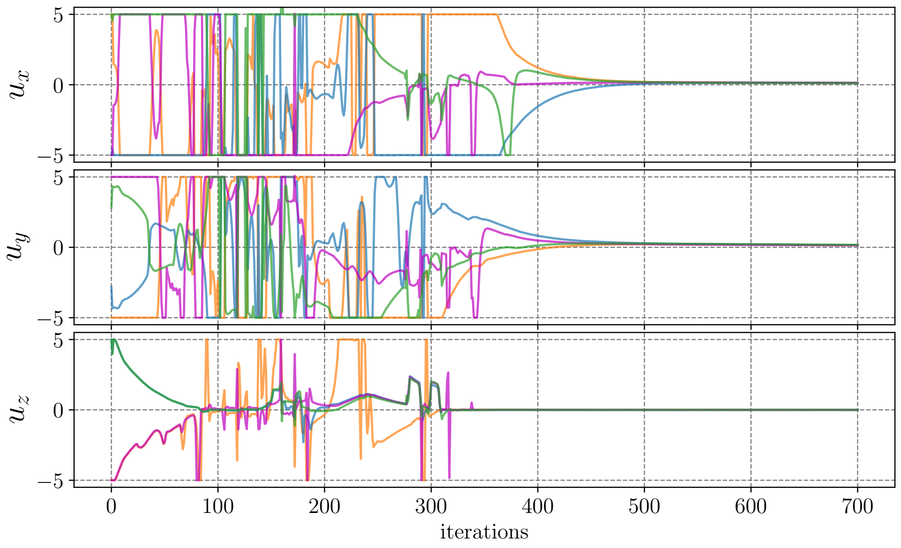


Figure 5.12: Computed controls for the formation experiment of Figure 5.11.

then communicated to the aircraft via WiFi using the `bebop_autonomy` driver [Mon15], so the computations are not performed on board.

A distributed communication architecture is set in `ROS` so that each agent can publish its computed controls and also read the computed controls published by its neighbors.

Figure 5.13: The *Bebop 2* quadrotor from *Parrot*®.

The `bebop_autonomy` driver allows us to directly publish velocity commands to the *Bebop 2.0* quadrotor, that uses its own low level control to transform the velocity commands into velocities for each rotor.

Finally, due to the lack of space, only formation experiments with three quadrotors have been performed. Also, one of the quadrotors stays at hovering along the experiment, acting as a leader, resulting on a communication graph given by

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix},$$

where the last agents send its states to the other two but does not receive any information.

5.2.1 Experiments results

The main issue we faced when implementing our control was the inertia accumulated by the quadcopter when moving at more than $0.05 m/s$, and the inability of the low level control to overcome it; this results in the quadcopter not changing its direction adequately or stopping, even when the computed and communicated controls were demanding it.

To overcome this, we set the limits of the control inputs in -0.03 and 0.03 meters per second, resulting on the quadcopters moving slower but accumulating less inertia. Nevertheless, as we can see in Figure 5.12 or in Figure 5.9, the proposed control always saturate the control inputs at the first iterations (due to the use of the local consensus error as reference that is large at the beginning), causing a lot of accumulated inertia when we ran experiments with the *Bebop 2.0* quadrotors.

Given that decreasing more the limits of the control inputs will lead the quadrotors in not moving at all, we decided to significantly increase the parameter γ_i of the cost function (4.22) that penalizes the size of the control inputs. With a value of $\gamma_i \in [750, 1000]$ the control inputs saturation significantly decreased, resulting on less problems associated with the quadrotor inertia.

A large penalty of the control inputs leads to a slower and more controlled movement, but avoiding to reach the formation at the last iterations. Considering that increasing the penalty of the consensus local error β_i of equation (4.22) will lead again to saturated control inputs, and given that these saturation problems arise mainly during the first iterations, we introduced a dynamic value for β_i, depending on the consensus error e_i at the present iteration, given by

$$\beta_i(e_i) = \begin{cases} \beta_{\max} - \dfrac{\beta_{\max} - \beta_{\min}}{e_r} e_i & e_i < e_r, \\ \beta_{\min} & e_i \geq e_r, \end{cases} \tag{5.1}$$

where $\beta_{\min}$ and $\beta_{\max}$ are the bounds for the parameter β_i and e_r is a predefined level of error.

The dynamic value of β_i given by (5.1) produces low values when the consensus error is high, i.e., when the quadcopters are far from the target formation. In this case, the computed controls will focus in a slower and controlled movement. Then, when the error starts to decrease and the computed controls become lower, the value of β_i becomes higher in order to penalize more the consensus error, resulting in higher computed inputs that can lead to the actual achievement of the target formation.

Finally, due to the computation time required to solve the optimization problem in each computer and in order to make the sampling period uniform for all the agents, the parameter T was set to 0.1 seconds.

Here we present the results for two experiments: a formation control experiment for three quadcopters without obstacle, and the same experiment but considering a single column obstacle. The dimensions of the space where the quadcopters can navigate was of $4.75m \times 3m$ and the limits of this space were introduced as hard constraints for the optimization problem. Due to the lack of physical space, we did not perform experiments with connectivity constraints. Also, to ensure that the formation was achieved within the available space, one of the three quadcopters remains in hovering state along all the experiments, i.e., it is not controlled and is not meant to move, but that agent acts as

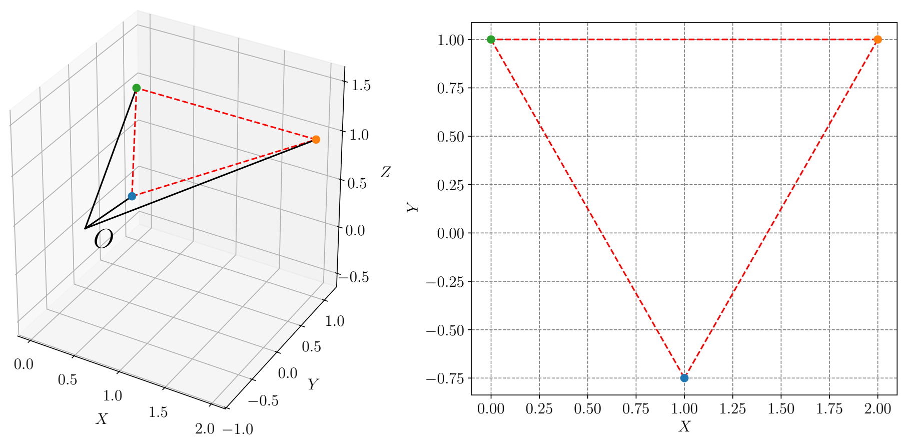


Figure 5.14: Target formation for experiments with three *Bebop 2.0* quadrotors.

an anchor for the formation. Figure 5.14 shows the target formation used along the experiments.

Both experiments were run with the same parameters: The prediction horizon H_p is 15 and the control horizon H_u is equal to 10. The gain for the evasion penalty term is $k_d = 1$. The control penalty parameter is $\gamma_i = 1000$ and the consensus error penalty parameter is set to be within the $[20, 100]$ interval, with a level of error $e_r = 3$.

Formation experiment without obstacles

Figure 5.19 shows some snapshots of the evolution of a formation experiment with 3 *Bebop 2.0* quadrotors as seen from the top. Two quadrotors start at the right and the third one at the lower left. At the end of the experiment the three quadrotors reach the desired configuration given by Figure 5.14. A video of this experiment can be consulted in `https://youtu.be/1vbekKOyr7U`.

The generated paths for the three quadrotors, along with the initial and final formation, can be seen in Figure 5.16. The green agent moves in a practically straight line until it reaches the vicinity of its target position. The blue agent tries to minimize its error with respect to the green agent and that is why its path approaches to $(0, -1)$; but when the green agent is closer, the blue agent gets also closer to its target position. Some oscillations can be observed near the target position as the agents try to converge to the desired formation. Finally, even though the orange agent is not controlled and is meant to stay at the same position, we can see some displacement due both to the inner hovering control and to the external disturbance generated by the turbulence of the other quadrotors. Nevertheless, the controlled agents adjust to this movement, reaching

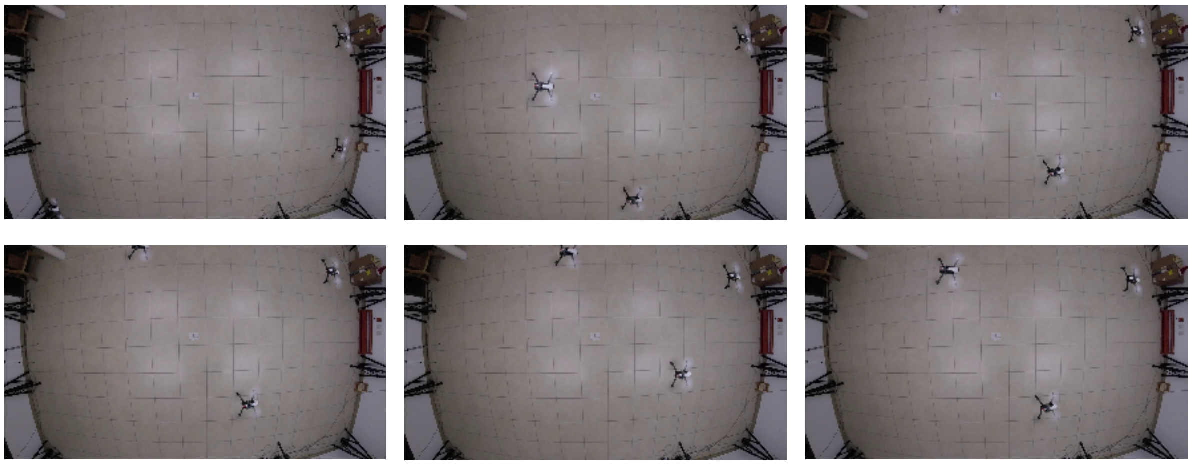


Figure 5.15: Evolution of a formation experiment with 3 real *Bebop 2.0* quadrotors.

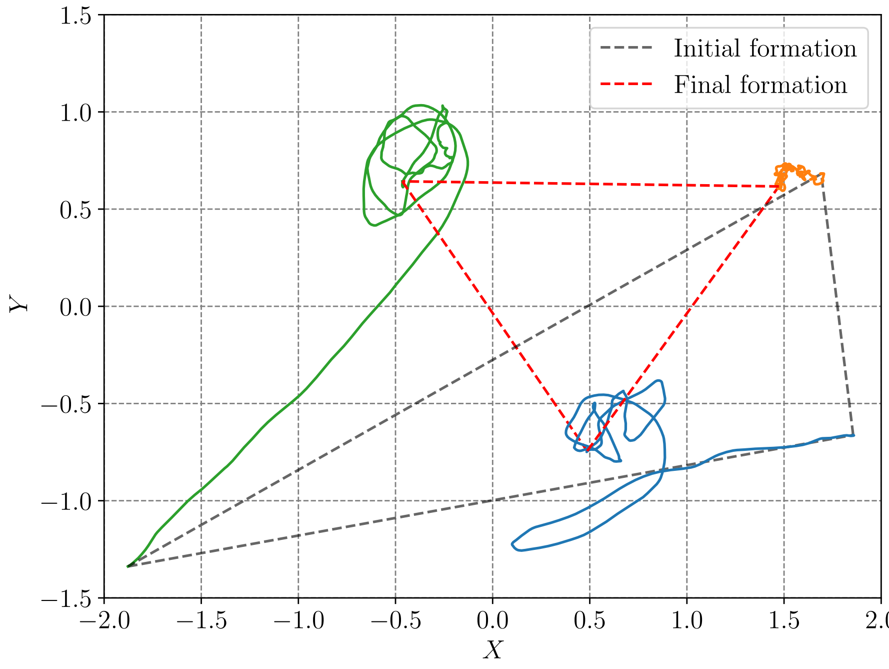


Figure 5.16: Paths generated by the formation experiment of Figure 5.15.

the target position.

As can be seen in Figure 5.16, there is no risk of collision along the experiment. Therefore in Figure 5.17 we observe that the distance between agents does not approach a lot to the security distance.

Finally, the computed control inputs appear in Figure 5.18. We observe very little saturation, because of the very high value of the control penalty parameter and the adaptive value of consensus error penalty parameter. The control velocities oscillate but decreasing in amplitude, allowing the system to converge. The computed control

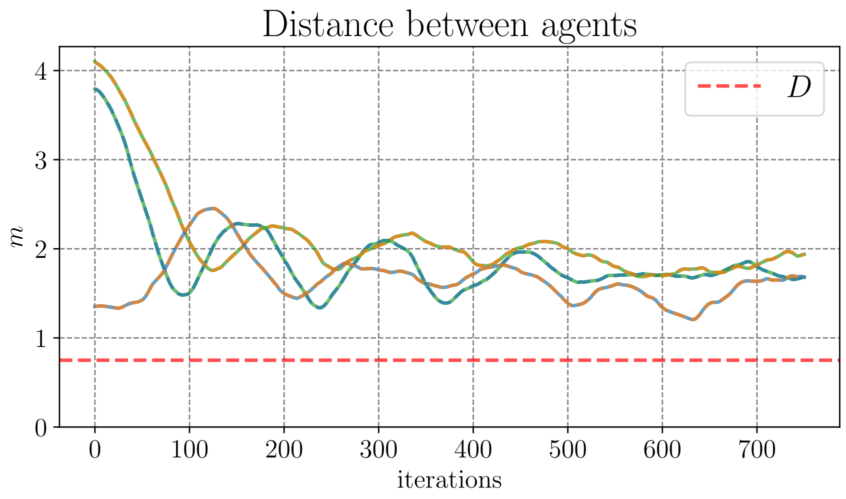


Figure 5.17: Distance between agents in the formation experiment of Figure 5.16.

input of the orange agent remains in 0 as it is not controlled.

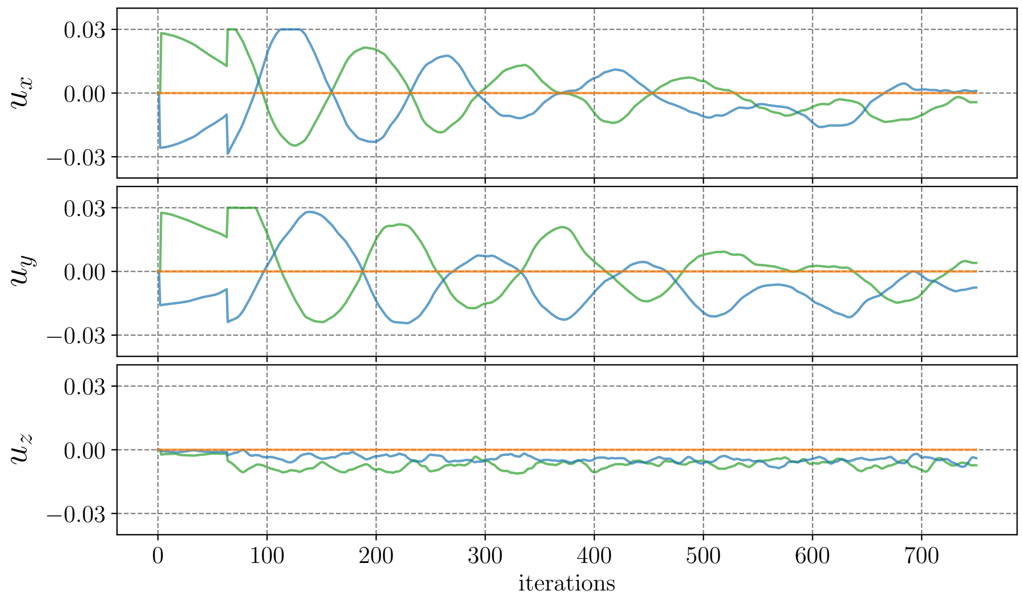


Figure 5.18: Computed control inputs for the formation experiment of Figure 5.16.

Formation experiment with a column obstacle

Figure 5.19 shows some snapshots of the evolution of a formation experiment with 3 *Bebop 2.0* quadrotors with an obstacle as seen from the top. The target formation is the same as in the previous experiment, given by Figure 5.14. The initial configuration is very close to the previous one also, with two quadrotors aligned at the right and the third one at the bottom left corner. The obstacle corresponds to the white column observed in the scene and is located at the right of the agent in the left, with the purpose of avoiding an straight trajectory for this agent. This experiment can be seen in the video `https://youtu.be/aDLBDzEqpB0`.

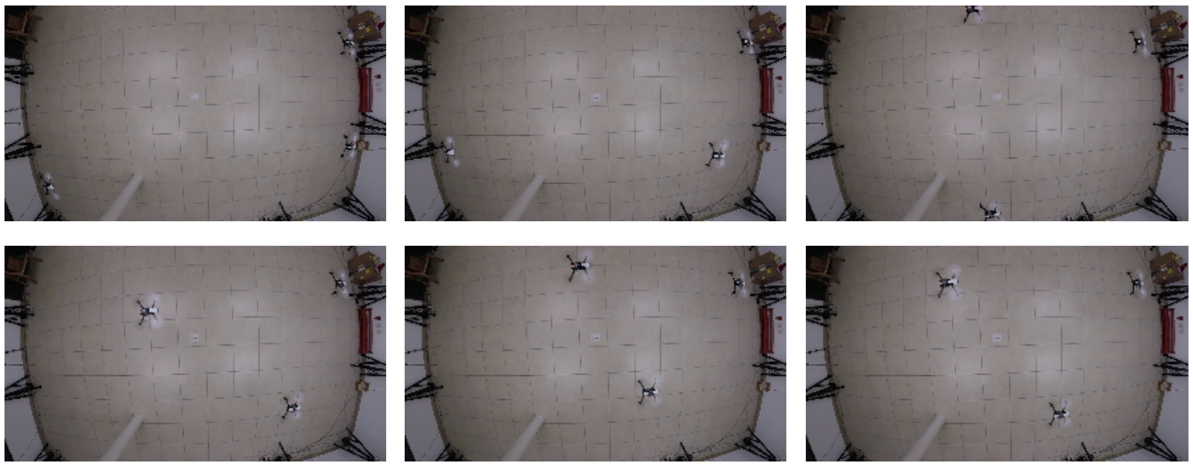


Figure 5.19: Evolution of a formation experiment with 3 real *Bebop 2.0* quadrotors with an obstacle.

As can be seen in Figure 5.20, the green agent does not follow anymore a straight trajectory, as it has to evade the obstacle. The trajectory of the blue agent is very similar to the one of the previous experiment, trying to decrease the error relative to green agent first, and when this one is closer, converging to the desired formation. In this experiment, the movement of the noncontrolled orange agent has a lower amplitude than in the previous experiment, but it can still be appreciated.

The distance between the agents and to the obstacle along the experiment can be seen in Figure 5.21. At the beginning, the distance between the green agent and the obstacle is close the security distance, and following a straight trajectory as in Figure 5.16 would have lead in trespassing this limit, that is why the green agent chooses a curved trajectory farther from the obstacle. The obstacle also stops the blue agent movement to the left around the iterations 100 and 150. It is important to remark that, even though the gain for the evasion penalty term $k_d = 1$ is much lower than the value of the control penalty parameter $\gamma_i = 1000$ or the consensus error penalty parameter whose lower value is 20, it is enough to avoid collisions. This is because of the form of the potential field of the evasion penalty term, that increases infinitely as the distance to an obstacle or between agents approaches the security distance. This small value for the penalty term gain also permits the control to focus on the formation task when there are no risk of collision.

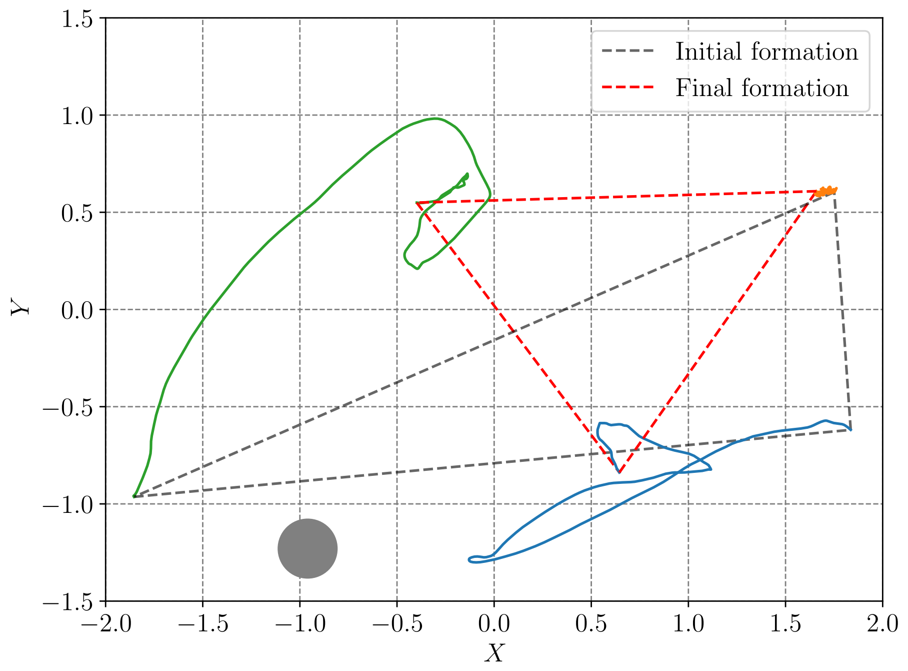


Figure 5.20: Paths generated by the formation experiment of Figure 5.19.

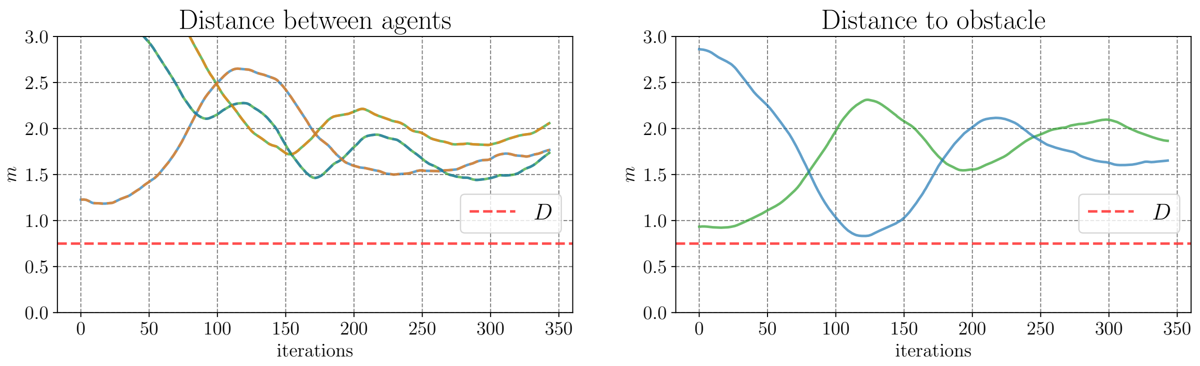


Figure 5.21: Distance between agents and to the obstacle in the formation experiment of Figure 5.20.

Finally, Figure 5.22 shows the computed controls. Some saturation can be observed at the first 100 iterations, and then again between iterations 100 and 150 for the blue agent when there is a need to evade the obstacle and to adapt to the movement of the green agent. Again, the controls oscillate but with decreased amplitude, leading to a safer convergence.

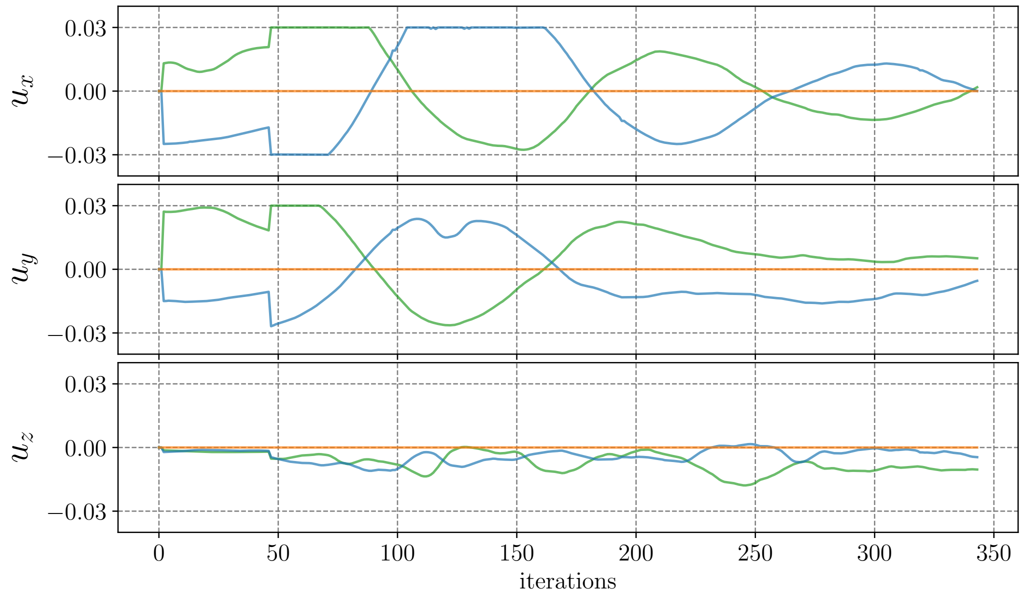


Figure 5.22: Computed control inputs for the formation experiment of Figure 5.20.

Chapter 6

Conclusions and future work

This thesis has addressed the problem of controlling a group of quadrotors in a distributed way to achieve desired formations with guarantees of collision avoidance between the agents themselves and obstacles in the environment, and connectivity maintenance. A Model Predictive Control strategy has been proposed for the formation control of quadrotors, showing a good performance both in realistic simulations and in real experiments with quadrotors. The main advantage of this control strategy is its capability and flexibility to consider constraints relative not only to the evasion and connectivity between agents, but also to the elements of the environment, such as obstacles and dimensions of the space where the quadrotors can move. Moreover, the predictive component of this control technique allows the agents to anticipate (in a time window) the realization of the required actions to guarantee collision avoidance and connectivity maintenance.

Two ways of reaching consensus of virtual agents, achieving this way the desired formation have been explored, the first one generating trajectories for consensus and then tracking them using MPC, and the second one introducing directly the local consensus error at each iteration to the MPC cost function. Regarding obstacle avoidance and connectivity maintenance, the main question was choosing between adding hard and/or soft constraints to the MPC optimization problem. Finally, regarding control architecture, both decentralized and distributed control were studied.

Based on simulation results, the best formation control strategy resulted on using the local consensus error directly in the MPC cost function, and including both hard and soft constraints for obstacle avoidance and connectivity maintenance; all of this inside a distributed control architecture. This final strategy showed advantages in the smoothness of the generated paths, the velocity of convergence to the desired configuration, and the effectiveness to avoid obstacles.

The proposed control scheme behaves well also in real experiments, despite the fact of using a very simple first order linear model for the dynamics of the quadrotor. Nevertheless, further work can be done in this aspect in order to decide if a more complex model improves the behavior of the control and whether this improvement has a significant effect on computation time.

Because of its simple model, the proposed formation control strategy can be applied not only to quadrotors, but also to any holonomic robot or even to a non-holonomic robot, provided that there exists a low level control that transforms the computed velocity controls into controls for each particular robot, permitting to apply this control even to heterogeneous systems of multiple agents.

Also, the proposed formation control strategy, can be extended to a leader-followers scheme where, knowing the position but also the future control inputs of a leader, the followers can maintain a given formation while moving along the leader's trajectory. In principle, this can be achieved by defining a particular communication graph, and applying the proposed formation control to the followers, while the leader moves around. We are interested to endow to the whole formation capacities of navigation in unknown environments, however, for that goal, the proposed scheme must be extended to use onboard sensing, since the current results rely on the use of a motion capture system that limits the workspace of the group of quadrotors.

Bibliography

[CBA99] Eduardo F. Camacho and Carlos Bordons-Alba, *Model predictive control*, 1 ed., Advanced Textbooks in Control and Signal Processing, Springer-Verlag London, London, UK, 1999.

[CFZ15] Zhaomeng Cheng, Ming-Can Fan, and Hai-Tao Zhang, *Distributed mpc based consensus for single-integrator multi-agent systems*, ISA Transactions **58** (2015).

[CS18] Chia-Wei Chang and Jaw-Kuen Shiau, *Quadrotor formation strategies based on distributed consensus and model predictive controls*, Applied Sciences **8** (2018), 2246.

[CSdlPL12] Panagiotis D. Christofides, Riccardo Scattolini, David Munoz de la Peña, and Jinfeng Liu, *Distributed model predictive control: A tutorial review and future research directions*, Computers & Chemical Engineering **51** (2012), 21–41.

[CW13] X. Chen and L. Wang, *Cascaded model predictive control of a quadrotor uav*, 2013 Australian Control Conference, 2013, pp. 354–359.

[DLL17] X. Dai, C. Liu, and F. Liu, *Consensus protocol based on distributed model predictive control for first-order multi-agent systems*, 2017 13th IEEE International Conference on Control Automation (ICCA), 2017, pp. 982–987.

[DS18] Loïc Dubois and Satoshi Suzuki, *Formation control of multiple quadcopters using model predictive control*, Advanced Robotics **32** (2018), no. 19, 1037–1046.

[FBAS16] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart, *Robot operating system (ros): The complete reference (volume 1)*, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625, Springer International Publishing, Cham, 2016.

[HT12] Meiyun Y. He and André L. Tits, *Constraint reduction with exact penalization for model-predictive rotorcraft control*, 2012.

[IN17] M. Ille and T. Namerikawa, *Collision avoidance between multi-uav-systems considering formation control using mpc*, 2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), 2017, pp. 651–656.

[JG17] Jingfu Jin and Nicholas R. Gans, *Collision-free formation and heading consensus of nonholonomic robots as a pose regulation problem*, Robotics Auton. Syst. **95** (2017), 25–36.

[KBS16] Mina Kamel, Michael Burri, and Roland Siegwart, *Linear vs nonlinear MPC for trajectory tracking applied to rotary wing micro aerial vehicles*, CoRR **abs/1611.09240** (2016).

[KG17] S. Kumar and R. Gill, *Path following for quadrotors*, 2017 IEEE Conference on Control Technology and Applications (CCTA), 2017, pp. 2075–2081.

[KH04] N. Koenig and A. Howard, *Design and use paradigms for gazebo, an open-source multi-robot simulator*, 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), vol. 3, 2004, pp. 2149–2154 vol.3.

[Kha02] Hassan K Khalil, *Nonlinear systems; 3rd ed.*, Prentice-Hall, Upper Saddle River, NJ, 2002.

[KN15] Yasuhiro Kuriki and Toru Namerikawa, *Formation control with collision avoidance for a multi-uav system using decentralized mpc and consensus-based control*, 2015 European Control Conference, ECC 2015, Institute of Electrical and Electronics Engineers Inc., November 2015, European Control Conference, ECC 2015 ; Conference date: 15-07-2015 Through 17-07-2015, pp. 3079–3084 (English).

[Lee11] Jay H. Lee, *Model predictive control: Review of the three decades of development*, International Journal of Control, Automation and Systems **9** (2011), 415–424.

[LLM10] T. Lee, M. Leok, and N. H. McClamroch, *Geometric tracking control of a quadrotor uav on se(3)*, 49th IEEE Conference on Decision and Control (CDC), 2010, pp. 5420–5425.

[MBM01] V. Mistler, A. Benallegue, and N. K. M'Sirdi, *Linéarisation exacte et découplage entrées-sorties. comparaison entre l'hélicoptère standard et l'hélicoptère 4 rotors*, Proceedings 10th IEEE International Workshop on Robot and Human Interactive Communication. ROMAN 2001 (Cat. No.01TH8591), 2001, pp. 586–593.

[MKC12] R. Mahony, V. Kumar, and P. Corke, *Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor*, IEEE Robotics Automation Magazine **19** (2012), no. 3, 20–32.

[Mon15] Mani Monajjemi, *bebop_autonomy - ros driver for parrot bebop drone (quadrocopter) 1.0 & 2.0*, `http://bebop-autonomy.readthedocs.io/`, 2015, Accessed: 2020-12-05.

[OM04] R. Olfati-Saber and R. M. Murray, *Consensus problems in networks of agents with switching topology and time-delays*, IEEE Transactions on Automatic Control **49** (2004), no. 9, 1520–1533.

[OSFM07] Reza Olfati-Saber, J. Fax, and Richard Murray, *Consensus and cooperation in networked multi-agent systems*, Proceedings of the IEEE **95** (2007), 215–233.

[RSCSC15] Tiago Ribeiro, Andre Scolari Conceicao, Inkyu Sa, and Peter Corke, *Nonlinear model predictive formation control for quadcopters*, IFAC-PapersOnLine **48** (2015), 39–44.

[Sab15] Francesco Sabatino, *Quadrotor control: modeling, nonlinear control design, and simulation*, Master's thesis, KTH Electrical Engineering, 6 2015.

[Sas99] Shankar Sastry, *Nonlinear systems: Analysis, stability, and control*, 1 ed., Interdisciplinary Applied Mathematics 10, Springer-Verlag New York, 1999.

[WG18] Alexander Waldejer and Ashkan Ghodrati, *Distributed model predictive control for a coordinated multi-agent system*, Master's thesis, Chalmers University of Technology, 2018.

[YSN17] Kenta Yamamoto, Kazuma Sekiguchi, and Kenichiro Nonaka, *Experimental verification of formation control by model predictive control considering collision avoidance in three dimensional space with quadcopters*, 2017 11th Asian Control Conference (ASCC), 12 2017, pp. 1602–1607.