FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Drone Flock: formation control of multi-drones

João Costa

Master Degree in Electrical Engineering and Computer Science

Supervisor: António Pedro Aguiar

October 2019

Resumo

Nas últimas décadas tem-se assistido a uma crescente utilização de drones para substituir veículos tripulados em aplicações de alto risco, alto custo ou restringidas a nível de espaço. As suas capacidades levaram ao crescimento constante do número de aplicações e os drones deixaram de ser usados apenas para fins militares. Novos algoritmos para controlo do movimento têm sido desenvolvidos e optimizados para tornar estes sistemas mais independentes e efectivos. No entanto, é importante salientar que este tipo de veículos sub-atuados têm uma dinâmica altamente não-linear e naturalmente instável, o que faz com que sejam problemas especialmente desafiantes do ponto de vista do controlo não-linear. Os avanços tecnológicos mais recentes conduziram a uma nova direção de investigação referente ao controlo de sistemas multi-agente a agirem cooperativamente. Existem inúmeras aplicações onde sistemas adaptáveis de pequenos veículos autónomos são a melhor solução.

Nesta dissertação, é proposto um algoritmo de controlo baseado em funções de Lyapunov. O objetivo principal deste algoritmo denominado *cooperative path following* é capacitar os drones de realizarem a tarefa de seguir o trajeto desejado e em formação de modo a atingirem consenso entre as velocidades. Para isso, numa primeira fase começa-se por um controlador mais simples, a duas dimensões, de *cooperative path following* implementado num sistema de robôs tipo monociclo para introduzir as ferramentas a usar, alguns fundamentos teóricos e, principalmente, para mais facilmente abordar o problema do consenso.

O controlador começa por lidar com o problema do seguimento de trajetória fazendo com que o drone siga um ponto virtual desejado nesse trajeto, parameterizado por uma variavel de controlo. Em seguida, a velocidade desejada para a variável de parametrização de cada drone é ajustada de modo a manter todo os drones em formação, o que acontece quando todas as variáveis de parametrização estão em consenso e evoluem à mesma velocidade. Para verificar as soluções obtidas, simulações são efetuadas numa ferramenta do Matlab, desenvolvida especificamente para a projecção de controladores e simulação de sistemas, chamada Virtual Arena. Simulações de robots e de drones em formação são feitas em duas configurações da rede de comunicações diferentes. Os resultados mostram que as leis de controlo não-linear baseadas em funções de Lyapunov funcionam devidamente tanto no seguimento de trajetória, como no consenso dos drones.

Abstract

For decades, humans have been using drones to replace human-crewed vehicles in high risk, high cost, or space-constrained applications. Their capabilities lead to a constantly growing range of applications and drones are no longer military. To account with the increasing of applications and their complexity, motion control algorithms have been developed and optimized to make these systems as independent and effective as possible. This kind of under-actuated vehicle has highly non-linear and naturally unstable dynamics, which make their non-linear control especially challenging. The recent technological advances have incited developers towards cooperative motion control of multiple agent systems. There are several applications where adaptable flocks of small, autonomous vehicles seem to be the best solution.

In this dissertation, a Lyapunov-based control algorithm is proposed and analized. The main goal of this algorithm is to implement a cooperative path following controller that makes the drones follow a desired path and the flock achieves consensus. To this end, we start first with a simpler, 2 dimentional, cooperative path following controller that is developed to a unicycle type robot flock to get introduced to the tools, the theoretical preliminaries, and, firstly, easily approach the consensus problem.

This controller, firstly, deals with the path following problem by making drones follow a desired virtual point parametrized by a path parametric variable. Secondly, the desired parametric variable speed of each drone is adjusted in order to keep them in formation, which happens when these parametric variables are in consensus and evolving at the same desired speed.

To verify the obtained solutions, simulations are done using an object-oriented Matlab toolkit for control design and system simulation, called Virtual Arena. Simulations for both unicycle and drone flocks are performed with several distinct communication network configurations. The results show that the nonlinear Lyapunov based control laws proposed work properly on both path following and consensus problems.

Contents

List of Figures

Chapter 1

Introduction

1.1 Context

Since 1783, humans have been trying overcome their physical limitations using the technology to reach the sky, initially, through the use of aerostats and later with the first dirigible balloon.

In the 1890s the first fixed-wing aircraft is developed and, at the beginning of the twentieth century, the first human-crewed flights started emerging, in both fixed and rotary wing aircraft.

During this century of flights, its development was focused on aircraft piloted by humans and, as opposed to what happens right now in the twentieth-one century, autonomous or remote controlled flights were seen as something surreal and taken as a luxury or an exception.

The need for the pilot being in the control of the aircraft requires another crew member to execute specific tasks, as to control cameras or to handle guns.

Replace human-crewed vehicles due to its high risk, high costs of utilization and maintenance, or even the inability to use them in specific applications, with autonomous, non-crewed, smaller ones has become a growing need.

In the 1910s, the first autonomous stabilization systems emerged and, even before the First World War, test flights were made by the first aircraft without humans on board.

To those aircraft, without a crew or passengers on board, we call drones, also known as UAV from "Unmanned Aerial Vehicle". These are equipped with radio communication devices, and its control can be done inside (fully autonomously) or remotely by a radio controller or a ground station, for example.

Multirotor Drones have a particular interest thanks to its ease of implementation, and its acquisition and maintenance costs are relatively low.

Given its low cost, no need for a pilot on board to control it, ease of use and its vast versatility, UAV applications have been increasing drastically. These are the latest technology that aims to revolutionize people's daily lives. Its uses are no longer military only, such as surveillance and air strikes, and countless civil and commercial applications emerged. Aerial mapping, search and rescue, precision agriculture, sensors application, inspection of buildings and hard to reach places, and others, are some examples. In the last few years, drones generated interest by showing their surprising capability in monitoring applications due to their portability, rapid implementation, and accuracy in recognition, as well as the low environmental impact and risk to the human operators when compared with crewed aircraft. We can not forget the recreational activities that have been growing exponentially.

1.2 Motivation

In nature, it is possible to observe that, for example, bird flocks or bee swarms, that is, a certain number of small entities of low intelligence are capable of performing spectacular collective movements, in such a way that they catch the attention of researchers of several areas.

These collective movements have enormous potential in military, industrial, and civilian applications.

The development of small multirotor drones started with the control of its stability and maneuverability. Then came the creation of autonomous systems capable of performing tasks and following paths with little human intervention. With the latest advances in technology and the increasing market offer of this kind of vehicle, the development is now pointing toward cooperative multi-UAV systems.

Unlike single-agent systems, a decentralized multi-agent system has several advantages, such as:

- Parallel operation - efficiency increasing and execution time reduction in functions as surveillance or aerial mapping.
- The removal or addition of an agent should not affect the system.
- Each agent can organize itself in order to fulfill its task, without needing a central unit to control the system.

Taking into account the above, networks of UAV are the trend of evolution. These systems are particularly important for missions whose execution time is critical, the coverage area has to be maximized, or there are limitations on the battery capacity when using only one agent.

However, a network of UAVs is not just the sum of several UAVs. More agents add complexity to the system, but, on the other hand, the redundancy of the network generates more reliability and variety in the obtained data.

As a major drawback, this powerful technology continues to depend mostly on trained human operators for remote control. This drawback limits the use of drones severely in complex operating scenarios and makes robust, reliable, and efficient coordination in motion a fundamental step in the implementation of autonomous systems.

Therefore, motion control algorithms should be developed and optimized to make these systems as independent and effective as possible, and reliable communication systems, capable of operating under extreme conditions and ensure up-to-date information at the lowest cost possible, too.

1.3 Goals

In this work, we will focus on methods of control of unmanned aerial vehicles, more concretely in multi-drone scenarios operating cooperatively.

For this reason, it will be studied, designed, and implemented advanced decentralized nonlinear controllers for multiple vehicle coordination. The control algorithms must explicitly take into account the dynamics of the vehicles and the limitations imposed by the dynamic environment that surrounds them.

1.4 Document Structure

Chapter 1 focuses on the context, motivation and goals of the proposed work. Chapter 2 presents the state of the art (Section 2.1), followed by some theorectical preliminaries (Section 2.2) usefull for a better understanding of this work and, in the end, a brief introduction and explanation of the tool and its use in this work (Section 2.3). In Chapter 3 the systems' models are defined, and in Chapter 4 both path following and cooperative path following controllers are derived. Simulation results are presented in Chapter 5 and Chapter 6 concludes this work and addresses future relevant work in this subject.

Chapter 2

State of the Art, Fundamentals and Tools

In this chapter will be revised the state of the art and some essential matters in order to help us follow and understand further work more easily.

Since the goal is to project motion controllers to keep a flock of drones in formation, and this control must be done in a decentralized way, graph theory will be explained in subchapter 2.2.

In section 2.3, it will be explained what a state space representation is, and a simple example will be presented.

The following sections clarifies how Lyapunov's Stability Theorem works and how it is useful when we want to find the control laws for our system.

The tool used in this work will be also presented at the end of this chapter.

2.1 State of the Art

2.1.1 Drones

2.1.1.1 Drone tipologies

To better understand how drones work, it was necessary to study the kind of aerial vehicles existents, their advantages and disadvantages, and how the stabilization, maneuverability, and low-level control works on them.

That study took to a division of aerial vehicles in two main categories:

- Fixed-Wing Aircraft
- Rotary-Wing Aircraft

Inside rotary-wing aircraft, there are two subcategories that worth mentioning: helicopters and multicopters. Both are quite similar in many flight and control aspects, but when rotary-wing aircraft is mentioned, it applies mostly to multicopters.

In fixed-wing aircrafts lift is generated by the wing, like commercial airplanes, e.g., and offer longer flight times than rotary wings aircraft. However, they need long distances to take off and land, and a minimum linear speed to keep themselves in the air.

On the other hand, fixed-wing aircraft have the ability to take-off and land vertically, and, once they do not need minimum speeds to generate lift, they can hover in one place.

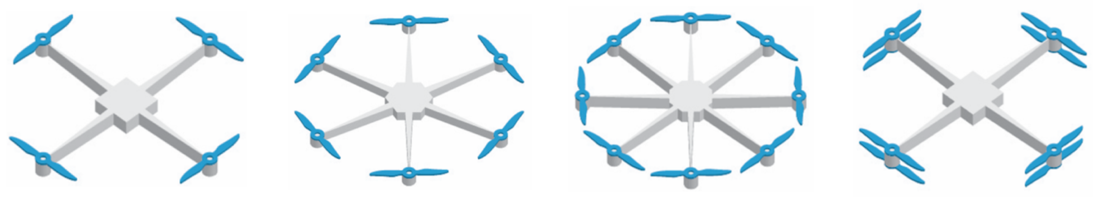
Another factor that differentiates both categories is its maneuverability.

Fixed-wing aircraft require, typically, a bigger turning radius when compared to rotary-wings. These can move freely in all axis from the point where they are.

The market supply of both configurations, nowadays, differs significantly. There is a higher offer in platforms, hardware, and control software for multicopters than for fixed-wing aircraft.

Together with its easy of use and smaller dimensions, those were decisive factors when choosing multicopters over fixed-wings as the platform to implement the path following controller designed in this dissertation.

2.1.1.2 Multicopter configurations and components



(a) Quadcopter (b) Hexacopter (c) Octocopter (d) Coaxial Octocopter

Figure 2.1: Multicopter configurations

Multicopters can be classified by the number of rotors they have, normally one rotor in each arm. Quadcopters, hexacopters, and octocopters are the most common configurations, figures 2.1 a), b) and c), respectively. Depending on the application requirements, bigger multicopters or co-axial configurations can be built. For example, if we need enough thrust to lift a heavy load but have space constraints, a coaxial quadcopter as in figure 2.1 d), or a coaxial hexacopter, are one of the ways to go.

This work will focus on quadcopters but, with minor adjustments, the controller may work for every multicopter configuration.

The components presented next are specific for the model used in this dissertation, but most of them are common to all models. Quadcopters are composed of a carbon fiber frame where motors and its controllers are installed. Motors' controllers are called ESC, from Electronic Speed Controller.

The flight controller is responsible for the low-level control of our drone. It controls the motors' speed, and, consequently, the drone's stabilization, piloting and positioning based on GPS coordinates.

There is also a high-level control that can be done, or not, by the same flight controller, and its in charge of the algorithms of path following, coordination, and communication between the flock's elements.

To these boards are connected all the sensors, antennas and external communication devices, like GPS antennas, radio receivers, and communication modules, to communicate with a ground station or other drones.

To distribute the energy from the battery to all components, a PDB (Power Distribution Board) is used.

To better understand the role of each motor in drone's movements, motors are enumerated as shown in the figure 2.2, with the x-axis pointing to the front of the drone, y-axis pointing to its right side and z pointing down.

2.1.1.3 Working principle

In order to prevent the drone from rotating around itself when motors are spinning, opposite motors have opposite rotation directions, so that in this way the total momentum when all motors spin at the same speed is equal to zero. Otherwise, if the intent is to rotate the drone around z-axis (yaw), the speed of a group of opposite motors must be higher, or lower, than the other, as shown in figure 2.3 (c) and (f)!

To hover, drones must keep all motors, approximately, at the same speed. Then, depending on what the desired move is, all four motors are adjusted accordingly. To move up or down, all motors must increase or decrease their speed.

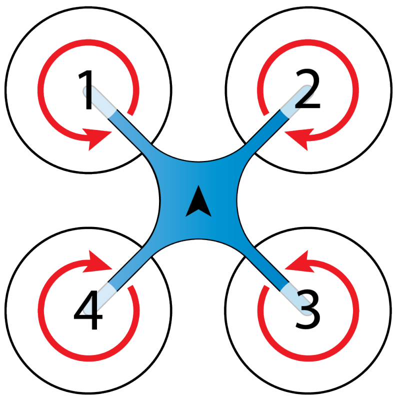
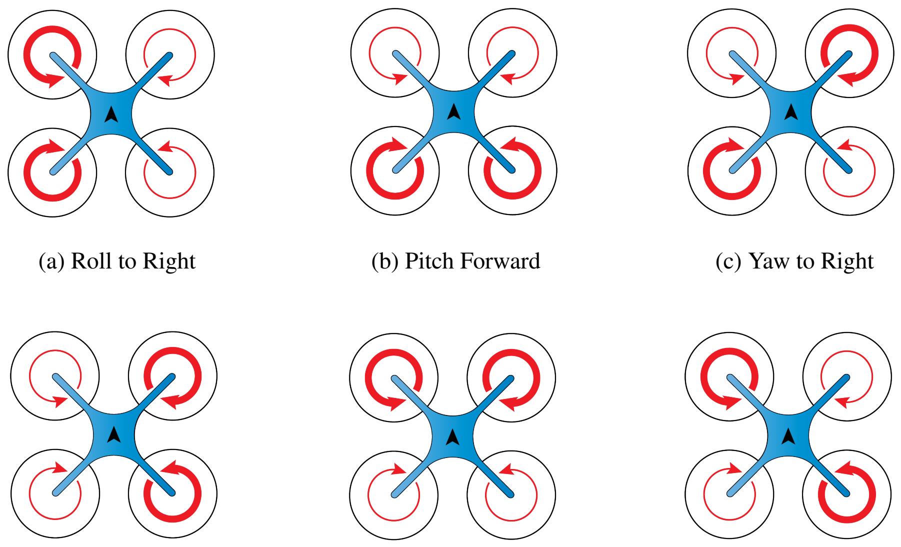


Figure 2.2: Direction of a quadcopter's motor rotation

Rotating right or left is achieved by having the speed of motors 1 and 4 higher than motors 2 and 3, and vice-versa, respectively, as shown in figure 2.3 (a) and (d). Rotating to the front or the back works the same way. Motors 1 and 2 must spin faster than 3 and 4, and vice versa (figure 2.3 (b) and (e)).



(a) Roll to Right (b) Pitch Forward (c) Yaw to Right

(d) Roll to Left (e) Pitch Backward (f) Yaw to Left

Figure 2.3: Motors speed in each rotation move.

2.1.2 Flocking and Cooperative Path Following Algorithms

A flock is a collective movement of individual interacting entities. This behavior is constantly observed in the nature, from small insect migrations to large mammals such as elephants, and obviously birds. This phenomenon can even be observed in microorganisms. The ability of these entities, normally of low intelligence, which communicate only with their nearest neighbors, to position themselves properly so that they move in a collective, collision free and ensuring the safety of the group, has awakened the interest of various researchers over the last decades. Several models of great relevance emerged from these investigations:

- Reynolds' Model (1987) [1] - which consists in three flocking rules:
  - Separation - attempt to avoid crowding and collision
  - Cohesion - attempt to stay close to its neighbors
  - Alignment - match velocity with nearby flock-mates

  This model led to the creation of the first computer animation of a flock.

- Vicsek's Model (1995) [2] - all agents move at the same speed with direction equal to the average of the directions in the previous time instant.

- Veysel Gazi's Attraction/Repulsion Model (2002) [3] - all agents are aware of the distance to there neighbors. These feel attraction at large distances and repulsion at short distances.

- Tanner's *Flocking Model* (2003) [4] - these control laws are a combination of attraction/repulsion and alignment forces, ensuring that there are no collisions and the whole group moves together towards a common point.

- Olfati-Saber's *Flocking Model* (2006) [5] - which features three algorithms that improve the group's collective movement:

  - Algorithm 1 is in charge of organizing the flock
  - Algorithm 2 adds to algorithm 1 terms that take into account the purpose of the group.
  - Algorithm 3 adds to previous algorithms parameters to avoid obstacles.

Among others, some of them are minor adaptations of previously mentioned models, for example Moreau (2005), Ren and Beard (2005), Olfati-Saber (2007), Tanner et al. (2007), Gazi (2008), and Su et al. (2009a, 2009b).

In the most recent bibliography of Jingyuan Zhan, Xiang Li and others, 2011 [19] and 2017 [20], predictive control models are introduced to control UAV networks. For several decades, from Woods (1959) to Montague et al. (1995), it has been experimentally proven that groups in nature have predictive intelligence, that is, each individual can predict their position and group's position based on their and on the flock's previous positions. Jingyuan Zhan, et al. (2008)|, shows that predictive mechanisms play an important role in the collective consensus of a flock. Therefore, the authors assume that predictive models also play an important role in multi-agent system's control, and created a new algorithm based on predictive models, developing for that purpose a *Model Predictive Controller (MPC)* that improves the flock performance.

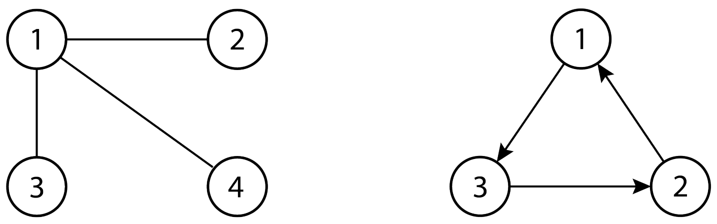
Years later, in [20] a flocking algorithm based on MPC for decentralized multi-agent systems was developed - *Decentralized Model Predictive Control Algorithm (DMPC Algorithm)*. After simulations and testing, the authors find that the proposed algorithm is applicable to multi-drone systems (with a passive leader) and has several advantages such as fast convergence, ease of tuning and low processing capacity.

2.2 Theoretical Preliminaries

2.2.1 Graphs

2.2.1.1 Concept

A graph is composed of a set of nodes or vertices, $V = \{v_1,..,v_n\}$, and a set of lines or edges, $E = \{(v_1v_2),(v_2v_3),..,(v_{n-1}v_n)\}$, and is usually denoted by $G(V,E)$, or simply G.



(a) Undirected graph (b) Digraph

Figure 2.4: Direct vs. Undirected graphs

Graphs are used in this work to model the communication between the vehicles in the fleet.

When applicated to multi-agent systems, the nodes represent each agent of the system and the lines the existence of communication between them.

If there is a pair of nodes with a edge joining them, they are called adjacent. Each pair is composed of a head and a tail, and information flows from the first node to the second one.

If communication occurs in both directions, the graph is undirected. Otherwise, it is called a directed graph or digraph (see figure 2.4).

2.2.1.2 Algebraic Representation

In order to represent graphs more understandable, a matrix will be used. In graph theory and computer science, this matrix is called Laplacian Matrix.

It is expressed as following:

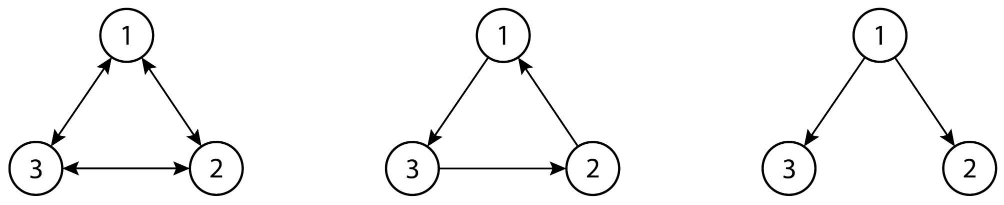
$$L = D - A \tag{2.1}$$

where D is the degree matrix, and A the adjacency matrix.

Let $A(G)$ be the adjacency matrix with respect to G. Its elements indicate whether pairs of vertices are adjacent or not in the graph and are defined as $a_{ij} = 1$ if $(v_i, v_j) \in E$, $a_{ij} = 0$ otherwise.

$D(G)$ is the degree matrix of a graph G. The diagonal elements of this square matrix are the out-degrees of the node v_{ii}. That is, each diagonal element denotes the number of nodes that v_{ii} is connected to. In other words, how many edges of the graph start from this node.

By definition, each row of the Laplacian Matrix L sums up to zero.



(a) All nodes communicate with each other (b) Each node receives information from the previous one (c) Nodes 2 and 3 receive information from node 1

Figure 2.5: Three diferent graph configurations

$$\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \qquad \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.2}$$

In figure 2.5, we can see some directed graphs and, underneath it, the respective Laplacian matrices.

2.2.2 Space State Model Representation

In control engineering, systems are frequently modeled by its state-space representation.

This representation provides the plant dynamics as a set of state, $x(t) = [x_1(t), x_2(t), ..., x_n(t)]^T$, input, $u(t) = [u_1(t), u_2(t), ..., u_r(t)]^T$, and output variables, $y(t) = [y_1(t), y_2(t), ..., y_m(t)]^T$, related by first-order differential equations.

The state of a system refers to a set of variables, the state variables, that describe the system at a given instant in response to the values of input variables. For Non-Linear systems, the state equations may be written as $\dot{x} = f(t, x, u)$. This way of describing a system expresses the time derivative of each state in terms of state variables and system inputs.

The output of a system is defined to be any system variable of interest, and, for a majority of physical systems, it depends only on the state variables.

The following equation shows us how the output equation of a system is normally written for this kind of systems, $y = g(t, x, u)$.

2.2.2.1 Unicycle Example

In this chapter, the model of a unicycle on a 2D plan will be designed using its space-state representation. Such a primary example is meant to clarify how space-state representation works, and the same study case will be later used to introduce the toolkit used during this dissertation.

Coordinate Frames

This model has two reference frames: the inertial coordinate frame, I, and the body coordinate frame, B, attached to the unicycle's body.

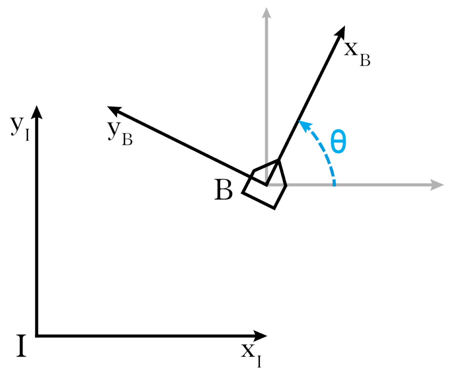


Figure 2.6: Inertial and body coordinate frames

The relation between both frames is given by the rotation matrix R, uses Euler angles, and for this 2D example takes the following form:

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{2.3}$$

θ is the orientation of the vehicle as represented in figure 2.6.

Model

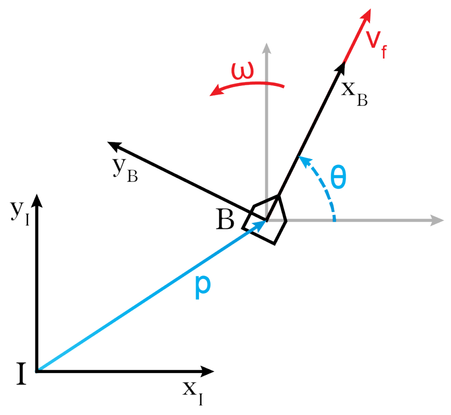


Figure 2.7: Complete representation of the unicycle model

The variables of interest of this system are the position and orientation of the unicycle. So, our state vector will look like this:

$$x(t) = \begin{bmatrix} p(t) & \theta(t) \end{bmatrix}^T \in R^3 \tag{2.4}$$

The inputs of the unicycle are the linear and angular velocities, v and ω, respectively. As our unicycle can only move forward, and backward, the velocity perpendicular to the body, $v_n(t)$, is equal to zero. Therefore,the linear vector velocity expressed in the body frame is given by $v(t) = \begin{bmatrix} v_f & 0 \end{bmatrix}^T \in R^2$.

The system's control input vector may be written as follow

$$u(t) = \begin{bmatrix} v_f(t) & \omega(t) \end{bmatrix}^T \in R^2 \tag{2.5}$$

The kinematic model of the unicycle is described by

$$\dot{p}(t) = R(t)v(t) \tag{2.6}$$

$$\dot{R}(t) = R(t)sk(\omega(t)) \tag{2.7}$$

where R is the rotation matrix and the matrix $sk(\omega(t))$ is the skew-symmetric matrix associated with the angular velocity $\omega(t)$.

The state-space model (2.6)-(2.7) can be further simplified to

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_f(t) \\ \omega(t) \end{bmatrix} \tag{2.8}$$

2.2.3 Lyapunov's Stability Theorem

This method, called the direct method, uses a Lyapunov function $V(x)$ to prove the stability of a system.

Let $\dot{x} = f(x)$ be a system with an equilibrium point at $x = 0$ and $D \subset R^n$ be a domain containing $x = 0$.

Consider a function $V(x) : D \to R$ continuously diferentiable such that

$V(0) = 0$, and

$V(x) > 0, \quad \forall x \in D \setminus \{0\}$.

Then,

$x = 0$ is **stable** if $\dot{V}(x) \leq 0, \quad \forall x \in D \setminus \{0\}$.

$x = 0$ is **asymptotically stable** if $\dot{V}(x) < 0, \quad \forall x \in D \setminus \{0\},$.

If $V(x) : R^n \to R$ is a function continuously diferentiable such that

$V(0) = 0$,

$V(x) > 0, \quad \forall x \in D \setminus \{0\}$, and

$\| x \| \to \infty \quad \Rightarrow \quad V(x) \to \infty, \quad \forall x \setminus \{0\}.$
If $\dot{V}(x) < 0, \quad \forall x \setminus \{0\}$
$x = 0$ is now **globally asymptotically stable** (GAS).

This method will be used later to derive the control laws for our systems. Control Lyapunov functions will be proposed to find suitable control laws for the dynamic tracking error equation, as we will see later on this work.

2.2.4 Agreement Protocol

Making a drone flock achieve consensus is the main goal of the controller designed in this dissertation. In this sub chapter, the agreement protocol used in that controller is introduced. That protocol is pretended to work on undirected and directed static networks.

Consensus is achieved when the agents of a flock, swarm or another kind of network, join all in a state value, and it is one of the fundamental problems in multi-agent systems coordination.

Static networks are those networks where the edges do not vary over time. Which is the case of this work, due to limitations of the toolkit used in the simulations.

Let n be the number of dynamic agents that make up our network, and they can change information with other elements.

The variation of each agent's state results from the sum of its relative states with respect to its neighborhood (a subset of agents adjacent to it).

Taking the graph presented in figure 2.5 a), an example of the agreement protocol will be presented next.

Let each node's state be represented as x_i, being i its respective number. The dynamic of each node's state is given by

$$\dot{x}_1 = (x_2 - x_1) + (x_3 - x_1) \tag{2.9}$$

$$\dot{x}_2 = (x_1 - x_2) + (x_3 - x_2) \tag{2.10}$$

$$\dot{x}_3 = (x_1 - x_3) + (x_2 - x_3) \tag{2.11}$$

Or, in its generalized form, as follows

$$\dot{x}_i(t) = \sum_{j \in N_i} (x_j(t) - x_i(t)), \quad i = 1, ..., n \tag{2.12}$$

2.3 Virtual Arena

VirtualArena is an open-source Object-Oriented Matlab Toolkit for control design and system simulation.

The use of this toolkit reduces the time spent on the design and validation of a control architecture.

This toolkit provides a set of ready-to-use functions often used in control design that, along with its object-oriented architecture, increase the modularity, reliability and reusability of the controller's components and reduce the development time.

After defining our model's system, its initial conditions, design a control input to drive it to the desired state, the architecture is validated via simulation. To make it more realistic, state and output disturbances can be added. All of these steps can be done with functions included in the toolkit. It makes the simulation phase as easy as specify the system, stopping criteria, discretization step and run it, for example.

More information about how to use the toolkit can be found in [13] and [14].

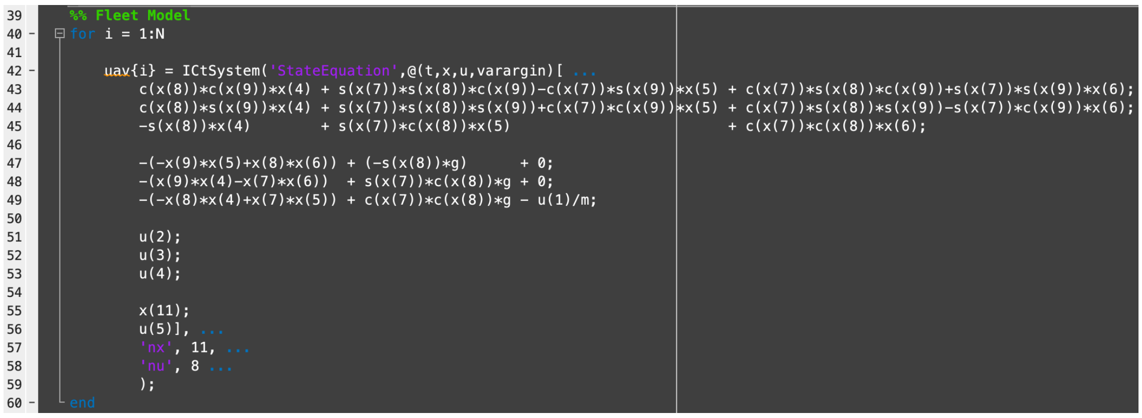
In the next subsection the final implementation of this work in Virtual Arena will be presented for two reasons. First, to show a different, more complex, example of implementation with the toolkit, that anyone who's trying to understand how it works or how to implement some specific component and don't find documentation for it. Secondly, to show how the following work was implemented. For that, it's advisable to come back after reading chapters 3 and 4.

2.3.1 Drone Flock Example

As said in the beginning of this chapter, Virtual Arena keeps the design and simulation simple and organized in a few functions.

2.3.1.1 Fleet Model

It starts with the Fleet Model definition. First specifying the type of system and some parameters as the state equations, lines 42 to 56, number of states and number of inputs, lines 57 and 58.



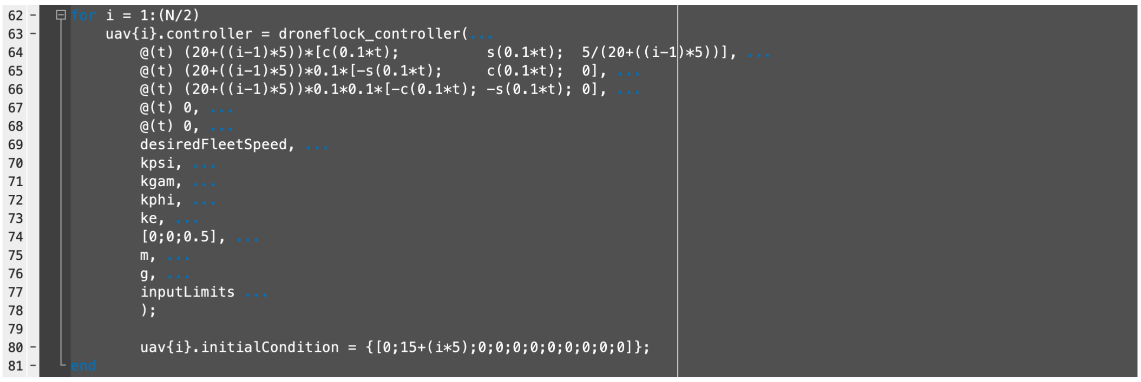
```
39    %% Fleet Model
40 -  for i = 1:N
41
42 -      uav{i} = ICtSystem('StateEquation',@(t,x,u,varargin)[ ...
43            c(x(8))*c(x(9))*x(4) + s(x(7))*s(x(8))*c(x(9))-c(x(7))*s(x(9))*x(5) + c(x(7))*s(x(8))*c(x(9))+s(x(7))*s(x(9))*x(6);
44            c(x(8))*s(x(9))*x(4) + s(x(7))*s(x(8))*s(x(9))+c(x(7))*c(x(9))*x(5) + c(x(7))*s(x(8))*s(x(9))-s(x(7))*c(x(9))*x(6);
45            -s(x(8))*x(4)        + s(x(7))*c(x(8))*x(5)                         + c(x(7))*c(x(8))*x(6);
46
47            -(-x(9)*x(5)+x(8)*x(6)) + (-s(x(8))*g)      + 0;
48            -(x(9)*x(4)-x(7)*x(6))  + s(x(7))*c(x(8))*g + 0;
49            -(-x(8)*x(4)+x(7)*x(5)) + c(x(7))*c(x(8))*g - u(1)/m;
50
51            u(2);
52            u(3);
53            u(4);
54
55            x(11);
56            u(5)], ...
57            'nx', 11, ...
58            'nu', 8 ...
59            );
60 -  end
```

Figure 2.8: Fleet Model implementation in Virtual Arena

Lines 43 to 45 represent the first three states, $\dot{x}(t)$. States 4, 5 and 6 are the speed evolution in each cycle, $\dot{v}(t)$.

As the inputs of the system (u(1) to u(5)) are the variation in each iteration and the dynamic kinematic equations need the absolute value of the inputs, we can take advantage of Virtual Arena's integrated state integration, as well as its automatic rotation matrix update in each cycle. Therefore, to integrate any state or input, we just have to create a new state variable and Virtual Arena will automatically do its integration for us. It happens for our inputs 2, 3 and 4, that are the variations in angular velocity. After each cycle, states 7, 8 and 9 will have the absolute value of the drone's angles.

The same happens in states 10 and 11. Our controller generates a second derivative of γ, the parametric variable of the controller. To get the variable itself, it is integrated two times by adding two new state variables to the system. State 11 gets $\dot{\gamma}$ and state 10 gets γ. As will be shown later, only state 10 will be used by the controller as the parametric variable.



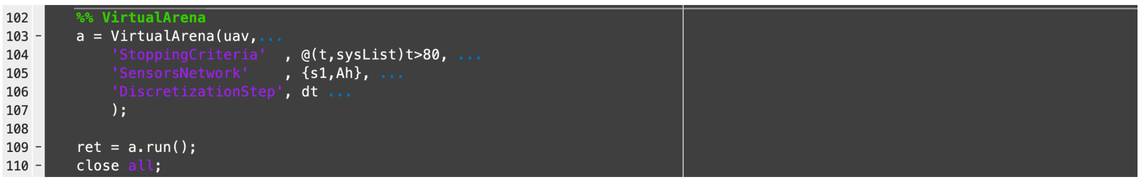
```
62   for i = 1:(N/2)
63       uav{i}.controller = droneflock_controller(...
64           @(t) (20+((i-1)*5))*[c(0.1*t);          s(0.1*t);  5/(20+((i-1)*5))], ...
65           @(t) (20+((i-1)*5))*0.1*[-s(0.1*t);     c(0.1*t);  0], ...
66           @(t) (20+((i-1)*5))*0.1*0.1*[-c(0.1*t); -s(0.1*t); 0], ...
67           @(t) 0, ...
68           @(t) 0, ...
69           desiredFleetSpeed, ...
70           kpsi, ...
71           kgam, ...
72           kphi, ...
73           ke, ...
74           [0;0;0.5], ...
75           m, ...
76           g, ...
77           inputLimits ...
78           );
79
80           uav{i}.initialCondition = {[0;15+(i*5);0;0;0;0;0;0;0;0;0]};
81   end
```

Figure 2.9: Drone Flock controller call in Virtual Arena

Then the controller, its arguments and its initial conditions are specified, as shown in figure 2.9.

The toolkit has controllers that fit many scenarios, as the simple case of the unicycle type robot and the basic fleet also studied in this work. For the main subject, it was necessary to create a new controller object named $droneflock\_controller()$.

This controller expect arguments as the desired position, its first and second derivatives (lines 64 to 66), some information about the environment and the drone used, as the value of the gravity force, the drones mass and their physical limitations (lines 75 to 77), the controller gains and desired fleet speed (lines 69 to 73).



```
%% VirtualArena
a = VirtualArena(uav,...
    'StoppingCriteria'  , @(t,sysList)t>80, ...
    'SensorsNetwork'    , {s1,Ah}, ...
    'DiscretizationStep', dt ...
    );

ret = a.run();
close all;
```

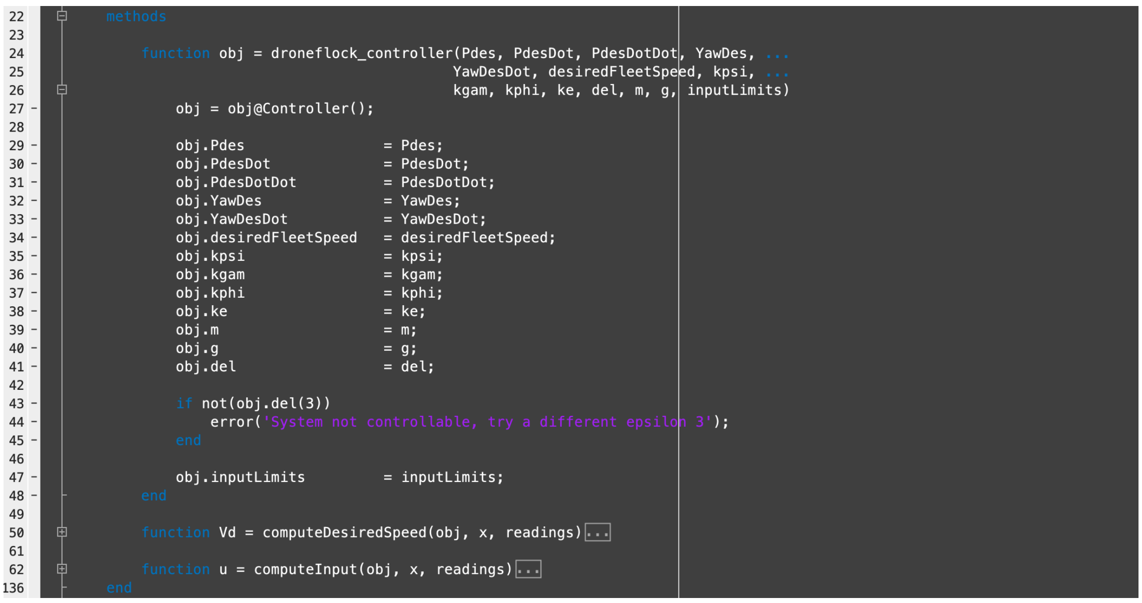
Figure 2.10: Virtual Arena simulation setting and run

Once the system is defined by its initial and desired behavior and its controller, its time to set and run the virtual arena simulation. As said in the toolkit introduction, with Virtual Arena, run this complex simulation can be as easy as specify the system, its stopping criteria, discretization step and, in this case, as the controller requires information from another agents from the fleet, a sensor network is also specified.

The sensor network used in this work is similar to the one in Virtual Arena's examples, that can be found in [14].

A plot function could also has been specified, but due to the size and time taken by this simulation, the plots are done after the simulation is finished by simply choosing the desired data in *ret*, where all simulation data is stored during the call of *run*(), and plotting it as it's usually done in Matlab.

The main contribution of this work to Virtual Arena was the creation of a cooperative path following controller for a drone flock.



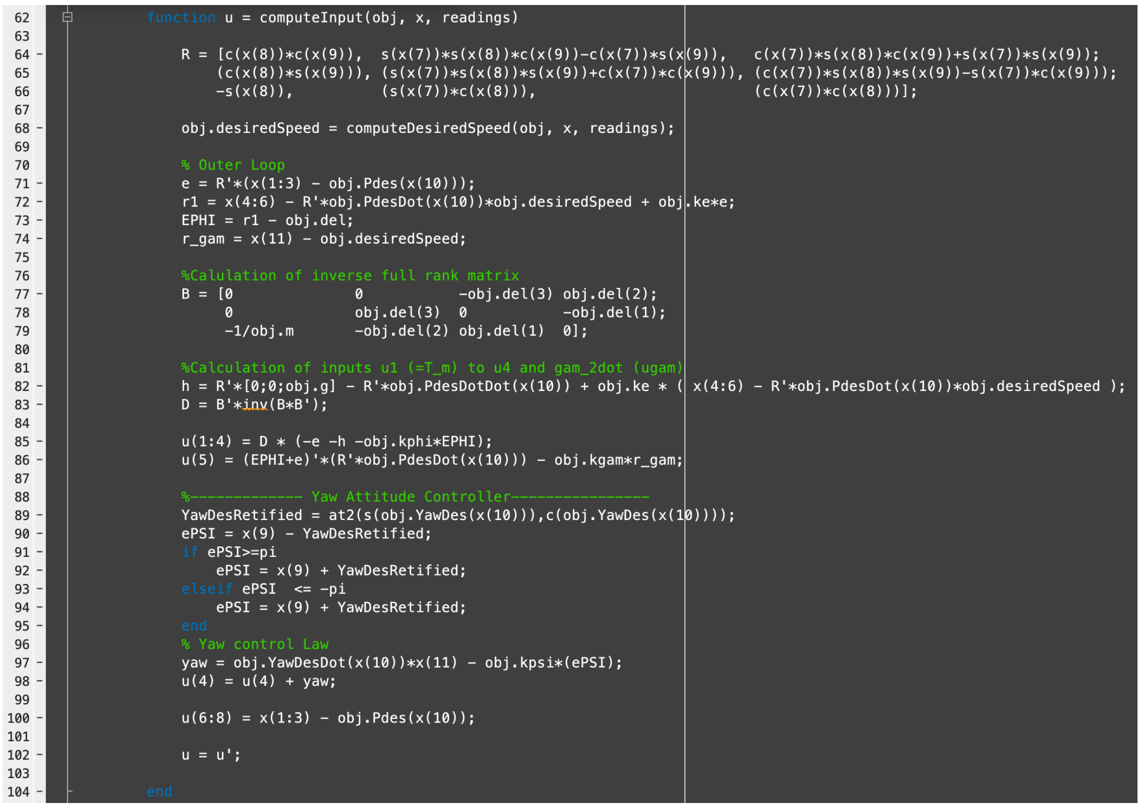
```
    methods

        function obj = droneflock_controller(Pdes, PdesDot, PdesDotDot, YawDes, ...
                                            YawDesDot, desiredFleetSpeed, kpsi, ...
                                            kgam, kphi, ke, del, m, g, inputLimits)
            obj = obj@Controller();

            obj.Pdes                = Pdes;
            obj.PdesDot             = PdesDot;
            obj.PdesDotDot          = PdesDotDot;
            obj.YawDes              = YawDes;
            obj.YawDesDot           = YawDesDot;
            obj.desiredFleetSpeed   = desiredFleetSpeed;
            obj.kpsi                = kpsi;
            obj.kgam                = kgam;
            obj.kphi                = kphi;
            obj.ke                  = ke;
            obj.m                   = m;
            obj.g                   = g;
            obj.del                 = del;

            if not(obj.del(3))
                error('System not controllable, try a different epsilon 3');
            end

            obj.inputLimits         = inputLimits;
        end

        function Vd = computeDesiredSpeed(obj, x, readings)...

        function u = computeInput(obj, x, readings)...
    end
```

Figure 2.11: Cooperative Path Following for a Drone Flock Implementation in Virtual Arena

The drone flock controller's method is presented in figure 2.11. Virtual arena will run, by default, the *computeInput()* function in each cycle. Any needed complementary functions were added to this method and are called by the *computeInput()* function when needed. It begins by assigning the value of the arguments to the corresponding property and checking for invalid arguments.

The function *computeDesiredSpeed()* is in charge of reading variables of interest from the neighbours of the drone and calculating the desired speed for each drone to reach consensus and the desired speed.

In *computeInput()* are the control laws and all the necessary calculations done in sections 4.1.2 and 4.2. Figure 2.12 shows a simple version of that function, due to size constraints. Drones' limitations and comments were added to the final version.



```matlab
        function u = computeInput(obj, x, readings)

            R = [c(x(8))*c(x(9)),  s(x(7))*s(x(8))*c(x(9))-c(x(7))*s(x(9)),   c(x(7))*s(x(8))*c(x(9))+s(x(7))*s(x(9));
                (c(x(8))*s(x(9))), (s(x(7))*s(x(8))*s(x(9))+c(x(7))*c(x(9))), (c(x(7))*s(x(8))*s(x(9))-s(x(7))*c(x(9)));
                -s(x(8)),          (s(x(7))*c(x(8))),                          (c(x(7))*c(x(8)))];

            obj.desiredSpeed = computeDesiredSpeed(obj, x, readings);

            % Outer Loop
            e = R'*(x(1:3) - obj.Pdes(x(10)));
            r1 = x(4:6) - R'*obj.PdesDot(x(10))*obj.desiredSpeed + obj.ke*e;
            EPHI = r1 - obj.del;
            r_gam = x(11) - obj.desiredSpeed;

            %Calulation of inverse full rank matrix
            B = [0             0           -obj.del(3) obj.del(2);
                 0             obj.del(3)  0           -obj.del(1);
                 -1/obj.m      -obj.del(2) obj.del(1)  0];

            %Calculation of inputs u1 (=T_m) to u4 and gam_2dot (ugam)
            h = R'*[0;0;obj.g] - R'*obj.PdesDotDot(x(10)) + obj.ke * ( x(4:6) - R'*obj.PdesDot(x(10))*obj.desiredSpeed );
            D = B'*inv(B*B');

            u(1:4) = D * (-e -h -obj.kphi*EPHI);
            u(5) = (EPHI+e)'*(R'*obj.PdesDot(x(10))) - obj.kgam*r_gam;

            %------------- Yaw Attitude Controller----------------
            YawDesRetified = at2(s(obj.YawDes(x(10))),c(obj.YawDes(x(10))));
            ePSI = x(9) - YawDesRetified;
            if ePSI>=pi
                ePSI = x(9) + YawDesRetified;
            elseif ePSI  <= -pi
                ePSI = x(9) + YawDesRetified;
            end
            % Yaw control Law
            yaw = obj.YawDesDot(x(10))*x(11) - obj.kpsi*(ePSI);
            u(4) = u(4) + yaw;

            u(6:8) = x(1:3) - obj.Pdes(x(10));

            u = u';

        end
```

Figure 2.12: Content of *computeInput()* function

Chapter 3

System Model

The first step to do for model based control controller design is to derive a model that describes the dynamic of our system.

In section 3.1 are presented the coordination frames used in our model representation, which is done in the following chapter.

After getting the model for a quadcopter, it is explained how the flock is composed, how it works, and what the desired behavior is.

3.1 Coordination Frames

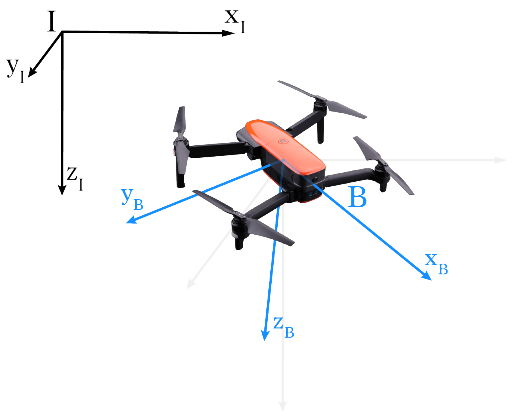


Figure 3.1: Inertial and Body Coordinate Frames

This model uses two coordinates frames, generally as illustrated in figure 3.1. The position and orientation of the UAV are expressed in the inertial reference frame I:

- $p = [p_x, p_y, p_z]^T$: Position of the origin of B, expressed in I
- $\eta = [\phi, \theta, \psi]^T$: Orientation of B with respect to I

Usually, this frame is placed on the ground. It is formed by three orthonormal axes, x_I, y_I, z_I, that are pointing to north, east and down, respectively.

The linear velocities of the drone are expressed in the second frame, the body frame B, and are measured relative to I:

- $v = [v_x, v_y, v_z]^T$ the linear velocities of body frame B relative to I, expressed in B;
- $\omega = [\omega_1, \omega_2, \omega_3]^T$: the angular velocities of B with respect to I, expressed in B

This frame is usually fixed in the center of gravity (CG) of the vehicle, and oriented as shown in the figure 3.1 to simplify the model of the system:

- x_B is placed on the roll axis, pointing to the front of the drone.
- y_B is on the pitch axis, pointing to de right side of the vehicle.
- z_B is placed vertically on the yaw axis, pointing downward.

The transformation of body frame coordinates to corresponding inertial frame coordinates it's achieved by the following transformation matrix,

$$R_{b-i}(\theta, \phi, \psi) = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \tag{3.1}$$

with c being *cos*, and s being *sin*.

3.2 Quadcopter Model

The vehicle used in this dissertation is a multicopter with four motors, also known as a quadcopter.

Our quadcopter's model has six degrees of freedom (DoF), including three translational movements and three-axis rotation (roll, pitch, and yaw), generally expressed by Euler angles.

As explained in the introduction, the only actuators in a quadcopter are its four motors and are responsible for controlling all six DoF of our model. Its velocities are obtained by adjusting the difference between motors' torque accordingly to the desired movement.

In this dissertation, the focus goes only to a higher level of control of the drone. Our goal is to find the roll, pitch, and yaw angles as wells as the thrust force of the vehicle. Exactly as if we were controlling it via a radio controller. The conversion to each motor's torque is trusted to a lower level controller, in our case, a flight controller.

Before getting started with the model derivation, a simplified model is taken into consideration, and, so that, the following assumptions can be made:

- The drone's frame is small enough to be considered rigid and symmetric.

- The center of mass coincides with the center of lift.
- Aerodynamic effects are neglected.

These assumptions are commonly used in UAV model design and seemed a proper point from where to start the design of our controller. Then a general model of a 3D rigid body is used in this work [18].

Equations (3.2) and (3.3) are the kinematic equations of our model:

$$\dot{p}^I = Rv^b \tag{3.2}$$

$$\dot{R} = Rsk(\omega^b) \tag{3.3}$$

Equation (3.4) describes the dynamic of our model:

$$M\dot{v}^b = -sk(\omega^b)v^b + F_g + F_T \tag{3.4}$$

where $sk(.)$ is a skew symmetric matrix such that:

$$sk(\omega) = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{3.5}$$

As we can see in equation (3.4), two main forces are acting in our drone: the thrust, F_T, and its weight, F_g, as shown next.

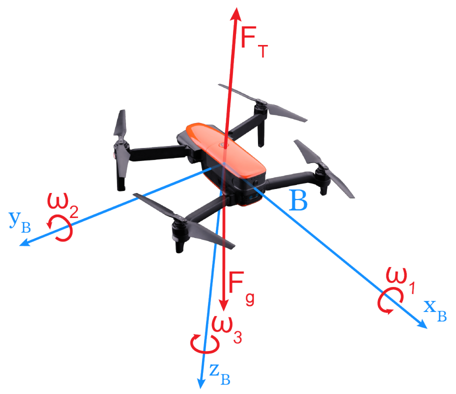


Figure 3.2: Forces and inputs acting on our body

F_T has only influence on the vertical speed, v_z, of the drone (on frame B). However, the change of its angles causes a change in the drone's coordinates (on frame I) due to the rise of thrust's components. F_g is a constant force acting on z-axis on frame I. As opposed to F_T, the change of drone's angles makes components of F_g emerge on the body's frame.

$$F_T = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} u_1 \tag{3.6}$$

$$F_g = R^T \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{3.7}$$

Some limitations must be taken into consideration to make these equations represent the model as accurately as possible. In practical applications, drones have limits, imposed by the user or its physical limitations, in thrust force's intensity, on its angular velocities, and, depending on the flight mode, on the values that its angles can take.

Therefore, when the controller is implemented on Virtual Arena, the following limitations are applied

$$0 \leq F_T \leq F_{TMax} \tag{3.8}$$

Our model is from a quadcopter with a conventional propulsion system, it does not have 3D capabilities, as varying pitch helicopters, 3D freestyle airplanes or 3D drones, that can propel themselves downward.

$$-Roll_{MaxRate} \leq \omega_1 \leq Roll_{MaxRate} \tag{3.9}$$

$$-Pitch_{MaxRate} \leq \omega_2 \leq Pitch_{MaxRate} \tag{3.10}$$

$$-Yaw_{MaxRate} \leq \omega_3 \leq Yaw_{MaxRate} \tag{3.11}$$

Chapter 4

Motion Control

In this chapter, the system's motion control is the case of study. Initially, control laws for path following are derived for a simple case of a unicycle and then for a quadcopter. Lastly, the control law that keeps the agents in formation is presented, which is the main goal of this dissertation.

To easily understand the flocking behavior of a system and how the toolkit used in this dissertation works, a basic 2D example of a unicycle flock is first presented, and later by doing it for our, more complex, in 3D, quadcopter flock.

4.1 Path Following

When we talk about motion control, trajectory tracking and path following are many times misunderstood. Path-following problems are concerned with the design of control laws that drive the vehicle to a geometric path parameterized in space, while simultaneously satisfying the dynamic specification of the parametric variable, unlike trajectory tracking. This variable is introduced in our system as a new control input, called γ, and can be seen as a virtual point traveling along the path.

In the next two study cases, the problem is approached in two parts: the tracking error problem and the dynamic of the path parametric variable.

4.1.1 Unicycle

4.1.1.1 Path Following Problem Statement

Consider our unicycle that is described by the kinematic model introduced in 2.2.2.

Let $p_d(\gamma) : R \to R^2$, be our desired path.

A controller must be designed such that the position of the unicycle converges to the desired one, that is,

$$t \to \infty \quad \Rightarrow \quad p(\gamma) \to p_d(\gamma) \tag{4.1}$$

In other words, when t tends to infinite, the following error must tend to zero, as shown next:

$$e = R^T(p(\gamma) - p_d(\gamma)) \to 0 \tag{4.2}$$

To make our vehicle follow the path with the desired dynamic, we provide the following requirement for the parameterization variable. Let $v_d \in R$ be our desired speed assignment along the path. A second controller's goal is to make the drone's parameterization speed meet v_d, that is,

$$t \to \infty \quad \Rightarrow \quad \dot{\gamma} \to v_d \tag{4.3}$$

It also means that the speed's error must converge to zero over time.

$$z = \dot{\gamma} - v_d \to 0 \tag{4.4}$$

4.1.1.2 Proposed Solution

The path following tasks are formally introduced.

Design a control law for u and $\ddot{\gamma}$ such that the two conditions mentioned above are satisfied.

Consider the same kinematic model described in 2.2.2.

The error dynamic is given by

$$e = R^T(p - p_d(\gamma)) \tag{4.5}$$

$$\dot{e} = -sk(\omega)e - R^T\dot{\gamma}\dot{p}_d(\gamma) + \Delta u \tag{4.6}$$

With Δ and $sk(\omega)$ as follows

$$\Delta = \begin{bmatrix} 1 & -\varepsilon_2 \\ 0 & \varepsilon_1 \end{bmatrix} \tag{4.7}$$

$$sk(\omega) = \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix} \tag{4.8}$$

And the parametric variable error as exposed next

$$z = \dot{\gamma} - v_d \tag{4.9}$$

$$\dot{z} = \ddot{\gamma} \tag{4.10}$$

It is now possible to define a Lyapunov Function composed by a Lyapunov function to the tracking error problem and another one to the path speed error:

$$V_c = V_e + V_z \tag{4.11}$$

With the tracking error function:

$$V_e = \tfrac{1}{2} e^T e \tag{4.12}$$

$$\dot{V}_e = e^T \dot{e} = e^T(-sk(\omega)e - R^T \dot{\gamma}\dot{p}_d(\gamma) + \Delta u) \tag{4.13}$$

and

$$V_z = \tfrac{1}{2} z^2 \tag{4.14}$$

$$\dot{V}_z = z\dot{z} = z\ddot{\gamma} \tag{4.15}$$

Back to the Lyapunov composite function.

$$V_c = \tfrac{1}{2} e^T e + \tfrac{1}{2} z^2 \tag{4.16}$$

Then, its derivative with respect to time is

$$\dot{V}_c = \dot{V}_e + \dot{V}_z = e^T(-sk(\omega)e - R^T \dot{\gamma}\dot{p}_d(\gamma) + u + z\ddot{\gamma} \tag{4.17}$$

With $\dot{\gamma} = z + v_d$,

$$\begin{aligned} \dot{V}_c &= e^T(-sk(\omega)e - R^T(z + v_d(\gamma))\dot{p}_d(\gamma) + u + z\ddot{\gamma} \\ \dot{V}_c &= e^T(-sk(\omega)e - R^T v_d(\gamma)\dot{p}_d(\gamma) + u + z(\ddot{\gamma} + eR^T \dot{p}_d(\gamma)) \end{aligned} \tag{4.18}$$

To ensure stability, both terms of $\dot{V}_c$ must be negative definite, what happens for the u and $\ddot{\gamma}$ presented below.

$$u = \Delta^{-1}(R^T v_d(\gamma)\dot{p}_d(\gamma) - k_e e(\gamma)) \tag{4.19}$$

$$\ddot{\gamma} = eR^T \dot{p}_d(\gamma) - k_\gamma z \tag{4.20}$$

with k_e and k_γ positive values.

Control laws to make our unicycle following a path were found. Simulation results can be found in chapter 5.

4.1.2 Quadcopter

4.1.2.1 Path Following Problem Statement

Moving now to a 3 dimension path following example, both path tracking and path following objectives are the same as in the unicycle case. The position error, that has now three coordinates instead of only two, and the speed error must converge to zero over time.

$$t \to \infty \quad \Rightarrow \quad p(\gamma) \to p_d(\gamma) \tag{4.21}$$

$$t \to \infty \quad \Rightarrow \quad \dot{\gamma} \to v_d(\gamma) \tag{4.22}$$

Error equations:

$$e = R^T(p - p_d(\gamma)) \tag{4.23}$$

$$z = \dot{\gamma} - v_d(\gamma) \rightarrow 0 \tag{4.24}$$

4.1.2.2 Proposed Solution

To solve this problem, a Lyapunov approach is used, as done in the unicycle example.

Our position dynamic error is given by

$$\dot{e} = -sk(\omega)e + v - R' p_d(\gamma)\dot{\gamma} \tag{4.25}$$

From the speed error equation, $\dot{\gamma} = z - v_d(\gamma)$,
by replacing $\dot{\gamma}$ in the previous expression we get

$$\dot{e} = -sk(\omega)e + v - R'\dot{p}_d(\gamma)v_d - R'\dot{p}_d(\gamma)z \tag{4.26}$$

A candidate Lyapunov function is now proposed.

$$V_1 = \frac{1}{2}e^T e \tag{4.27}$$

Its derivative is

$$\begin{aligned} \dot{V}_1 &= e^T \dot{e} = e^T(-sk(\omega)e + v - R'\dot{p}_d(\gamma)v_d - R'\dot{p}_d(\gamma)z) \\ \dot{V}_1 &= e^T(v - R'\dot{p}_d(\gamma)v_d) - e^T R'\dot{p}_d(\gamma)z \end{aligned} \tag{4.28}$$

In equation (4.25), v is seen as a virtual input used to ensure that the derivative of the Lyapunov function is negative-definite, which it happens for

$$v = R^T \dot{p}_d(\gamma)v_d - k_e e, k_e > 0, \tag{4.29}$$

Since v is a virtual input, we call v_{d2} the desired control law for v_d in equation (4.26)

The goal is now to make v to converge to v_{d2}. To this end, a new error variable, r_1, is defined

$$r_1 = v - v_{d2} = v - R'\dot{p}_d(\gamma)v_d + k_e e \tag{4.30}$$

$$\Leftrightarrow v = r_1 + R'\dot{p}_d(\gamma)v_d + k_e e \tag{4.31}$$

Replacing v in equation (4.28)

$$\dot{V}_1 = -k_e e^T e + e^T r_1 - e^T(R^T \dot{p}_d(\gamma))z \tag{4.32}$$

The first and last term of $\dot{v}_1$ are negative, but the second one is not. We must then make r_1 converge to 0 and for that backstepping technics are used.

Backstepping for r1:

$$\begin{aligned}\dot{r}_1 &= \dot{v} - \dot{v}_{d2} \\ \dot{r}_1 &= S(r_1)\omega + \begin{bmatrix} 0 \\ 0 \\ -1/m \end{bmatrix} u_1 - R^T \ddot{p}_d(\gamma) + k_e(v - R^T \dot{p}_d(\gamma) v_{d2}) - R^T \dot{p}_d(\gamma) z\end{aligned} \tag{4.33}$$

The new error variable r_1 can not always be driven to zero. To derive an expression for the virtual control inputs a matrix of values must be inverted, what happens only if it is made full rank by driving the error variable r_1 to a constant design vector δ. A new error variable Φ is then defined and, after that, a new, composite, Lyapunov function created.

$$\Phi = r_1 - \delta \tag{4.34}$$

$$\begin{aligned}V_2 &= V_1 + \frac{1}{2}\Phi^T\Phi \\ &= \frac{1}{2}e^T e + \frac{1}{2}\Phi^T\Phi\end{aligned} \tag{4.35}$$

Taking the time derivative of V_2, we get

$$\begin{aligned}\dot{V}_2 &= \dot{V}_1 + \Phi^T\Phi \\ &= -k_c e^T e + e^T\delta + \Phi(B(.)\zeta - R^T\ddot{p}_d(\gamma) + k_e(v - R^T\dot{p}_d(\gamma)v_{d2}) + e) \\ &\qquad - (\Phi^T + e^T)(R^T\dot{p}_d(\gamma))z\end{aligned} \tag{4.36}$$

$$\mathrm{B}(.)\zeta = \begin{bmatrix} 0 & \vdots & 0 & -\delta_3 & \delta_2 \\ 0 & \vdots & \delta_3 & 0 & -\delta_1 \\ -\frac{1}{m} & \vdots & -\delta_2 & \delta_1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{4.37}$$

More details on the backstepping derivation in equation (4.33) and on the derivation above in [12].

Note that ζ is our control input vector used to enforce $\dot{V}_2$ to be negative.

To do that, a proper expression for ζ is presented in equation (4.38).

$$\zeta = B^T(BB^T)^{-1}(-e - R^T\ddot{p}_d(\gamma) + k_e(v - R^T\dot{p}_d(\gamma)v_{d2}) - k_\Phi\Phi), \quad k_\phi > 0 \tag{4.38}$$

To get the control law for the path parametric variable, the existing Lyapunov function is modified to equation (4.39).

$$V_3 = V_1 + V_2 + \tfrac{1}{2}z^T z \tag{4.39}$$

With its derivative

$$\dot{V}_3 = \dot{V}_1 + \dot{V}_2 + r_\gamma(\ddot{\gamma} - v_d) \tag{4.40}$$

$$\begin{aligned}\dot{V}_3 = -k_e e^T e + e^T \delta + \Phi(B(.)\zeta - R^T \ddot{p}_d(\gamma) + k_e(v - R^T \dot{p}_d(\gamma) v_{d2}) + e) \\ -z(R^T \dot{p}_d(\gamma)(\Phi^T + e^T) - \ddot{\gamma})\end{aligned} \tag{4.41}$$

Lastly, an expression for $\ddot{\gamma}$ is derived to make $\dot{V}_3$ negative definite, as follows

$$\ddot{\gamma} = R^T \dot{p}_d(\gamma)(\Phi^T + e^T) - k_\gamma z \tag{4.42}$$

where k_γ is a positive constant.

The equations (4.38) and (4.42) are the control laws necessary to make our quadcopter follow the desired path.

4.2 Cooperative Path Following

Let $N = \{1, 2, \ldots, n\}$ be the set that represents a fleet with n vehicles, and $N_i = \{1, 2, \ldots, m\}$ be the set of the ith vehicle's neighbors, with m as the maximum number of elements of this set.

Assume that each vehicle converges to its respective virtual target point, and that this virtual point travels along the desired path at the desired speed V_d.

If all vehicles reach the virtual target point, which is, in another words, all γ have the same value, consensus is achieved and they are in formation.

In the previous chapter, control laws for path following were derived. These laws make the vehicle follow a desired path parameterized by γ, and its value converge to V_d over time.

In order to make each vehicle's γ have the same value, we must adjust the value of V_d accordingly.

Assume that each vehicle $i \in N$ has access to the variables γ_i and γ_j, $j \in N_i$. A control law for V_d must be derived, such that, $(\gamma_i - \gamma_j) \to 0$ as $t \to \infty$.

The desired speed for each vehicle of the fleet is then described by

$$V_{d\_i} = V_{formation} + V_{correction} \tag{4.43}$$

$V_{formation}$ is the desired flock speed, and is defined when the controller is initialized.

In subchapter 2.2.4, the agreement protocol was presented, as well as equation (2.12), that represents the dynamic of a node's state. With γ_i being the state we want to be controlled, $V_{correction}$ is defined by the following equation.

$$V_{correction} = \frac{1}{m} \sum_{j \in N_i} (\gamma_j(t) - \gamma_i(t)) \tag{4.44}$$

where $\frac{1}{m}$ is a gain. In practice, it calculates the average weight that the correction speed must have in each of the vehicles.

Equations (4.43) and (4.44) are the control laws that solve our cooperative path following problem.

Chapter 5

Simulation Results

The objective of this chapter is to show the results of the simulations done in Matlab using a toolkit called Virtual Arena. The goal of the first simulation is to show us the proposed controller working on a 2D case scenario. In the second simulation, the path following and coordination capabilities of our drone flock are demonstrated in a 3D environment.

For each simulation, the desired and actual position will be compared, the values of $\dot{\gamma}$, the velocities and the absolute position error will be shown and analyzed.

All simulations used a fixed sample time interval of 1ms. The desired speed was equal to 1 for the single-agent scenarios, and 0.75 for the flock scenarios.

The desired trajectories will be described in each subchapter but are all based on circumferences parallel to the ground.

5.1 Unicycle Flock

As it has been done throughout this document, the simulations' phase starts with the simplest example: a unicycle flock. More concretely, by the simulation of the path following controller and lately the cooperative path following one.

In chapter 3, it was told that our vehicles have limitations. In the unicycle case, the limitations are

$$-2m/s \leq v \leq 2m/s \tag{5.1}$$

$$-1.57rad/s \leq \omega \leq 1.57rad/s \tag{5.2}$$

For both cases, $\varepsilon = \begin{bmatrix} -0.5 & 0 \end{bmatrix}$.

5.1.1 Path Following

5.1.1.1 Desired Path

The desired path for this example is a circumference with a 10m radius centered in origin, that is,

$$x_d = 10\sin(\gamma) \tag{5.3}$$

$$y_d = 10\cos(\gamma) \tag{5.4}$$

5.1.1.2 Initial Conditions and Gains

The initial position of the vehicle is the origin of the referential, and its orientation starts in 0 rads.

The gains have the following values.

$$K_e = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

$$K_\gamma = 10$$

5.1.1.3 Simulation Results

The following figure shows both the desired and actual position of the unicycle during the simulation.

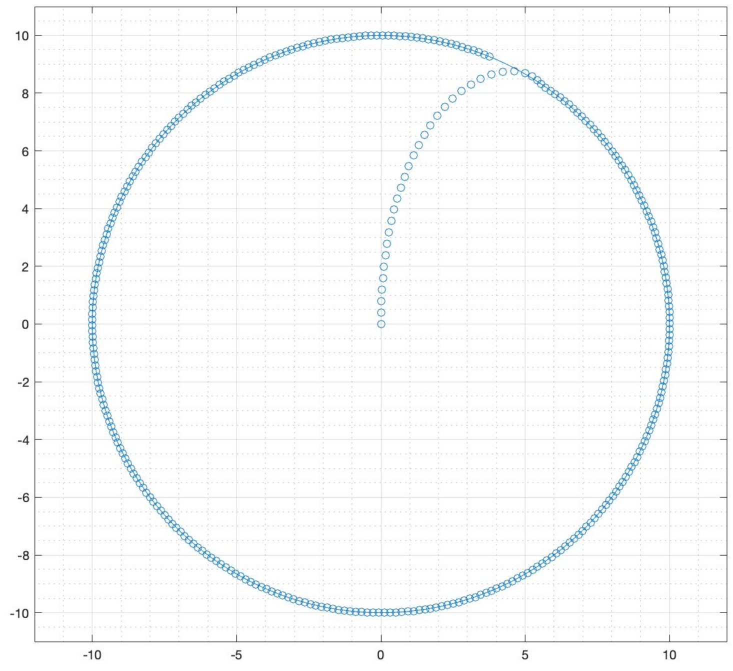


Figure 5.1: Actual (circles) and desired (line) positions of the unicycle (m)

As expected, the vehicle starts in the origin and goes to the desired circumference in just a few seconds. It remains there as long as the simulation runs.

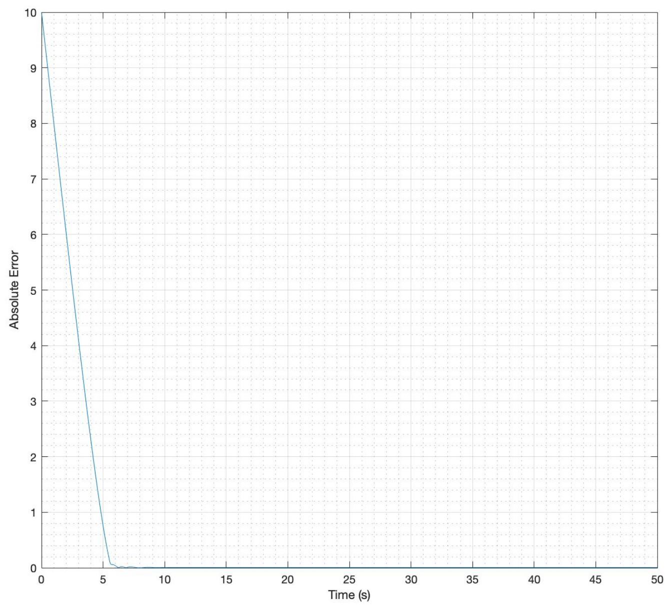
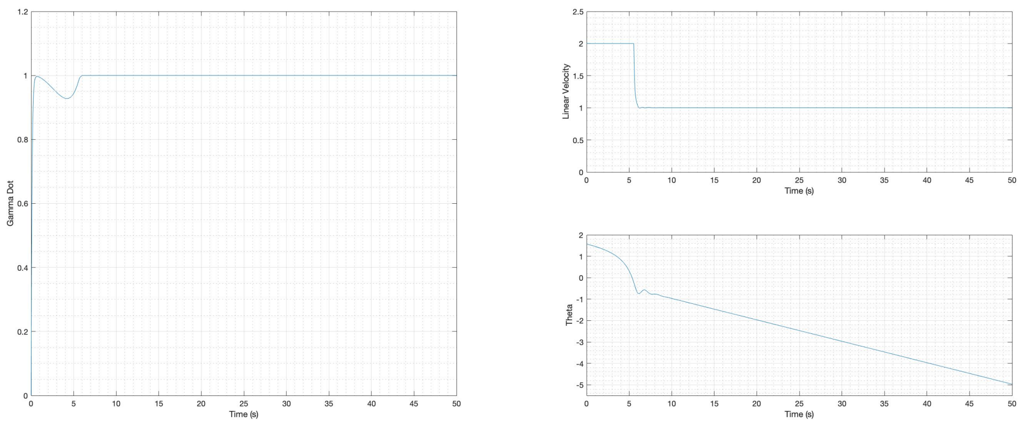


Figure 5.2: Absolute position error of the unicycle (m)

In figure 5.2, we can see the absolute position error of the vehicle. It starts around 10 meters, which matches the radius of the desired path, and converges to zero. It is not equal to zero due to the value of ε, and it will be closer to zero as smaller the value of ε is.



(a) Value of $\dot{\gamma}$

(b) Linear velocity (m/s) and angle (rad) of the unicycle

Figure 5.3: Simulation results for unicycle's path following controller

After analyzing figure 5.3 a), we conclude that this controller can make our vehicle follow the desired path at the desired speed of 1 m/s. We can also see, in figure 5.3a) , that it reaches constant velocities, both linear and angular by making θ grow linearly over time.

5.1.2 Cooperative Path Following

After proving that our controller is suitable for path following, its time to add the coordination part to the controller and see how it works on a multi-agent scenario.

5.1.2.1 Desired path

For a fleet with N vehicles, the desired trajectories are circumferences centered in origin, starting at 15 meters of radius and spacing two meters between each other, that is,

$$x_d = (15 + (2(n-1)))\sin(\gamma) \tag{5.5}$$

$$y_d = (15 + (2(n-1)))\cos(\gamma) \tag{5.6}$$

5.1.2.2 Initial Conditions and Gains

Equations (5.7) and (5.8) give the vehicles' initial positions, and its orientations were the same for all vehicles, $pi/2$ rads, or 90^o degrees.

$$x_0(n) = (n-1) - 2(n-1) \tag{5.7}$$

$$y_0(n) = 5 \tag{5.8}$$

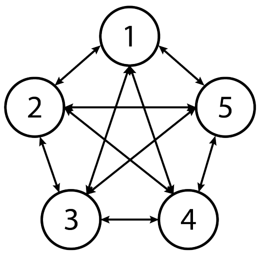
This center all vehicles around y-axis spaced by two meters in x-axis direction.

The gains have the following values.

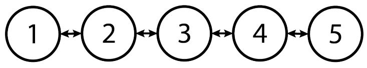
$$K_e = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$$

$$K_\gamma = 10$$

5.1.2.3 Network Configurations



(a) Configuration 1



(b) Configuration 2

Figure 5.4: Diferent network configurations for the unicycle simulations

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{5.9}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.10}$$

For the flock simulations, two different network configurations were used. Both configurations are represented in the figure 5.4. The respective adjacency matrices are presented right after that.

In the first configuration, represented by figure 5.4 a) and matrix (5.9), all agents can change information inside the network. In this chapter, this is called *Configuration 1*. Then, for the second simulation, represented by figure 5.4 b) and matrix (5.10), each vehicle can only communicate with its closest neighbor. *Configuration 2*, is its name in this chapter from now on.

5.1.2.4 Simulation Results - Configuration 1

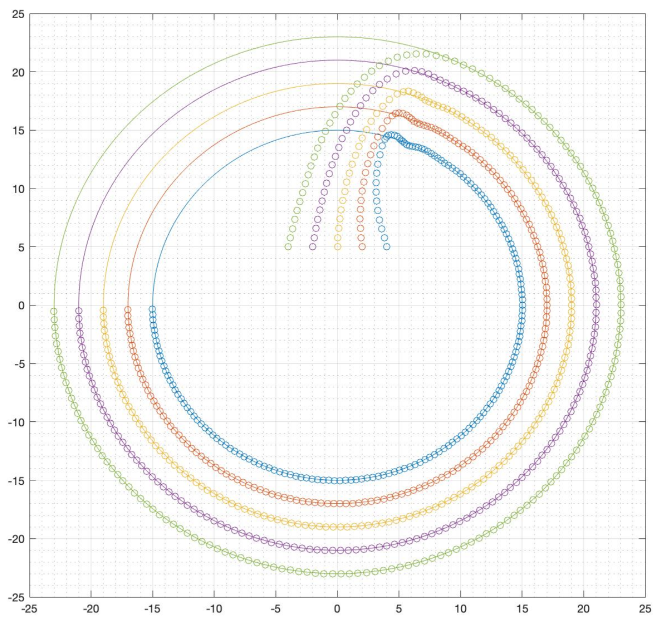
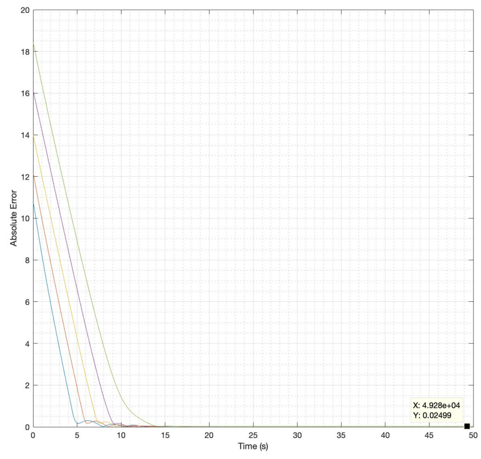


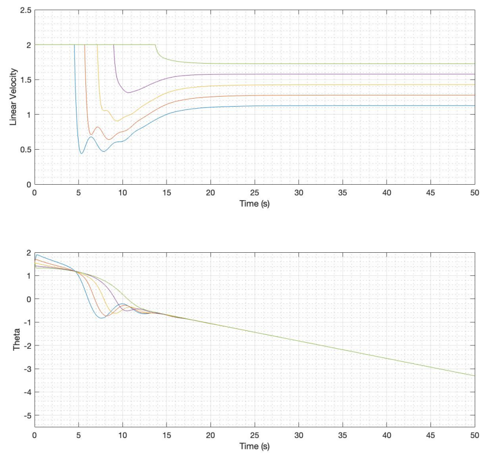
Figure 5.5: Actual (circles) and desired (line) positions of the unicycle flock (m) - configuration 1

This simulation was executed in the best scenario possible for achieving consensus, where all agents have access to the entire set of network's γ.

Figure 5.5 shows the desired and actual positions of the unicycle fleet during the simulation, and it is confirmed that all vehicles drive themselves to the desired path as wanted.



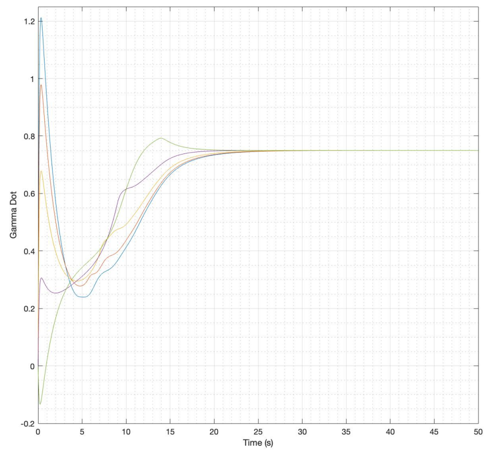
(a) Absolute position errors (m)



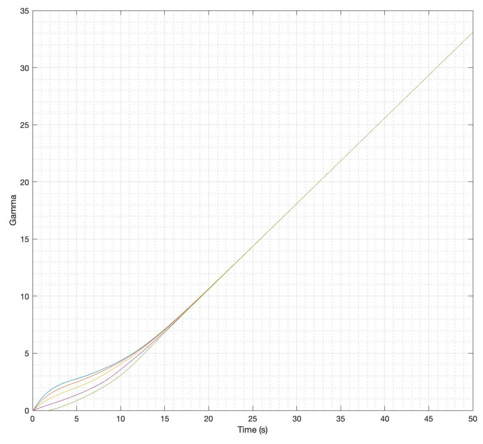
(b) Linear velocity (m/s) and angle (rad) of the unicycles

Figure 5.6: Simulation results for unicycle's cooperative path following controller - configuration 1

The absolute error of each unicycle (figure 5.6 a)) starts on expected values and converges to zero as time goes on. Figure 5.6 b), demonstrate how the wider trajectories require more speed than the tighter ones to keep the formation speed. Also, as soon as they are in coordination, their angular speeds are constant and their θ have all the same value.



(a) Value of all $\dot{\gamma}$



(b) Value of all γ

Figure 5.7: Values of $\dot{\gamma}$ and γ of all agents of the network - configuration 1

The plot in figure 5.7 a) shows the controller adjusting the speed of the parametrization variable of each vehicle accordingly to the coordination rule in order to keep the formation coordinated and tracking the desired path. After coordination is achieved all γ converge to the same value, as required in equation (4.3). By looking at those pictures we can conclude that our system achieves the desired flock speed.

5.1.2.5 Simulation Results - Configuration 2

The second simulation for the unicycle flock scenario has some communication limitations as explained in subchapter 5.1.2.3. Each unicycle only knows the γ value of its direct neighbors.

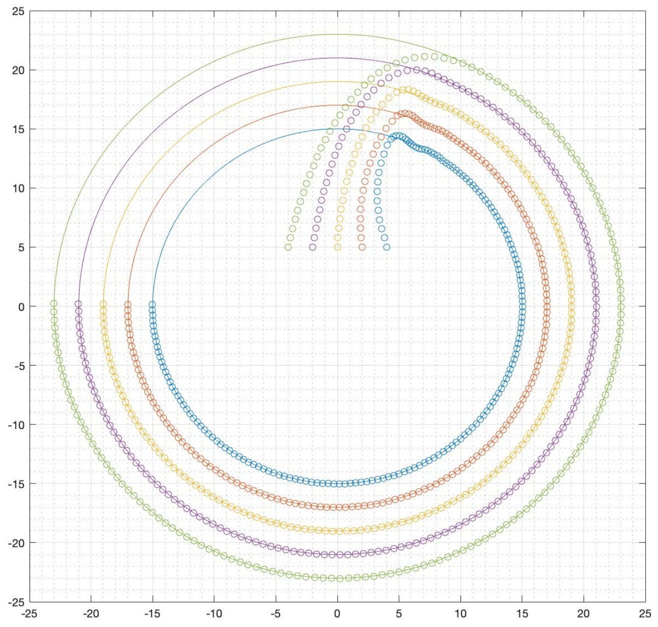
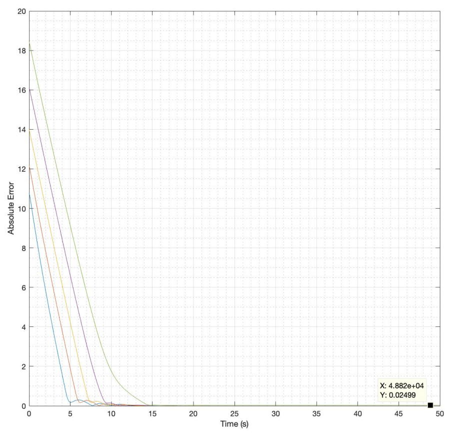


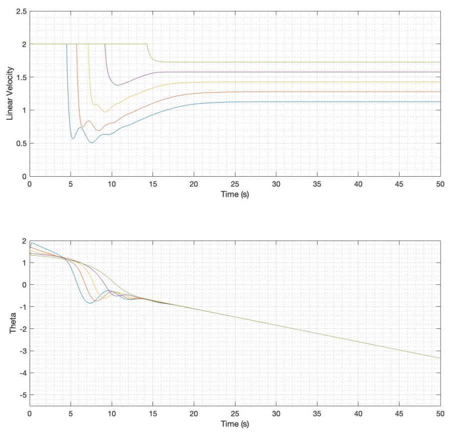
Figure 5.8: Actual (circles) and desired (line) positions of the unicycle flock (m) - configuration 2

The results of the simulation with configuration 2 are quite similar to the one with the network in configuration 1. For the sake of avoiding repetitions, only the most notable differences are commented.

As in the first scenario, all vehicles go to the desired trajectories over time. However, trajectories are not the same as before.



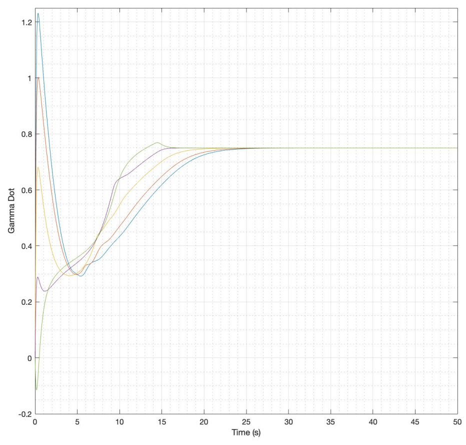
(a) Absolute Position Errors (m)



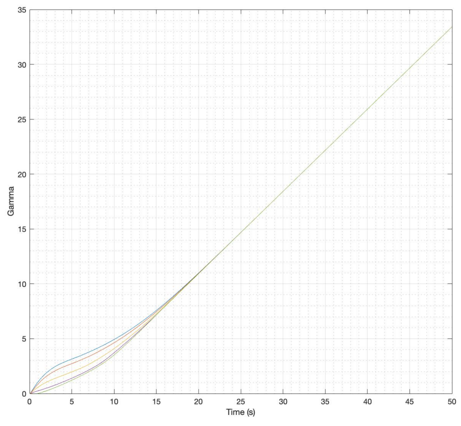
(b) Linear velocity (m/s) and angle (rad) of the unicycles

Figure 5.9: Simulation results for unicycle's cooperative path following controller - configuration 2

The difference in unicycle's positioning error is easily understandable by looking at the absolute errors' value, figure 5.9 a). Compared to the last scenario, the wider is the drone's path, or in other words, the higher is its number, the biggest is the delay until it reaches zero error. It is easily noticeable in the green vehicle absolute error line where the time it takes to reach zero is more than one second that it was before. It happens because the dynamic of each agent is now depending on only one, or two, neighbors. It is expected that unicycles at the ends of the formation take more time until reach the formation desired speed.



(a) Value of all $\dot{\gamma}$



(b) Value of all γ

Figure 5.10: Values of $\dot{\gamma}$ and γ of all agents of the network - configuration 2

In figure 5.9 b), can be noticed that changes of speed happens later in this configuration than in the one simulated before. The same happens in the intensity of its changes that is smaller than before. It happens because the smaller number of neighbors leads, in this case, to a smaller coordination error and so that minor adjustments than before. Each drone is correcting its point in the path accordingly to only its closest neighbor.

This makes the flock take a bit longer to achieve coordination, as can be seen in figure 5.10.

5.2 Drone Flock

Simulations of the 3D scenarios are done in this subchapter. Again, it starts with the simulation of the path following controller and then with the cooperative path following one.

For both simulations, the drones' mass is 650 grams and the gravity acceleration is equal to 9.81m/s.

The following expressions describe the input limitations of the drone that were imposed.

$$0 \leq Thrust \leq 20m/s \tag{5.11}$$

$$-1.57rad/s \leq \omega_1 \leq 1.57rad/s \tag{5.12}$$

$$-1.57rad/s \leq \omega_2 \leq 1.57rad/s \tag{5.13}$$

$$-1.57rad/s \leq \omega_3 \leq 1.57rad/s \tag{5.14}$$

For both cases, $\varepsilon = \begin{bmatrix} 0 & 0 & 0.5 \end{bmatrix}$.

5.2.1 Path Following

5.2.1.1 Desired path

The desired path for this example is a circumference with a 10m radius centered in origin, 5 meters high off the ground.

$$x_d = 10\sin(\gamma) \tag{5.15}$$

$$y_d = 10\cos(\gamma) \tag{5.16}$$

$$z_d = 5 \tag{5.17}$$

During all flights, the controllers were instructed to keep the drones all pointing to the same starting direction, which means keep the yaw angle always at 0 rads.

5.2.1.2 Initial Conditions and Gains

The vehicle's starting position is the origin of the referential, and its orientation starts in 0 rads.

The gains for the controller have the following values.

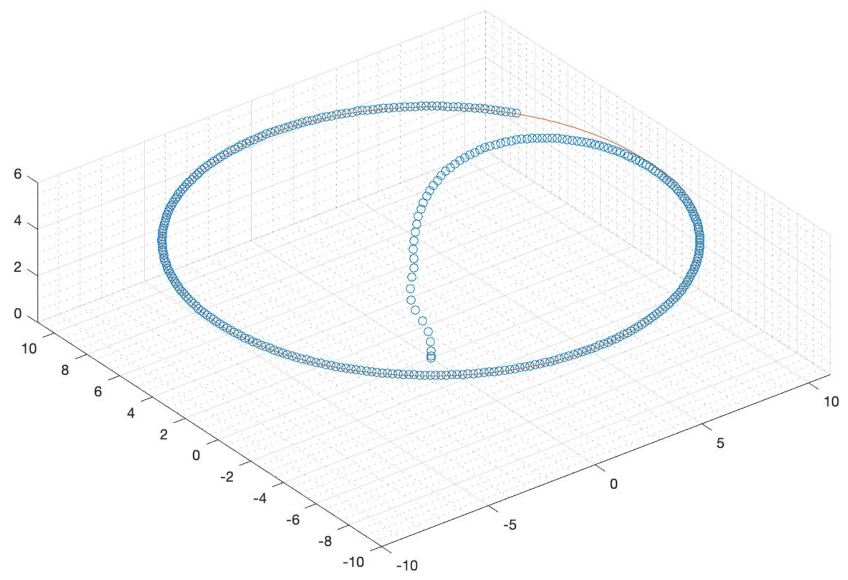
$k_e = 5$

$k_\phi = 0.1$

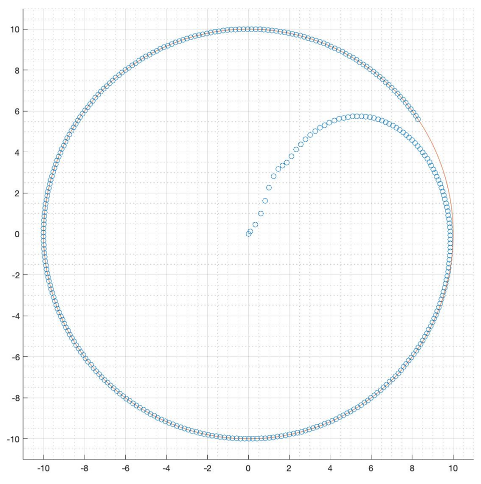
$k_\psi = 5$

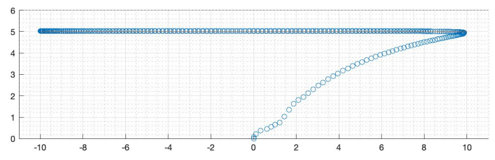
$k_\gamma = 10$

5.2.1.3 Simulation Results



(a) 3D View





(b) Top View

(c) Front View

Figure 5.11: Actual (circles) and desired (line) positions of the drone (m)

As figure 5.11 shows, the controller presented to the 3D path following problem works as it should. The drone starts the simulation in the origin and drives itself to the desired path until its absolute position error gets almost zero, which can be verified in the next figure.

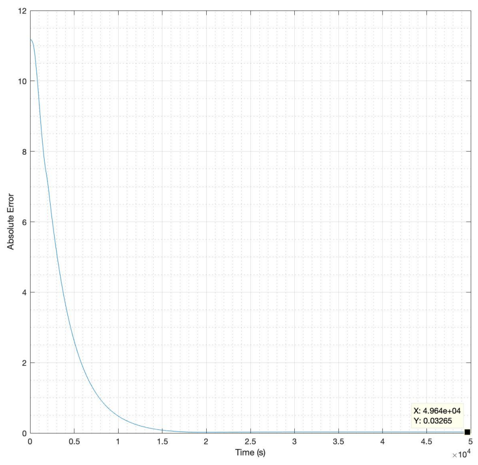
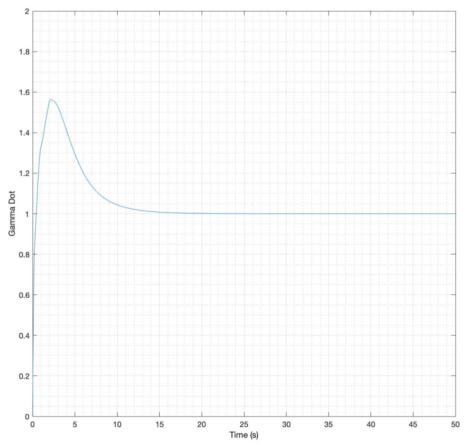
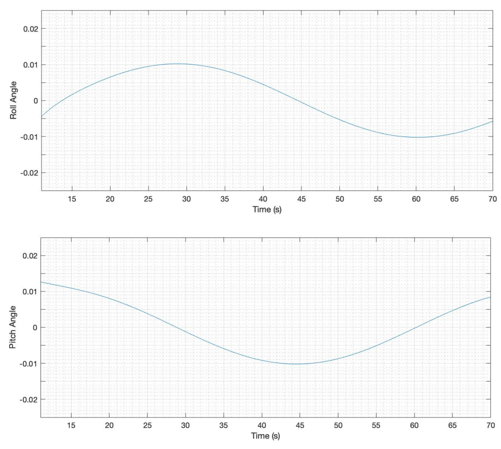


Figure 5.12: Absolute position error of the drone (m)

Regarding its speed behavior, we must look to the plots of $\dot{\gamma}$. If this value leads to the desired speed, then our drone is following the speed assignment as desired. In this simulation, the desired speed value is constant and equal to 1m/s. After checking figure 5.13 a), we verify that $\dot{\gamma}$ tends to the given desired speed and the path following task is correctly done by our controller.



(a) Value of $\dot{\gamma}$



(b) Roll and pitch angles of the drone

Figure 5.13: Values of $\dot{\gamma}$, roll and pitch of all agents of the network

In figure 5.13 b) are presented the roll and pitch angle values from a few seconds before the drone reaches the desired path. The desired yaw angle for all simulations is 0 rads, so, by looking at its roll and pitch angles we can find some characteristic points in its path, more precisely, when drones are in a point of the path where the tangent to it is parallel to the x-axis or perpendicular to the y-axis.

When body's x-axis is parallel to the inertial x-axis, it is only using pitch to move forward or backward, depending if its y coordinate is positive or negative. It happens for time around 44.5 seconds, where the roll angle is equal to 0. Accordingly to what was said above, this point must be a local maximum (or minimum) for the pitch angle value, which is verified. The opposite happens for $t = 60$s, when the drone has its y-axis parallel to inertial's y-axis.

5.2.2 Cooperative Path Following

Now we get to the main goal of this dissertation. A cooperative path following controller to manage a flying drone flock.

The flock used in this simulation has 6 drones (N = 6).

As did in the unicycle example, first we tested the path following ability of our controller then we add the coordination controller to the equation.

5.2.2.1 Desired path

For this simulation, a fleet of 6 drones was used. The desired trajectories are divided into two height levels, 5 and 10 meters, and in both levels, the trajectories are circumferences centered in origin and with 20, 25 and 30 meters radius. These trajectories are detailed next:

$$x_d(n) = \begin{cases} (20+(5(n-1)))\sin(\gamma), & 1 \le n \le 3 \\ (20+(5(n-4)))\sin(\gamma), & 4 \le n \le 6 \end{cases} \tag{5.18}$$

$$y_d(n) = \begin{cases} (20+(5(n-1)))\cos(\gamma), & 1 \le n \le 3 \\ (20+(5(n-4)))\cos(\gamma), & 4 \le n \le 6 \end{cases} \tag{5.19}$$

$$z_d(n) = \begin{cases} 5, & 1 \le n \le 3 \\ 10, & 4 \le n \le 6 \end{cases} \tag{5.20}$$

5.2.2.2 Initial Conditions and Gains

The drones started on the ground (z=0), three on them in front, aligned with the desired path, and the remaining three, 5 meters beyond them. More precisely,

$$x_0(n) = \begin{cases} 0, & 1 \le n \le 3 \\ 5, & 4 \le n \le 6 \end{cases} \tag{5.21}$$

$$y_0(n) = \begin{cases} 20+5(n-1), & 1 \le n \le 3 \\ 20+5(n-4), & 4 \le n \le 6 \end{cases} \tag{5.22}$$

The gains have the following values.

$k_e = 5$

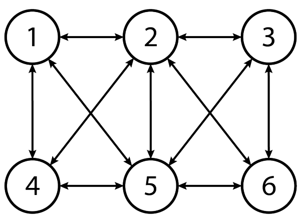
$k_\phi = 0.1$

$k_\psi = 5$

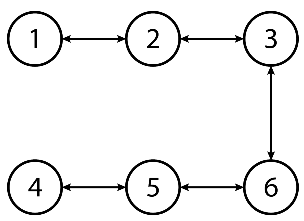
$k_\gamma = 10$

5.2.2.3 Network Configurations

For the unicycle flock simulation, two different network configurations were used. In the first configuration, all agents can change information inside the network. Then, for the second simulation, each vehicle can only communicate with its closest neighbor.



(a) Configuration 1



(b) Configuration 2

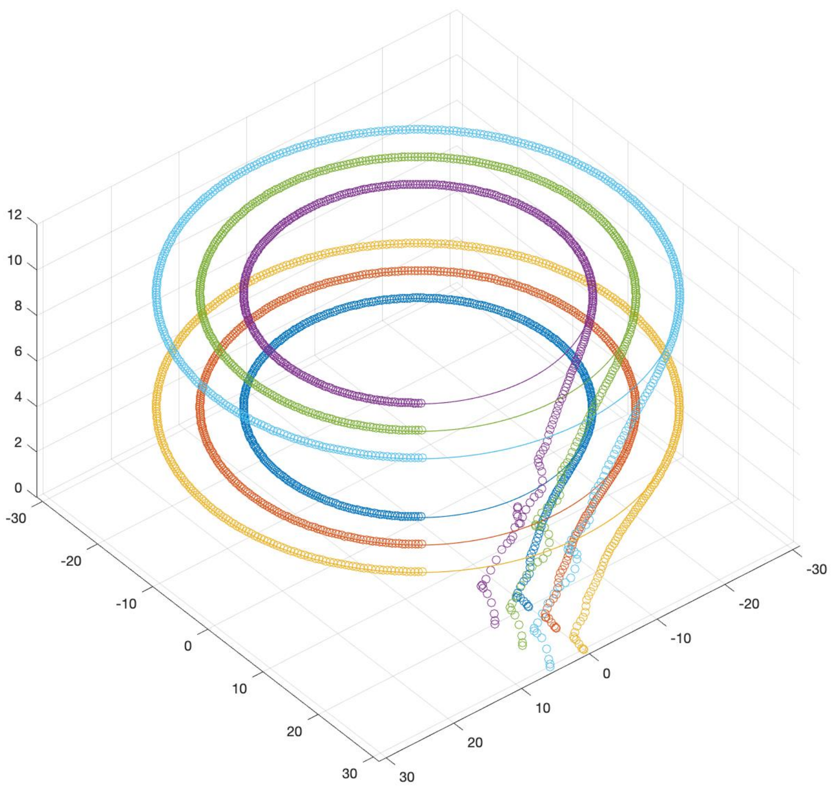
Figure 5.14: Diferent network configuration for the drones simulations

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{5.23}$$

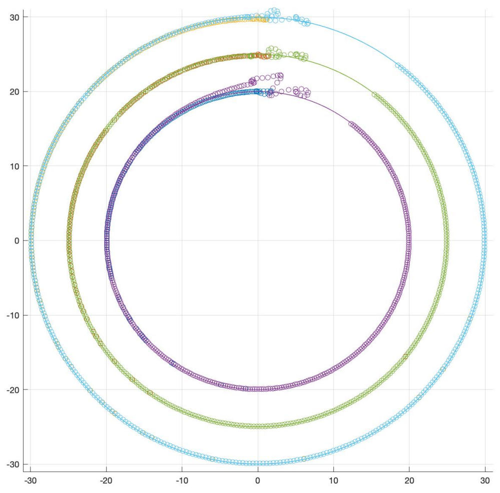
Both configurations are represented in the graphs above. The respective adjacency matrices are presented right after that.

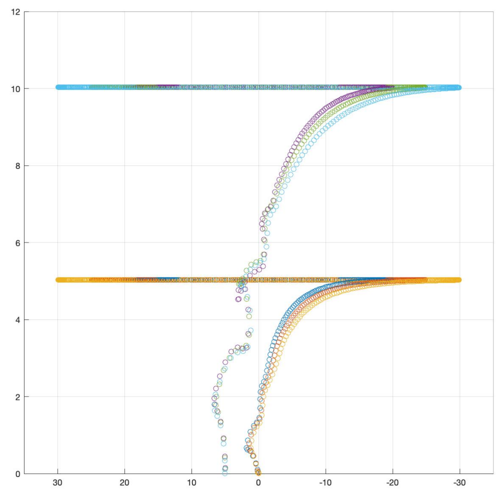
5.2.2.4 Simulation Results - Configuration 1

The simulation of a flock in the first configuration is here presented. Figure 5.15 illustrates the behaviour of the flock in this simulation. In figure 5.15 a), the drones reached the desired trajetory and are proper alligned as desired. Figures 5.15 b) and c) , are the top and front views, respetively, of their trajectories.



(a) 3D view





(b) Top view

(c) Front view

Figure 5.15: Actual (circles) and desired (line) positions of the drone flock (m) - configuration 1

To confirm the previous statement, the absolute position error is presented in figure 5.16. As observed in the previous figures, The value of the absolute position error converges and remains closer to zero over time.

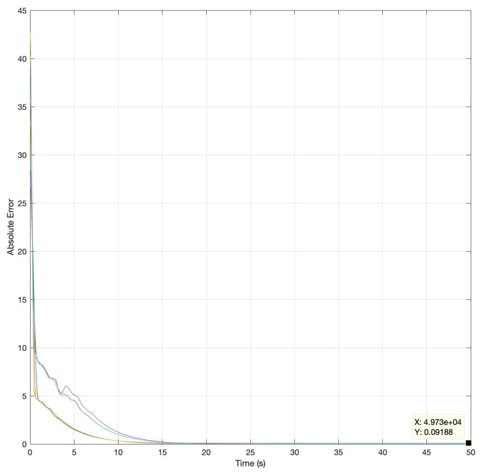
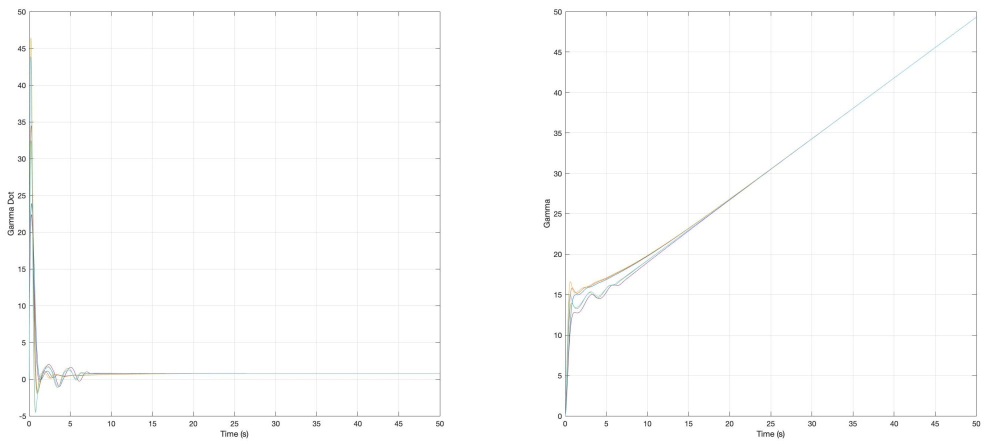


Figure 5.16: Absolute position errors (m) - configuration 1

Looking to γ and $\dot{\gamma}$ values, we can see that consensus has been reached. All γ converged to the same value, as well its speed, that ended up around 0.75, as required.

The drones from the top trajectories had their γ value lower than the drones at the beggining, what is expected once they take more time to reach a farthest path. This extra effort to reach the formation desired speed is reflected in the $\dot{\gamma}$ values that were changing more abruptly.



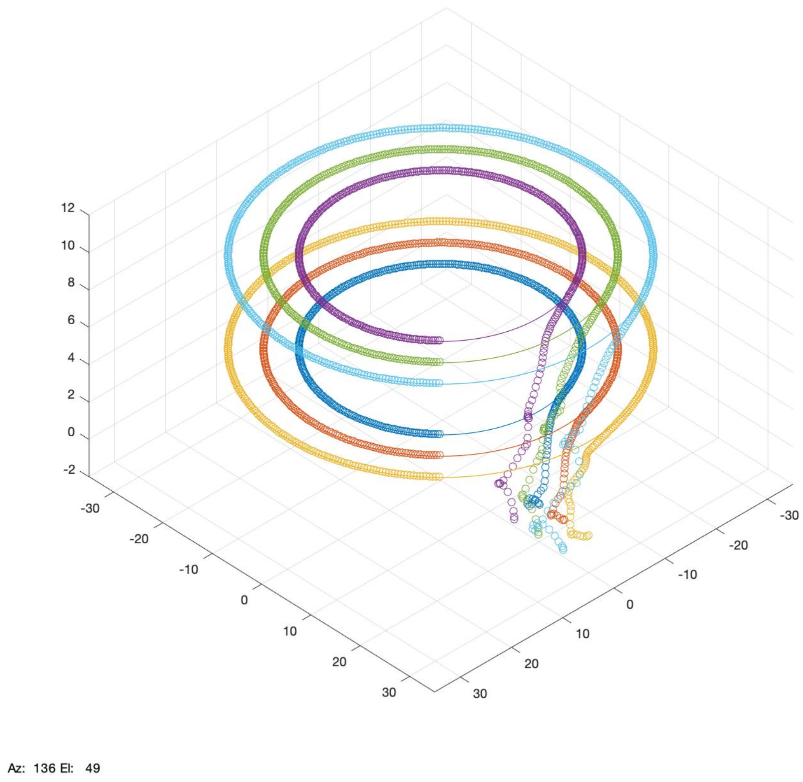
(a) Values of $\dot{\gamma}$

(b) Values of γ

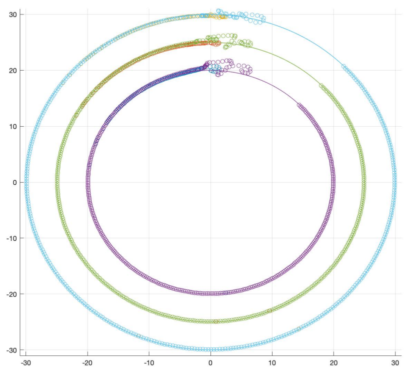
Figure 5.17: Values of $\dot{\gamma}$ and γ of all agents of the network - configuration 1

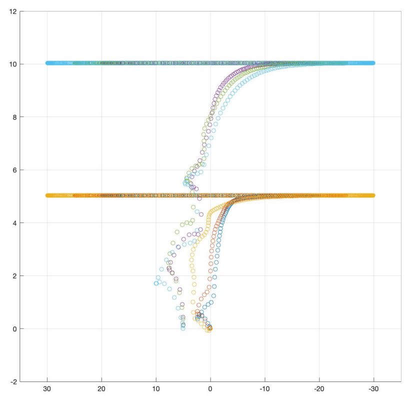
5.2.2.5 Simulation Results - Configuration 2

The second simulation had a few changes in the way the communication network was set up. As explained in subchapter 5.2.2.3, for this simulation drones can only change information with only those drones that are right next to them, in the same level. The only drones that can communicate between levels are the ones at the wider path, 3 and 6, accordingly to figure 5.14 b).



(a) 3D view





(b) Top view (c) front view

Figure 5.18: Actual (circles) and desired (line) positions of the drones (m) - configuration 2

When the absolute position error, represented in figure 5.19, is compared with the previous one (figure 5.16), it is noticeable the difference between both scenarios. By grouping the network into two smaller groups (top and bottom trajectories) the flock takes more time to reach the desired position but the positioning error is very similar among each group's elements.

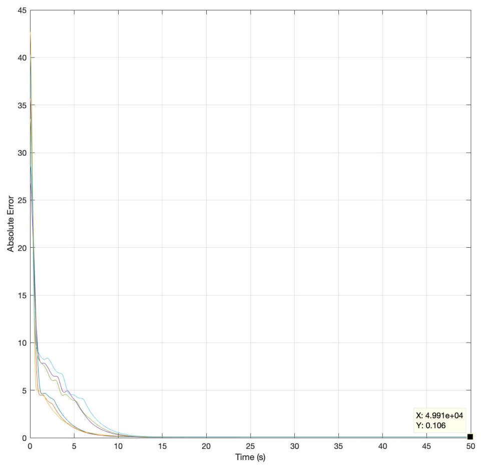
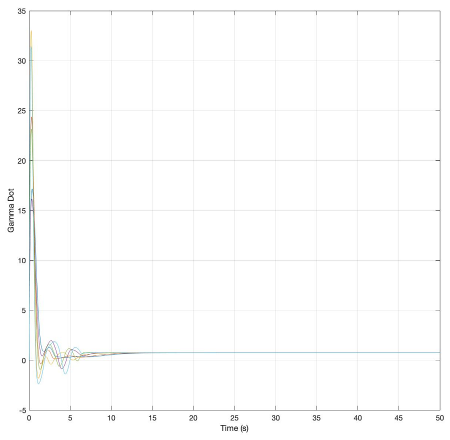
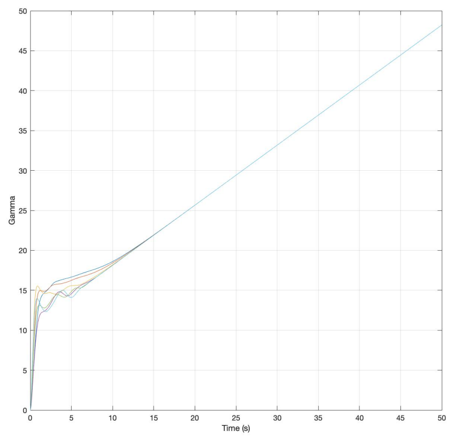


Figure 5.19: Absolute position error (m) - configuration 2

As noticed in the case of the unicycle flock, later and more intense changes of speed are noticeable. This flock also takes more time to reach consensus than the previous one using Configuration 1.



(a) Values of $\dot{\gamma}$



(b) Values of γ

Figure 5.20: Values of $\dot{\gamma}$ and γ of all agents of the network - configuration 2

Chapter 6

Conclusions

In this dissertation, a Lyapunov-based control algorithm was proposed and analized. This algorithm implemented a cooperative path following controller that made a group of robots and a flock of drones follow a desired path and achieve consensus.

This work started with the design of the simpler, 2 dimentional, cooperative path following controller for a unicycle type robot flock. The cooperative path following controller for the drone flock was done next.

This controller dealt with the path following problem by making drones follow a desired virtual point parametrized by a path parametric variable. Then adjusted the desired parametric variable speed of each drone in order to keep them in formation.

Six simulations were presented. Three for the robot flock scenario and the other three with the drones. Both type of vehicles had a simulation to verify its path following controller. For both cases, it worked as desired and the controllers proved to be efective and robust. Its robustness was once again demonstrated when we added more vehicles to the simulation and the consensus controller. This time the cooperative path following controllers were tested and, once again, the results were satisfatory. Two different network configuration were tested, the first one where all agents had access to the entire network positions, and the second one, more realistic, where drones could only communicate with its closest neighbors. Once again, robustness was demonstrated and showed that flock control can be achieved even when only two agents are connecting an entire multi-agent system.

The cooperative path following controllers obtained accomplished the desired goals of this dissertation but future work must be done in order to make our model more realistic and the controller more suitable to that model, such as:

- Drone's inertia and drag must be added to the dynamic equations of the model.
- Collision avoidance with the ground, other drones and obstacles.

Bibliography

[1] Craig W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model", Julho 1987

[2] Tamás Vicsek et al., “Novel type of phase transition in a system of self-driven particles”, 1995

[3] Veysel Gazi, "Stability Analysis of Swarms", 2002

[4] Herbert G. Tanner, "Stable Flocking of Mobile Agents, Part I: Fixed Topology", 2003

[5] Reza Olfati-Saber, “Flocking for multi-agent dynamic systems: algorithms and theory”, 2006

[6] Raffaello D’Andrea, "Meet the dazzling flying machines of the future" - TED2016, 2016 (Visualizado em Novembro, 2019)

[7] Cirque du Soleil, ETH Zurich, and Verity Studios, "SPARKED: A Live Interaction Between Humans and Quadcopters, 2014

[8] Silva, F., "Eyes in the sky: multi-drones surveillance technology", Fevereiro 2017.

[9] Yao P, Wang H. e Su, Z., "Cooperative path planning with applications to target tracking and obstacle avoidance for multi-UAVs", Abril 2016.

[10] Vanni, F., Aguiar, A. e Pascoal, A., "Cooperative Path-Following of Underactuated Autonomous Marine Vehicles with Logic-based Communication", February 2017.

[11] Cichella, V., Kaminer, I., Dobrokhodov, V., Xargay E., Choe, R., Hovakimyan, N., Aguiar A. e Pascoal A. "Cooperative Path Following of Multiple Multirotors over Time-Varying Networks", Julho 2017

[12] Rodrigues R. e Aguiar, A., "Cooperative Motion Control of a Formation of UAVs", Novembro 2018

[13] A. Alessandretti, A. P. Aguiar, C. N. Jones, "VirtualArena: An Object-Oriented MATLAB Toolkit for Control System Design and Simulation", 2017

[14] https://github.com/andreaalessandretti/VirtualArena (Visualizado em Dezembro 2019)

[15] M. Mesbahi and M. Egerstedt, "Graph Theoretic Methods in Multiagent Networks", 2010

[16] R. Praveen Jain, A. Pedro Aguiar, and J. Borges de Sousa, "Cooperative Path Following of Robotic Vehicles using an Event based Control and Communication Strategy"

[17] R. Hammond, "Motion Control of an Autonomous Helicopter - A Lyapunov based approach to Controller Design", May 2007

[18] A. Pedro Aguiar, Member, IEEE, and João P. Hespanha, IEEE, "Trajectory-Tracking and Path-Following of Underactuated Autonomous Vehicles with Parametric Modeling Uncertainty"

[19] Jingyuan Zhan, Xiang Li, "Flocking of Discrete-time Multi-Agent Systems with Predictive Mechanisms", Setembro 2011

[20] Quan Yuan, Jingyuan Zhan, Xiang Li, "Outdoor flocking of quadcopter drones with decentralized model predictive control", Julho 2017