POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Modeling and Consensus-based Control of Multi-Agent Systems

Supervisors

Prof. Juan-Antonio ESCARENO

Prof. Elisa CAPELLO

Candidate

Giuliana ANSELMO

16 April 2021

Abstract

Nowadays, Unmanned Aerial Vehicles (UAVs) applications are crucial in several domains, either civilian or military as these systems are mainly used within hostile environments while ensuring user integrity or industrial and academic sectors while coping with specific needs and services.
For instance, UAVS are significantly used in the field research missions, as for example volcanic-activity's monitoring, climate change studies, crowds-management, management/surveillance and epidemiological tracking. Multi-Agent Systems (MASs) represent a trending topic in the UAVs field since the related literature shows that many applications are enhanced. Indeed, MAS is an ensemble of individual agents sharing some information to achieve a collective objective, as the cooperation could enhance the effectiveness of the mission accomplishment with respect to the action of a single agent.
Collective behaviour implies an efficient inter agent's and surrounding environment information flow (topology) and a coordination protocol must shape the formation, the agreement value and the synchronization strategy for completing the mission. Many examples can be found in nature, for example in migratory birds, forming a network of collaborating entities that fly together in different shaped coordinated formations (flocking) or other species that agree about an interest variable (motion, communication, ...) to enhance mutual benefits, that include predators' survival, collective foraging, long distance migration, and, thus, the acquirement of more dynamic efficiency as the main advantage.
In particular, the flocking behaviour is a one of the key aspects of the MASs development and implementation, including the interaction and communication between the elements of a group, the perception that each entity has of the whole flock (or sub-group of the flock) and of each flock mate, the detection of external disturbances and obstacles.
Despite the evolving trending topic and the promising MAS applications, numerous technological and scientific challenges are still to be addressed and solved: firstly the efficient definition of a communication protocol, the implementation on real UAVs, subjected to several software and hardware failures.
The main focus of this thesis is the MAS model definition for consensus problems. In detail, by efficiently representing the system by graphs, the consensus algorithms are verified in MATLAB&Simulink environment and completed including the knowledge of the leader of the desired trajectory. A centralized-based model is firstly proposed, but a distributed model is the target of the work.
Different formations are compared (e.g. spatial placement, sensors' sensitivity and detection area, addition of obstacles, ...) in terms of observability and

controllability, aiming to the identification of the topologies that allow to reconstruct the initial states by implementing an observer and, afterwards, a PID controller. At last, the model are experimented on real MASs together with the initialization of a new environment for UAVs wherein basic flight tasks will be tested.
In the future, the model will be expanded by considering topologies changing overtime, such as the addition/removal of nodes or by considering the efficient action of the formation when avoiding an obstacle or going through a narrow passage and by an in-depth application to the real environment, with the option of estimating external disturbances.

Acknowledgements

A special thanks to my supervisors that gave me the possibility to partecipate to this project and helped me.
I would like to express my gratitude to my family, my parents and Daria, that supported me during these years and allowed me to follow my dreams. Thanks to all my friends, who stood by me when I was far from my home and helped me in my personal development.
I want to thank Michele, that was with me in the hardest moments and contributed in the best moments.

"Non basta guardare, occorre guardare
con occhi che vogliono vedere,
che credono in quello che vedono."
Galileo Galileo

Table of Contents

List of Tables

List of Figures

Acronyms

Ax
: Agent x

cf
: Crazyflie

CoM
: Center of Mass

LPS
: Loco Positioning System

MAS
: Multi-Agent System

SHL
: Straight Horizontal Line

SVL
: Straight Vertical Line

UAV
: Unmanned Aerial Vehicle

SVL
: V-shaped Line

Chapter 1

Introduction

Nowadays, Unmanned Aerial Vehicles (UAVs) applications are crucial in several domains, either civilian and military. Considering their operational profile, these systems are mainly used within hostile environments while ensuring user integrity. The application range of UAVs spans largely within the civilian domain. Industrial and academic sectors are currently involving different types of UAVs (aircrafts, rotorcrafts and hybrids) to cope with specific needs and services.
For instance, unmanned aerial robots are significantly used in-the-field research missions: volcanic-activity monitoring, climate-change studies, crowds management/surveillance and epidemiological tracking.

In the last years, MASs have received an extensive attention from the scientific community, this is justified by the numerous applications that have been enhanced by increasing the number of agents. Popular applications (see Figure 1.1) include agriculture (e.g. crops imagery), weather monitoring (e.g. high-resolution mapping), collective transport (platooning), smart cities (e.g. traffic management), surveillance (e.g. homeland security), search and rescue (e.g. earthquake assessment), entertainment industry (e.g. movie shootings), just to mention a few.
A MAS is composed of interconnected sub-systems whose control algorithm is commonly based on distributed schemes and depend on the information collected from surrounding agents (neighbors). In fact, achieving collective coordinated behavior based on local information stands as one of the main challenges in multi-vehicle schemes.
Despite the promising MAS applications, it encompasses a wide spectrum of technological and scientific challenges to be addressed and solved. In this regard, numerous are the works in literature that have faced the related issues and have inspired the present thesis. The very first simulation of a flock is given by [1], that, starting from the bio-inspired behavior, models a distributed system of particles performing aggregate motion, as result of the communication between the particles, basing on three basic rules: flock centering, collision avoidance, velocity

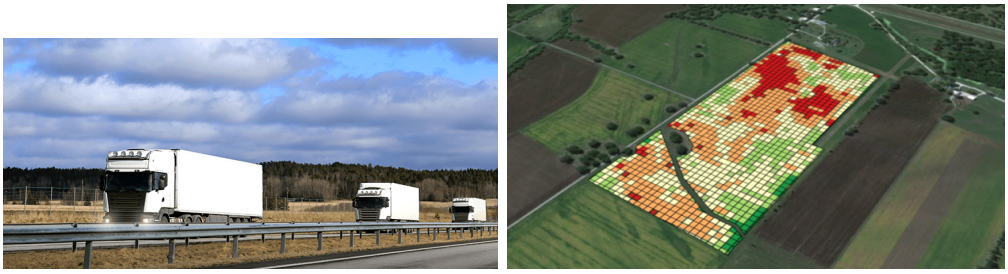


Figure 1.1: MAS current applications: (left) platooning, (right) aerial crops cartography

matching. A simulation of the three rules can be found in [2], where, additionally, the definition of a group objective, obstacle avoidance and split/rejoin maneuver are investigated. A similar behavior-based approach to flocking is presented and simulated on ground robots by [3], in which the formation, again, follows a waypoint trajectory, split and rejoin to avoid hazards. Another method is given by [4], that develops a discrete-time model of autonomous agents moving with same velocity but different headings: in this research it is demonstrated that the agents can be forced to move in the same direction if the heading of a single agent is updated referring to the average heading of his neighbourhood. This behavior is furthermore investigated, theoretically demonstrated and explained in [5]. In [6], the authors address general consensus algorithms, present theoretical results on the consensus convergence properties and conclude with a numerical example to validate the proposed consensus algorithms. The MAS is described through matrix theory, graph theory and control theory. In detail, in the simulations it is demonstrated the importance of the connection degree of the graph in the determination of the speed of consensus reaching. Similarly, in [7], the same relation between algebraic connectivity and consensus is demonstrated, by analysing three cases: 1) directed networks with fixed topology; 2) directed networks with switching topology; and 3) undirected networks with communication time-delays and fixed topology. In [8] it is considered a decentralized information exchange between the agents and each one is supplied with a common reference, describing the effect of the topology on the stability of the formation.
One of the most common consensus algorithm is represented by the randez-vous goal, as studied in [9], in which a model predictive control is actuated under some constraints, or in [10], in which it is proposed an algorithm able to converge all the agents to a common value, providing theoretical results on both time-invariant and time-variant topologies. The velocity matching algorithm, instead, is addressed by [11], where the topology is considered to be constant overtime, leading to an unrealistic model of flocking, in opposition with [12] in which the authors address a dynamic topology to simulate the collision avoidance rule. In [13], [14], [15] and [16],

the authors study a second-order consensus for MAS with fixed directed topology and communication constraints, concluding that the consensus can be reached if the general algebraic connectivity is larger than a threshold value and if the topology can be considered as fixed for long-enough intervals of time. Fascinating conclusions are retrieved by [17], in which, by analysing a time-dependent topology, it is observed that convergence's speed to the desired state isn't directly proportional to the communication links, as an increasing number of communication links could lead to a degradation of the performance.
The consensus algorithms are ulteriorly expanded by the definition of a desired trajectory in [18], in which the several agents follow virtual leaders, representing the reference waypoint track, in a distributed and coordinated fashion. At the end, the analysis of observability and controllability of different topologies, as well as the basic linear control of the model, is addressed by [19], [20],[21], where the desired trajectory is provided to a single agents, determining a leader/follower structure. In particular, [19] explores the requirements that satisfy the controllability and observability of the dynamic model's topologies, with interesting simulation of flock trajectory tracking control: a state feedback control, applied only to the leader, allows the tracking of the center of mass's motion and the states of the followers are observed directly from the input and the output of the leader.
Concerning what it is listed above, this thesis is focused on the modeling and the control of specific MAS topologies meant to reach spatial consensus, i.e. position and velocity consensus, and investigates the relation between communication flow and control.

1.1 Problem statement

The actual thesis addresses the consensus of Multi-Agent Systems considering diverse topologies. In practice, the success of the collective motion consensus relays on the capacity of agents to acquire inter-agent states, for both, position, in the kinematic case, and velocity in the dynamic one. Some scenarios require specific formations, however, these are not compatible with specific inter-agent sensory profile: e.g. the conical perception shape of ultrasound range finder or cameras. The latter will degrade the formation (consensus) to eventually have an unstable behavior. In the case where some states are not available, state observers are commonly used; provided the system verifies the observability property/condition. In the MAS case, observability is computed considering the MAS state-space system (process model) as well as the output system (measurement model). Depending on the formation configuration, and the underlying inherent topology, the observabilty condition might not be met. Thus, considering that in the MAS observability is topology-dependent suggests that there exists a set of topologies guaranteeing

observability while meeting certain navigation profile.

1.2 Contributions

In general, the actual thesis provides a detailed description of the relationship between collective navigation and topologies. Specifically, the contributions are:

- An observability extensive study conducted for three formations considering different perception profiles;
- In a similar way, a controllability study is presented to verify feasible topologies/formations for specific perception profiles;
- A simulation stage, performed to validate consensus algorithms;
- An experimental stage, described in detail envisioning forthcoming validation of the herein proposed configurations.

1.3 Outline

The remainder of thesis is the following: in Chapter 2 some theoretical preliminaries are given, with a detailed introduction to the graph theory, an overview of MAS classification and a deepening of consensus algorithms.
In Chapter 3, the information flow elapsing between the entities is characterized, through the definition of a topology described by graphs, meaning that a coordination protocol will shape the formation, the whole-set agreement value and the synchronization strategy for completing the goal. In this chapter the simulation of the position and velocity consensus algorithms is addressed, on a model of three-agents formation, characterized by homogeneous point-mass nodes organized in hierarchical anatomy. Subsequently to the verification of the two algorithms, it is investigated the addition of an observer to the simulation.
In Chapter 4 a brief introduction to the very first configuration of the quadrotor Crazyflie will be given with the experimentation of simple algorithms implementing basic flight manouvers.
Finally, in Chapter 5, the conclusion of the thesis are collected, with some future perspectives.

Chapter 2

Theoretical Preliminaries

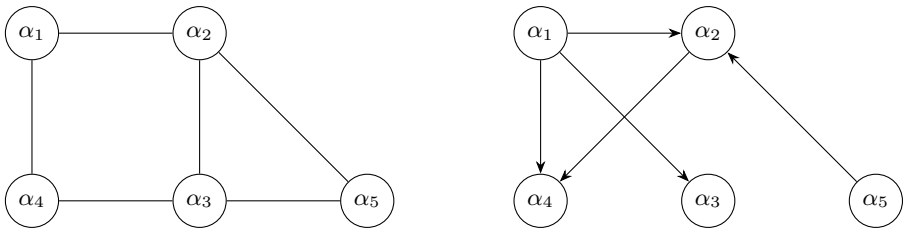
This chapter details essential concepts necessary to acquire an in-depth understanding of MAS. Going throughout these aforementioned concepts is crucial regarding forthcoming simulation and experimental stages.
This is outlined as follows:

- Basics of graph theory;
- Basics of Observability and Controllability;
- Basics of Multi-Agent Systems;
- Centralized, Decentralized and Distributed systems;
- Bioinspired Consensus;
- MAS model;
- Consensus theory and algorithms.

2.1 Basics on Graph Theory

A Multi-Agent System can be modeled as a group of dynamical systems (or agents), which has an information exchange topology represented by information graphs[22][23].
A graph is defined as $\mathcal{G} = \{\mathbb{V}, \mathbb{E}\}$ and consists of a set of vertices (or nodes) $\mathbb{V} = 1, ..., N$ and edges $\mathbb{E}$. Some distinguishing concepts used in the study and characterizing the graph theory are listed below.

Undirected graph: in an undirected graph the nodes i and j can get information from each other, i.e. $\forall i, j \in \mathbb{V}, (i, j) \in \mathbb{E} \Leftrightarrow (j, i) \in \mathbb{E}$. An example of



(a) Undirected graph (b) Directed graph (digraph)

Figure 2.1: Topology's examples

undirected graph can be seen in Fig.2.1(a).

Directed graph: a directed graph (or *digraph*) is formed by vertices connected by edges with a specified direction [24]. The edges are represented by arrows: consider two nodes $i, j \in \mathbb{V}$, if the arrow points from i to j then the first looks for the information from the latter, not viceversa. In the case in which the arrow is bidirect, the two vertices gain information from each other. A directed graph can be seen in Figure 2.1(b).

Connected graph: in a connected graph every vertex is connected to, at least, another vertex. In other words, if for every two nodes $i, j \in \mathbb{V}$ there is a path from x to y, e.g. the graph in Figure 2.1(a) is not connected.

Distance: the distance $d(i, j)$ between two nodes is the number of edges of the shortest path from i to j, e.g. in Figure 2.1(a) the distance between the nodes *1* and *6* is 3.

Diameter: the diameter $diam(\mathcal{G})$ of the graph $\mathcal{G}$ is the maximum distance $d(i, j)$ over all pairs of nodes:

$$diam(\mathcal{G}) = \max(dist_{\mathcal{G}})(i, j) \mid i, j \in v) \tag{2.1}$$

Adjacency matrix: the adjacency matrix could be represented in two different ways, based on the adjacency between nodes: two nodes i and j are called *adjacent* if there is an edge $\mathbb{E} : (i, j)$ between the two nodes, i.e. $\xi = (i, j) \in \mathbb{V} \times \mathbb{V} \mid i, j : adjacent$. The binary adjacency matrix $\mathcal{A} = [a_{ij}] \in \mathbb{V} \times \mathbb{V}$ is defined as:

$$a_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are adjacent;} \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

Instead, the weighted adjacency matrix $\mathcal{A}_w = [a_{w_{ij}}] \in \mathbb{V} \times \mathbb{V}$ is defined as:

$$a_{w_{ij}} = \begin{cases} \alpha_{ij} & \text{if } i \text{ and } j \text{ are adjacent;} \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

with α_{ij} as the weight of the edge $\mathbb{E} : (i, .j)$.
Note that the adjacency matrix is always a square matrix and it is symmetric ($\mathcal{A} = \mathcal{A}^T$) for undirected graph.

Incidence matrix: in an incidence matrix the rows correspond to the vertices of a graph, the columns to the edges. It is defined as $\mathcal{I} = [i_{ij}] \in \mathbb{V} \times \mathbb{E}$ [25], with:

$$i_{ij} = \begin{cases} 1 & \text{if the edge goes from } i \text{ to } j; \\ -1 & \text{if the edge goes from } j \text{ to } i; \\ 0 & \text{if } i \text{ and } j \text{ are not connected by an edge.} \end{cases} \tag{2.4}$$

Degree matrix: The degree matrix $\mathcal{D}$ of $\mathcal{G}$ is the diagonal matrix $\mathbb{V} \times \mathbb{V}$ with elements d_{ii} equal to the cardinality of node i's neighbor set $N_i = j \in \mathbb{V} : (i, j) \in \mathbb{E}$.

Laplacian matrix: the Laplacian matrix $\mathcal{L} = [l_{ij}]$ of $\mathcal{G}$ is defined as $\mathcal{L} = \mathcal{D} - \mathcal{A}$:

$$l_{ij} = \begin{cases} d_i & \text{if } i = j; \\ -1 & \text{if } (i, j) \in \mathbb{E}; \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

The row sums of $\mathcal{L}$ are zero and, thus, the vector of ones is an eigenvector corresponding to eigenvalue $\lambda_i(\mathcal{G}) = 0$, i.e., $\mathcal{L} = 0$.
For connected graphs, $\mathcal{L}$ has exactly one zero eigenvalue, and the eigenvalues can be listed in increasing order $0 = \lambda_1(\mathcal{G}) < \lambda_2(\mathcal{G}) \leq ... \leq \lambda_N(\mathcal{G})$, where the second eigenvalue $\lambda_2(\mathcal{G})$ is called the *algebraic connectivity*.
For undirected graphs, $\mathcal{L}$ is symmetric ($\mathcal{L} = \mathcal{L}^T$) and positive semi-definite, i.e., considering a non-zero column vector $\boldsymbol{z}$ of n real numbers, $\boldsymbol{z}^T \mathcal{L} \boldsymbol{z} > 0$.
Moreover, $\mathcal{L} = \mathcal{I}\mathcal{I}'$ for a digraph, where $\mathcal{I}$ is the incidence matrix.

Example 1 - Undirected graph
Consider the undirected graph in Figure 2.2(a), to better clarify the concepts explained before.
The binary and weighted adjancency matrices, the degree matrix and the Laplacian matrix are, respectively:

$$\mathcal{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}, \quad \mathcal{A}_w = \begin{bmatrix} 0 & e_1 & 0 & 0 & 0 \\ e_1 & 0 & 0 & e_2 & e_4 \\ 0 & 0 & 0 & e_3 & e_6 \\ 0 & e_2 & e_3 & 0 & e_5 \\ 0 & e_4 & e_6 & e_5 & 0 \end{bmatrix}, \quad \mathcal{D} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}, \quad \mathcal{L} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 3 & 0 & -1 & -1 \\ 0 & 0 & 2 & -1 & -1 \\ 0 & -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & -1 & 3 \end{bmatrix}.$$

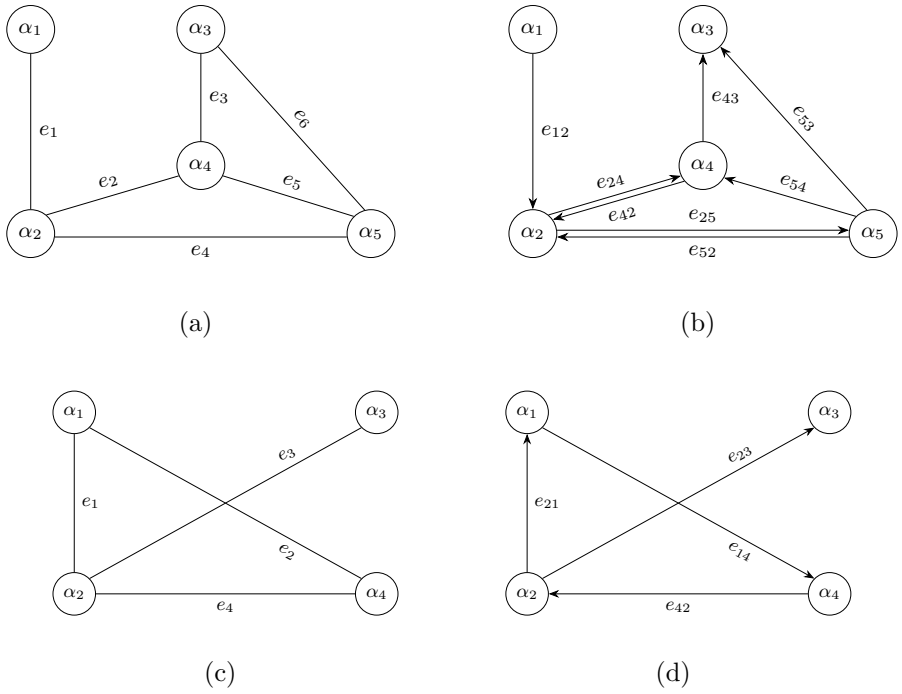


Figure 2.2: Examples of undirected graph (a), (c) and directed graph (digraph) (b), (d)

Example 2 - Directed graph
Another example is the directed graph in Figure 2.2(b).
Here, the binary and weighted adjancency matrices, the degree matrix and the Laplacian matrix are, respectively:

$$\mathcal{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}, \quad \mathcal{A}_w = \begin{bmatrix} 0 & a_{12} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{24} & a_{25} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & a_{42} & a_{43} & 0 & 0 \\ 0 & a_{52} & a_{53} & a_{54} & 0 \end{bmatrix}, \quad \mathcal{D} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}, \quad \mathcal{L} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 2 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 2 & 0 \\ 0 & -1 & -1 & -1 & 3 \end{bmatrix}.$$

Example 3 - Incidence matrix
By considering the undirected graph in Figure 2.2(c) and the directed graph in Figure 2.2(d),the incidence matrices related to both cases are calculated as follow:

$$\mathcal{I}_{undirected} = \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \qquad \mathcal{I}_{directed} = \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & 1 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} \end{array}$$

2.2 Basics of Observability and Controllability

In control theory, observability refers to the ability to monitor the internal states of a system given the values of the output[26]. The observability is strictly related to the controllability, that measures the capability to drive a system to a desired state, given an input with finite duration.
Given a state-space representation as the following:

$$\begin{cases} \dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} \\ \boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u} \end{cases} \tag{2.6}$$

the observability matrix $\mathcal{O}$ is defined as:

$$\mathcal{O} = \begin{pmatrix} C \\ C \cdot A \\ .. \\ C \cdot A^{n-1} \end{pmatrix} \tag{2.7}$$

where n is the number of states of the system.
The key concept is that, when the observability matrix is full rank, the initial conditions of the system can be found from the output values (the system is *observable*): $rank(\mathcal{O}) = n$.
The controllability matrix, for a system as the one in eq.(2.6) is:

$$\mathcal{C} = \begin{pmatrix} B & A \cdot B & ... & A^{n-1} \cdot B \end{pmatrix} \tag{2.8}$$

where n is, again, the number of states of the system.
If the controllability matrix is full rank, the system is fully *controllable*:
$rank(\mathcal{C}) = n$.
Controllability and observability are dual aspects of the same problem and, so, the study of the observability is an important issue to stabilize the controllability of the system. In fact, if the system is controllable, all the components of the output $\boldsymbol{y}$ of the state-space system can be derived.

2.3 Basics of Multi-Agent Systems

The MAS is defined as a network of individual entities, now on called agents, sharing information to achieve a collective objective [27]. An agent is defined as a flexible and intelligent autonomous entity capable of perceiving the environment and adapting accordingly [28][29]. It is straightforward noticing the effectiveness of single-agent versus multi-agent systems while fulfilling a determined assignment/mission. Collective behavior relies, indeed, on efficient inter-agent's and

surrounding environment information flow (i.e. topology) [30].
In general, the collective goal defines a coordination protocol that shapes, for instance, the formation and synchronized displacement (formation or coordinated motion)[31]. One can find examples in nature, migratory birds flying in different shaped coordinated formations (echelons), as ducks flying in V-shaped formations to optimize energy and, thus, increasing flight endurance/distance [32].
Collective behavior might be classified based on Architecture-Organization, Learning method, Communication protocol, Coordination.

Architecture and Organization

A first classification can refer to internal architecture of the system:

- an *homogeneous architecture* is composed of agents with the same internal architecture [33] (Local Goals, Sensor, Capabilities, Internal states, Inference Mechanism and Possible actions) but different physical location;
- an *heterogeneous architecture* is composed by agents that differ in ability, structure and functionality [34].

Another classification could be based on the agents' organization (Figure 2.3):

- *hierarchical organization* could help in the achievement of a common goal [35]: the control is distributed among the various agents (uniform hierarchy) or concentrated in a single agent, that is at the highest level of the hierarchy (simple hierarchy);
- in a *holonic organization* [36] an agent, that appears as a single entity, is composed of many sub-agents bound together by commitments;
- in *coalitions* a small group of agents is temporarily created until the achievement of a common goal;
- in *teams* the agents create a team and define a group goal which differs with their own goal.

Learning

- active learning;
- reactive learning;
- learning based on consequence.

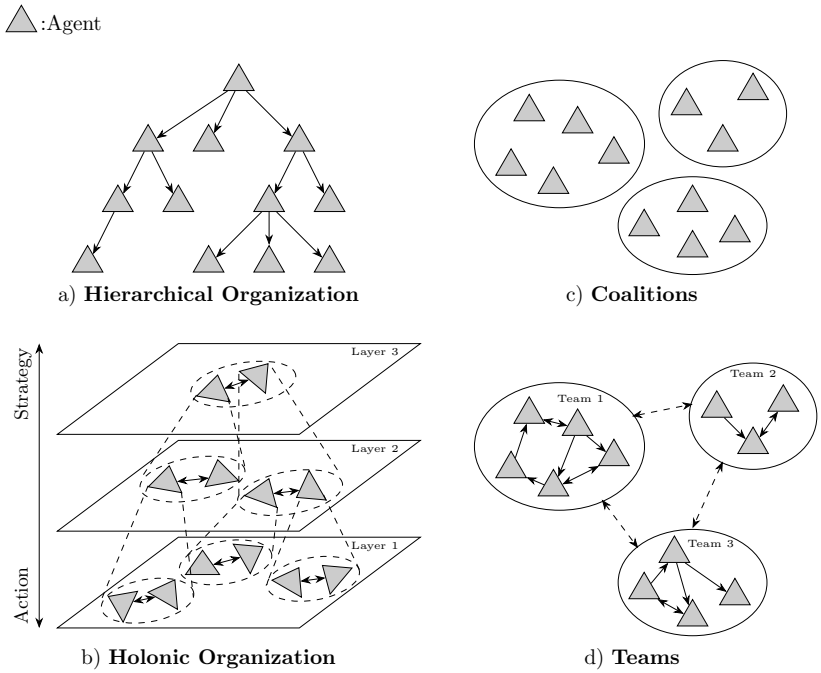


Figure 2.3: Examples of hierarchical organization (a), holonic organization (b), in coalitions (c), in teams (d)

Communication

- Local communication: information flow is bidirectional and not through intermediates;
- blackboards: a group of agents share a data repository.

Coordination

It can be achieved by applying constraints on the joint action choices of each agent or by utilizing he information collated from neighbouring agents.

- coordination through protocol;
- coordination via graphs;
- coordination through belief models.

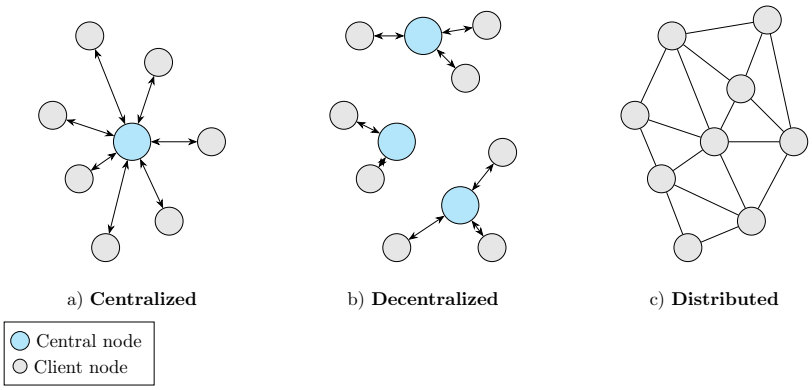


Figure 2.4: Examples of centralized (a), decentralized (b) and distributed (d) systems

2.4 Centralized, Decentralized and Distributed Systems

A distinction between Centralized, Decentralized and Distributed Systems is necessary to discriminate one from the other and identify the most suitable to depict the model of Multi-Agent Systems. All the three systems can work, all have advantages and disadvantages that make them more or less stable and reliable than others.

Centralized System

Centralized systems are systems in which one or more client nodes are directly connected to a central node. Main characteristics are:

- all the client node are synchronized to the central node clock (global clock);
- central node coordinates all the client nodes resulting in a single central unit;
- Risk of dependent failure since if the central node fails, all the system crashes;
- client/server architecture.

Decentralized System

In decentralized systems, there is not a central node but multiple, resulting with a final behavior of the system that is the aggregate of the decisions of the individual nodes. Main characteristics are:

- every node has its own clock, there isn't a global synchronization between nodes;
- multiple central nodes means multiple central units;
- also in decentralized systems the risk of dependent failure is present but, considering that a central node serves a part of the system, the whole system can continue with its mission;
- peer-to-peer structure (all nodes are peers of each other without any supremacy) or master-slave architecture (one node can become master, helping in the coordination of a part of the system, always without supremacy).

Distributed System

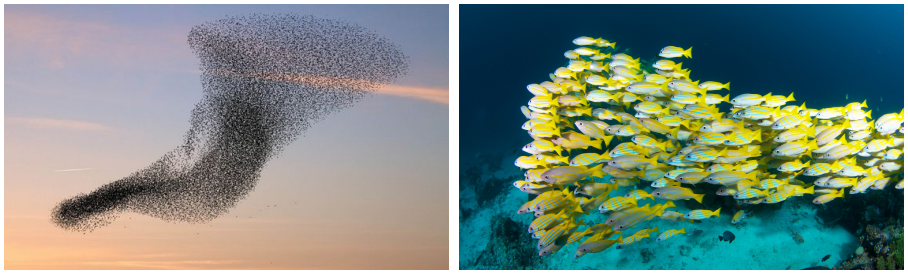
In distributed systems, similarly to the decentralized ones, every node makes its own decision. But here there isn't the centralization anymore, resulting in a cooperation, no single entity receives and responds to the request. The main characteristics are:

- no global clock;
- nodes apply consensus protocols to provide a concurrent action;
- every time a node fails, it doesn't affect the functioning of the others;
- peer-to-peer, client-server or n-tier architectures (in the last the operation is distributed between the nodes).

2.5 Bioinspired Consensus Problems

Prior works [37][38][39][40] describe the way animals "agree" about an interest variable (motion, communication, ...) to enhance mutual benefits. This includes: predators survival (wood pigeons, ostriches[41],...), collective foraging (wolves packs, dolphins[42], thresher sharks, lions, ..), long-distance migration (sardines, wildebeests, gazelles, swallows, ...), and, thus, the acquirement of more dynamic efficiency as the main advantage.
This collective animal behavior can be classified by considering the different species in which it's showed up:

- Flocking: the flocking behavior, also called murmuration, is exhibited by groups of birds in flight or during foraging (Figure 2.5(a));
- Swarming: it is a collective behavior seen in animals of similar size which aggregate together, migrating in some direction. In particular, it is applied to insects [43];



(a) (b)

(c) (d)

Figure 2.5: (a) birds flocking, (b) school of fishes, (c) herd of wildebeests, (d) ant colony

- Schooling: in nature fishes join for social purposes, forming schooling. Schooling is referred to a group of fishes that swim in the same direction in a coordinated manner [44] (Figure 2.5(b));
- Herding: it applies to groups of tetrapods, e.g. in herds, a social group of certain animals of the same species, or in packs, a group of canids (Figure 2.5(c));
- Ant colony: In a colony ants have their own role, cooperate and treat one another non-aggressively (Figure 2.5(d)).

Another interesting behavior is related to Algal Blooms, that is the accumulation in the population of algae. Although algae are not self-propelled as animals, blooms can be compared to the previous behaviors, as they are formed as consequence of a nutrient entering in the aquatic system.
The consensus problem seen in nature it's the main topic of biomimetics, that tries to emulates the models, systems, and elements of nature for the purpose of solving complex human problems [45] and can be a great hook for the flock simulation, as already done by Reynolds in 1986. Hence, it is essential a deep analysis of the relation and communication between the elements of a group, especially for what

concern the perception that each entity has of the whole flock and of each flock mate.

2.6 Multi-Agent Systems Model

MASs are complex and, thus, not easy to model.
The basic idea is to evaluate the real flocking behavior and to define a set of rules that could efficiently predict how a system changes over time, starting from an initial state. In the years, several models were developed, but the most reliable comes from the simulations of Craig Reynolds, that emulates boids (simple representation of birds) moving together according to basic rules[1].
In the model presented by Reynolds, a boid is said to perform a geometric flight since it moves along a path, not defined in advance, that can be described in 3D space. The object's own coordinate system is used to model small linear motions that together represent a continuous curved path, representing a discrete approximation of real flight, in which turning and moving happen continuously and simultaneously. In particular, the geometric flight is based on incremental translations along the object's local positive Z axis (forward direction) together with steering-rotations about the local X and Y axes (pitch and yaw).
Geometric flight adopts a simple model of viscous speed damping to represent a creature with a finite amount of available energy: if the boid continually accelerates in one direction, it will not exceed a certain maximum speed. Also, a maximum acceleration is used to truncate over-anxious requests for acceleration, hence providing for smooth changes of speed and heading.
Once the concept of geometric flight is explained, the building of a simulated flock can start. Three basic rules are adopted, that represent the opposing forces of collision avoidance and the urge to join the flock:

- Collision Avoidance (*Separation*): avoid collisions with nearby flockmates (Figure 2.6(a));
- Velocity Matching (*Alignment*): attempt to match velocity with nearby flockmates (Figure 2.6(b));
- Flock Centering (*Cohesion*): attempt to stay close to nearby flockmates (Figure 2.6(c)).

Static collision avoidance and dynamic velocity matching are complementary since the first represents the urge to steer away from an imminent impact and it is based on the relative position of the flockmates and ignores their velocity, the latter is based only on velocity and ignores position.
Thanks to velocity matching between neighbors, it is assumed separations between

(a) (b) (c)

Figure 2.6: Graphic representations of: (a) Collision avoidance (Separation), (b) Velocity Matching (Alignment), (c) Flock centering (Cohesion)

boids remains approximately invariant with respect to ongoing geometric flight, so collision is improbable. The main purpose of static collision avoidance is, then, to establish the minimum required separation distance, instead, velocity matching tends to maintain it.
Flock centering makes a boid want to be near the centre of the nearby flockmates (localized perception) and correctly allows simulated flocks to bifurcate (e.g. to avoid an obstacle). In fact, if the individual boid can stay close to its nearby neighbors, it does not care if the rest of the flock turns away.
This simplified model does not completely represent the birds' sense, because the entities are modelled with a short-range perception: in this case if two flocks are enough far apart, they would not join together.
Analysing real behaviour of birds, there is no limit to the complexity of the flocks, leading to assume that the birds can fly with any number of flockmates. In fact, it was supposed that the bird might be aware of only itself, its two or three nearest neighbours and the rest of the flock [44].
In simulation the unlimited number of members cannot be taken in account and a neighbourhood must be outlined. The neighborhood is defined as a spherical zone of sensitivity centered at the boid's local origin, in which the sensitivity decreases exponentially with the distance.
Moreover, the sensitivity should be increased in the forward direction and be proportional to the boid's speed, since being in motion needs a bigger consciousness in what is ahead.
Now, what can be considered part of a neighbourhood? Two different formulations are established: one establishes that all boids within a certain radius are part of a neighbourhood, based on the second, the neighbourhood is composed by the N closest boids.
A challenging issue is the splitting of a flock to avoid an obstacle, it increases the complexity of the model. Once the geometric shape and dimension of the obstacle

are well represented, it is possible to proceed with two different shapes of collision avoidance:

- Field of forces method: the boids are subjected to a growing repulsive force as they get nearer to the obstacle.
  Advantages: easiness of the model, in fact, the geometry of the field is easy and, so, the acceleration of the boids approaching the obstacle.
  Disadvantages: in the case in which the boid gets close to the obstacle with an opposite angle to the direction of his force field, the boid will be only slowed down and, in the worst case, it will not avoid the obstacle; the force field do not allow a boid to ignore an obstacle to which it passes alongside; boids must be properly modelled to avoid non homogeneous distribution of the field (e.g. too strong close to the obstacle but too weak far away from it).

- Steer-to-avoid method: the boid avoids only the obstacles it faces along the flight. This method is more robust and it models in a better way the real flocking behavior. The obstacle might not be on the way, but an object of fear (predators) in the surroundings.

2.7 Consensus algorithms

Consensus is a fundamental problem in the control of MASs and it concerns with the task of a group of interacting agents reaching a common goal based on the communication and exchange of information between the entities.
In fact, the coordination and synchronization of agents could improve the possibility to achieve a optimal global solution, in contrast to the action of independent agents. This topic is, also, strictly related to the system and graph theories, through which a consensus problem could be solved by dividing it into easier small problems.
Synchronization can be obtained by implementing a fault tolerant system, in which a limited number of faulty processes could be discarded in order to not affect the final result, and by applying constraints on the actions of agents. Some principal requirements of a consensus protocol are:

- *Agreement*: all correct processes (faults free) must agree on the same value;

- *Weak validity*: the output of a correct process must be the input of at least one correct process;

- *Strong validity*: if all correct processes receive the same input value, they must all output that value;

- *Termination*: all processes must eventually decide on an output value.

A special case of consensus problem concerns the development of protocols used to decide to which agent is assigned what task , so a set of rules to determine the individual roles of each vehicle.
Other relevant consensus problems could be the rendez-vous problem (in which a group of entities meets at a common location at a common time), synchronization, flocking and formation controls.
A consensus problem in MASs can be represented by a directed or undirected graph (see sec. 2.1), where each agent is represented by a vertex and it's characterized by a decision parameter [6]. Suppose that each agent has a dynamic as the one that follows:

$$\dot{x_i} = f(x_i, u_i), \quad i \in \mathbb{V} \tag{2.9}$$

A dynamic graph is a dynamical system described by its topology and the vector of states $\boldsymbol{x}$ [7], but in the case of fixed topology (during all the time or during the majority of the time), as the one considered here, the dynamical system is described by the vector $\boldsymbol{x}$ only, and, then, each agent is described by the following differential equation:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}).$$

Let χ be a function of $\boldsymbol{x} = (x_1, x_2, ..., x_V)^T$ and $x(0)$ be the initial state of the system. The following decision-value is generated:

$$\boldsymbol{y} = \chi(\boldsymbol{x})$$

The χ-consensus problem is asymptotically solved by a protocol if and only if there exists an asymptotically stable equilibrium $\boldsymbol{x}^*$ satisfying $x_i^* = \chi(x(0))$ for all $i \in \mathbb{V}$. Some special cases for χ-consensus are:

- *Average-consensus*: $\chi(\boldsymbol{x}) = Ave(x) = 1/N(\sum_{i=1}^{N} x_i), \quad i \in \mathbb{V}$;
- *Max-consensus*: $\chi(\boldsymbol{x}) = \max x_i, \quad i \in \mathbb{V}$;
- *Min-consensus*: $\chi(\boldsymbol{x}) = \min x_i, \quad i \in \mathbb{V}$.

In this thesis, the first consensus protocol taken into consideration, which will be called *Kinematic consensus protocol*, is the following [5][46][8][47][48]:

$$\dot{x}_i = -\alpha \sum_{j=1}^{N} a_{ij} (x_i - x_j) \Leftrightarrow \dot{x}_i = -\alpha \sum_{j=1}^{N} l_{ij} x_j, \quad i, j \in \mathbb{V} \tag{2.10}$$

with N the number of agents and α a weighting factor.
It can be easily demonstrated by considering the following example: consider a set of three agents defined by an initial position and the following Adiacency and Laplacian matrices:

$$\boldsymbol{x_0} = [x_{0_{A1}} \quad x_{0_{A2}} \quad x_{0_{A3}}]^T$$

$$\mathcal{A} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad \mathcal{L} = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

The first half of the equation, $\dot{x_i} = -\alpha \sum_{j=1}^{3} a_{ij}(x_i - x_j),\ \ i, j \in \mathbb{V},\ \ \alpha = 1$, lets to:

$$\begin{cases} \dot{x}_1 = -a_{11}(x_{0_{A1}} - x_{0_{A1}}) - a_{12}(x_{0_{A1}} - x_{0_{A2}}) - a_{13}(x_{0_{A1}} - x_{0_{A3}}) = -2x_{0_{A1}} + x_{0_{A2}} + x_{0_{A3}} \\ \dot{x}_2 = -a_{21}(x_{0_{A2}} - x_{0_{A1}}) - a_{22}(x_{0_{A2}} - x_{0_{A2}}) - a_{23}(x_{0_{A2}} - x_{0_{A3}}) = -2x_{0_{A2}} + x_{0_{A1}} + x_{0_{A3}} \\ \dot{x}_3 = -a_{31}(x_{0_{A3}} - x_{0_{A1}}) - a_{32}(x_{0_{A3}} - x_{0_{A2}}) - a_{33}(x_{0_{A3}} - x_{0_{A3}}) = -2x_{0_{A3}} + x_{0_{A1}} + x_{0_{A2}} \end{cases}$$

and the second half, $\dot{x_i} = -\alpha \sum_{j=1}^{N} l_{ij} x_j,\ \ i, j \in \mathbb{V},\ \ \alpha = 1$, leds to:

$$\begin{cases} \dot{x}_1 = -l_{11}x_{0_{A1}} - l_{12}x_{0_{A2}} - l_{13}x_{0_{A3}} = -2x_{0_{A1}} + x_{0_{A2}} + x_{0_{A3}} \\ \dot{x}_2 = -l_{21}x_{0_{A1}} - l_{22}x_{0_{A2}} - l_{23}x_{0_{A3}} = -2x_{0_{A2}} + x_{0_{A1}} + x_{0_{A3}} \\ \dot{x}_3 = -l_{31}x_{0_{A1}} - l_{32}x_{0_{A2}} - l_{33}x_{0_{A3}} = -2x_{0_{A3}} + x_{0_{A1}} + x_{0_{A2}} \end{cases}$$

verifying the equality.
The eq.(2.10) causes the information state x_i of vehicle i to be driven toward the information states of its neighbors[10], leading to an agreement, without, however, specifying the agreement value. So, the agents reach the consensus if

$$\lim_{t\to\infty} x_1(t) = ... = \lim_{t\to\infty} x_n(t)$$

The stability of the system $\dot{\boldsymbol{x}} = -\alpha \cdot \boldsymbol{\mathcal{L}x}$ depends on the location of the eigenvalues of the Laplacian matrix $\mathcal{L}$ [49].
The agreement value corresponds to the Center of Mass (CoM) of the initial positions of the three agents, as it will be demonstrated in sec.3.3. The kinetic consensus algorithm can be enriched by the addition of a *navigational term* that will specify the desired agreement value:

$$\begin{aligned} \dot{x}_i = -\alpha \sum_{j=1}^{n} a_{ij}\left(x_i - x_j\right)) - K_P \cdot \Gamma\left(x_i - x_d\right) \ \ \Leftrightarrow \\ \dot{x}_i = -\alpha \sum_{j=1}^{N} l_{ij} x_j - K_P \cdot \Gamma\left(x_i - x_d\right),\ \ i, j \in \mathbb{V}. \end{aligned} \tag{2.11}$$

with α and K_P as weighting term and Γ a matrix defining which agent is aware of the desired agreement value, namely determining the leader.
Since the protocol in eq.(2.10) is based on the kinematics of the system, the agents will move with an initial velocity that will be constant in the time (acceleration is null), thereby a second protocol must be considered to cover the dynamics of the agents.
The above-mentioned consensus algorithm can be extended by considering the

acceleration contribute, obtaining a *dynamic consensus algorithm*[13][50][51][52][53]:

$$\begin{cases} \dot{x}_i = v_i, \;\; i \in \mathbb{V} \\ \dot{v}_i = -\alpha \sum_{j=1}^{n} a_{ij} \left(x_i - x_j\right) - \beta \sum_{j=1}^{n} a_{ij} \left(v_i - v_j\right), \;\; i,j \in \mathbb{V} \end{cases} \Leftrightarrow \begin{cases} \dot{x}_i = v_i, \;\; i \in \mathbb{V} \\ \dot{v}_i = -\alpha \sum_{j=1}^{n} l_{ij} x_j - \beta \sum_{j=1}^{n} l_{ij} v_j, \;\; i,j \in \mathbb{V} \end{cases} \tag{2.12}$$

where $x_i \in \mathbb{R}^N$ is the position state vector of the i-th agent, $v_i \in \mathbb{R}^N$ is the velocity state vector of the i-th, $u_i \in \mathbb{R}^N$ is the control protocol of the i-th agent. Moreover $\alpha, \beta > 0$ represent weighting factors and a_{ij} represents the element of the adjancency matrix describing the formation, as well as l_{ij}, representing the elements of the Laplacian matrix.
The second-order consensus is reached when:

$$v_{f_i} = \sum_{j=1}^{N} \Lambda_j \cdot v_j(0), \;\; i,j \in \mathbb{V}$$

with v_{f_i} the final velocity of i-th agent, Λ_j the non-negative left eigenvector of the Laplacian and $v_j(0)$ the initial velocity of j-th agent.
When considering the second-order consensus algorithm, it is assumed that each agent can continuously and in a constant way sense its neighbors, thereby the topology is constant, because the time-varying variables lead to a non-linear dynamics. In the reality this is not reasonable, since the agents in motion cannot maintain a constant formation, but it can be assumed that the acceleration of the agents is zero, $\dot{v}_i = 0$, in the small time intervals in which the topology cannot be maintained constant.
As before, the dynamic consensus algorithm can be implemented by the specification of the desired trajectory, resulting in the following system of equations:

$$\begin{cases} \dot{x}_i = v_i, \;\; i \in \mathbb{V} \\ \dot{v}_i = -\alpha \sum_{j=1}^{n} l_{ij} x_j - K_P \cdot \Gamma \left(x_i - x_d\right) - \beta \sum_{j=1}^{n} l_{ij} v_j - K_D \cdot \Gamma \left(\dot{x}_i - \dot{x}_d\right), \;\; i,j \in \mathbb{V} \end{cases}$$

with K_P, K_D as weighting factors that will implemented in a PD-control of the system.
It will seen in the next chapter that other crucial requirements are directed topologies and communication constraints, as it will be verified that consensus can be guaranteed if the general algebraic connectivity of the strongly connected topologies and measure of the communication among the agents are larger than some threshold values, respectively.

Chapter 3

Application of Consensus Algorithms

3.1 Application of Consensus Algorithms

In this chapter it is aimed the identification of the topologies with the best observability and controllability trends. In addition, a model will be built, of a three-agents formation and the consensus algorithms will be simulated on it, constituting the background of the next exprimental stages.
The chapter is outlined as follow:

- Observability Analysis;
- Simulation of Kinematic Consensus Algorithm;
- Simulation of Dynamic Consensus Algorithm;
- Addition of the Observer

3.2 Observability analysis

The analysis of observability is intended to detect the optimal formation, in terms of observability and, then, controllability (see sec.2.2). It is recalled that an observable system allows to get the initial conditions from the outputs. In order to analyze multiple cases, in the tests there is a classification in terms of formation, sensor sensitivity shape and their operating situation (all the sensors are properly working or some failures arise).

Recalling that the observability matrix is calculated as:

$$\mathcal{O} = \begin{pmatrix} C \\ C \cdot A \\ .. \\ C \cdot A^{n-1} \end{pmatrix},$$

it is necessary to define a state-space representation of the system that includes the consensus algorithms: consider the kinematic consensus algorithm, it is reported below as:

$$\dot{x}_i = -\sum_{j=1}^{n} l_{ij} x_j, \;\; i, j \in \mathbb{V} \quad \Leftrightarrow \quad \dot{\boldsymbol{x}} = -\mathcal{L}\boldsymbol{x}$$

recalling that l_{ij} corresponds to the element of the Laplacian matrix describing the topology. It will considered only the equation in the Laplacian version since it results to be more versatile in the Simulink model.
By assuming a null input vector ($\boldsymbol{u} = 0$) and substituting the kinematic consensus protocol in the state-space system

$$\begin{cases} \dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} \\ \boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u} \end{cases}$$

the following system results:

$$\begin{cases} \dot{\boldsymbol{x}} = A\boldsymbol{x} = -\mathcal{L}\boldsymbol{x} \\ \boldsymbol{y} = C\boldsymbol{x} \end{cases} \tag{3.1}$$

coming to the important conclusion that

$$A = -\mathcal{L} \tag{3.2}$$

So the Observability matrix is modified as it follows:

$$\mathcal{O} = \begin{pmatrix} C \\ C \cdot (-\mathcal{L}) \\ .. \\ C \cdot (-\mathcal{L})^{n-1} \end{pmatrix}, \tag{3.3}$$

Hence, the observability of the system depends on the Laplacian matrix.
Three agents' arrangements in the space are identified, corresponding to three formation shapes:

- Straight Vertical Line (SVL);
- Straight Horizontal Line (SHL);

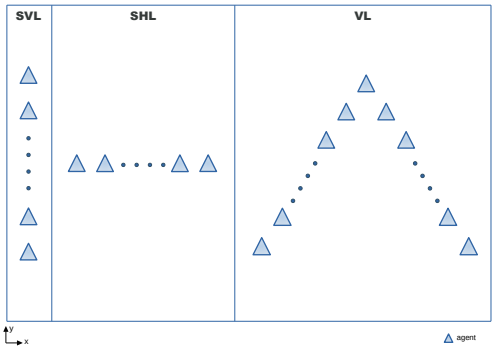
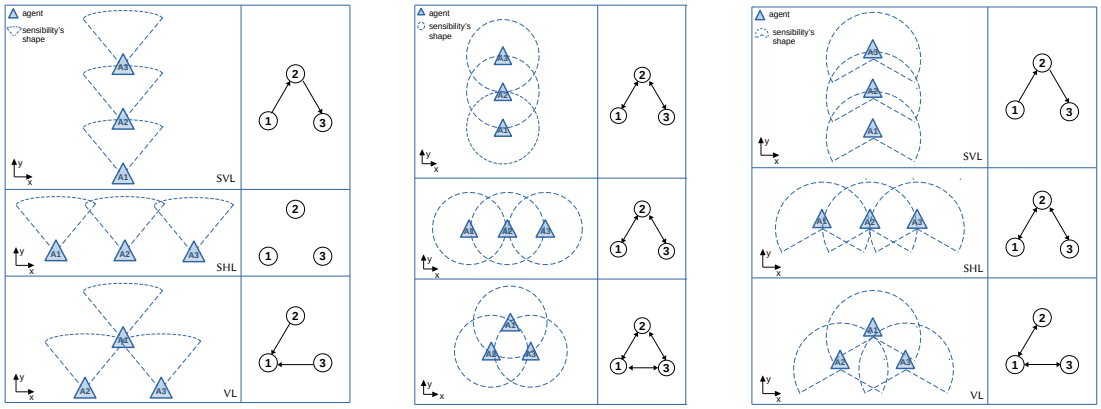


Figure 3.1: Straight vertical line, straight horizontal line and V-shaped line formations of several agents



(a) (b) (c)

Figure 3.2: Straight vertical line, straight horizontal line and V-shaped line formations with conical, circular and arched sensitivity shapes considering only three agents

- V-shaped Line (VL).

The succeeding figures, indeed, will refer to just three agents such as in the simulations. In Figure 3.2 the three-agents formations are displayed with three different sensitivity shapes of sensors (conical, circular and arched shapes) that lead to a different exchange of information between the agents, as described by the corresponding graphs.
The sensitivity depends on the distance between the agents, the radius of the sensitivity shape and, for the conical and arched shapes, on the angular diameter. Assumptions are made to obtain a simple protocol to test the different formations:

Sensor type	**Sensitivity shape**	**Field of View (FoV)**
Lidar Lite	conical	$\approx 71°$
M8 Lidar	spherical	360° horizontal 20° (+3°/-17°) vertical
URG-04LX-UG01 Laser Range Finder	arched	240° horizontal

Table 3.1: Real sensor's examples

- The distance between the agents is supposed to be almost constant and within the circular (Figure 3.2(b)), the conical (Figure 3.2(a)) or the arched (Figure 3.2(c)) sensitivity areas;
- The agents are supposed to keep the formation;
- The parameters that describe the sensitivity area of the agent's sensor are constant and, in case of an error, the sensitivity area is considered to be null.

The information coming from the sensors is represented by the output vector $\boldsymbol{y} = [y_1; y_2; y_3]$ and an error in the i-sensor leads the corresponding y_i to be null. Then, different cases are analyzed:

- a: the three agents' sensors correctly work, outputs y_1, y_2 and y_3 are known;
- b: a misbehavior affects the third agent, the two outputs y_1 and y_2 are known but not y_3;
- c: a misbehavior affects the first agent, the two outputs y_2 and y_3 are known but not y_1;
- d: a misbehavior affects the second agent, the two outputs y_1 and y_3 are known but not y_2;
- e: only the output y_1 is known since misbehaviors affect both the second and the third agents;
- f: only the output y_2 is known since misbehaviors affect both the first and the third agents;
- g: only the output y_3 is known since misbehaviors affect both the first and the second agents.

Sensor	**Formation**	***a*:** y_1,y_2,y_3	***b*:** y_1,y_2	***c*:** y_2,y_3	***d*:** y_1,y_3	***e*:** y_1	***f*:** y_2	***g*:** y_3
conical	SVL	o	o		o	o		
	SHL	o						
	VL	o		o				
circular	SVL	o	o	o	o	o		o
	SHL	o	o	o	o	o		o
	VL	o	o	o	o			
arched	SVL	o	o		o	o		
	SHL	o	o	o	o	o		o
	VL	o	o	o	o		o	o

Table 3.2: Table with the observability results for conical and circular sensors in SVL, SHL, VL formations, considering different cases

In Figure 3.3, 3.4 and 3.5, the several cases are represented in the three formations with the conical and the circular sensitivity shapes, but the same cases are considered also in the case of the arc-shaped sensor.
In the first analysis, it was considered the case in which both $\mathcal{L} = -A$ and C matrices change from case to case, in fact matrix C accounts for the known output components in the different cases and $\mathcal{L}$ depends on the communication.
Having three agents, the states are $n = 3$ and, then, observability matrix is always

$$\boldsymbol{O} = \begin{pmatrix} C \\ C \cdot A \\ C \cdot A^2 \end{pmatrix} = \begin{pmatrix} C \\ C \cdot (-\mathcal{L}) \\ C \cdot \left(-\mathcal{L}^2\right) \end{pmatrix}$$

Observability test's results are collected in Table 3.2. It is essential to understand how the different formations or the use of different sensors influence the observability of the system. As it can be seen from the results, a comparison between the sensors can be done, especially by considering the same graphs but different exchanging of information. Two examples can be considered: a first comparison can be done between the SVL formation with circular and conical sensitivity shapes, resulting in the improvement of performances in the case of undirected graph (SVL with circular shape), as this topology is observable in more cases with comparison with the directed graph (SVL with conical shape).
Same can be concluded by considering the VL formation with conical and arched shaped sensors: again, the observability is verified in more cases when the topology is undirected (arched shaped sensor).
Furthermore, the circular shaped sensor gives the best observability trend and, in particular, the two SHL and SVL formations result to be observable in more cases.

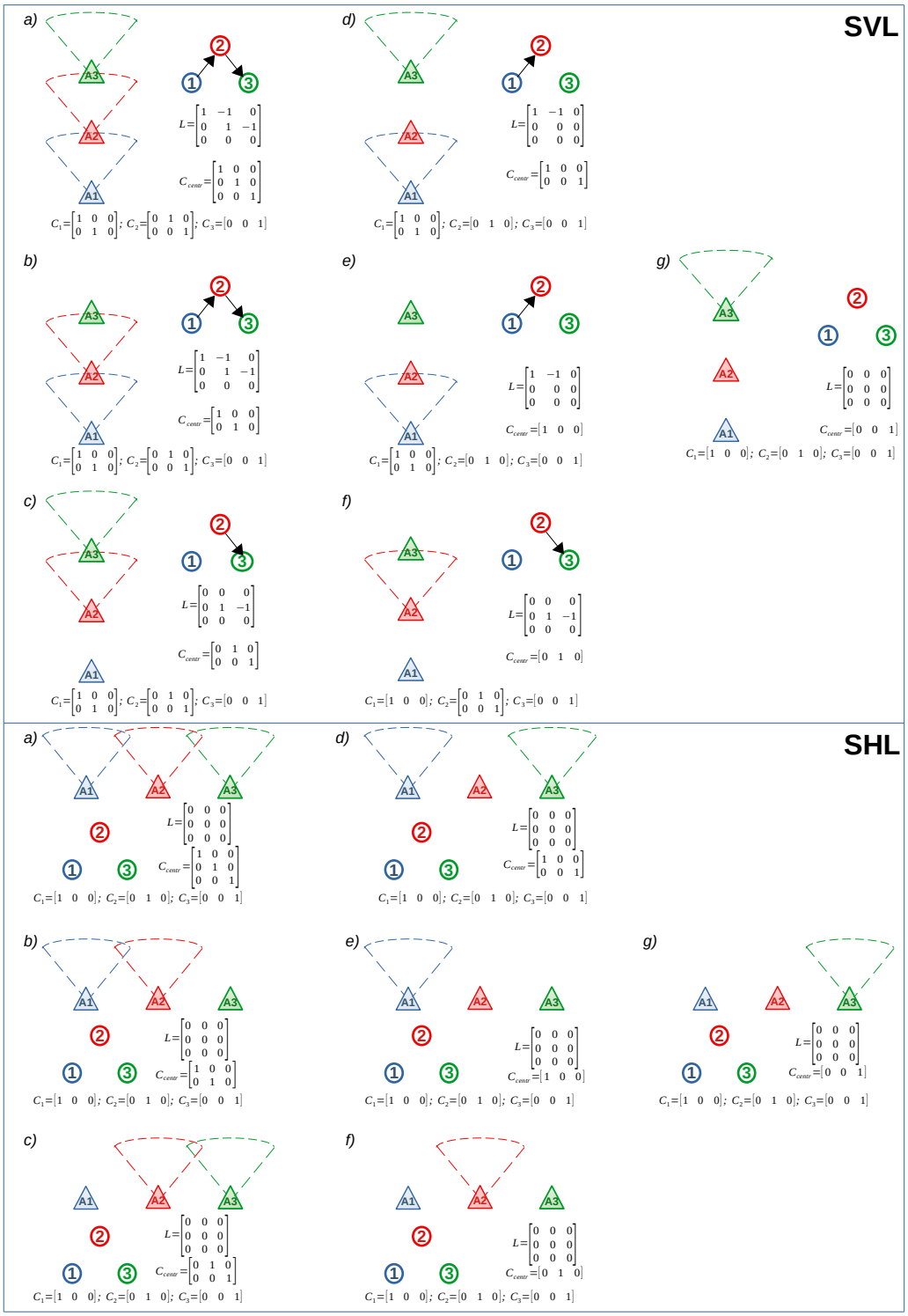


Figure 3.3: Different cases for SVL and SHL formations with conical shaped sensor

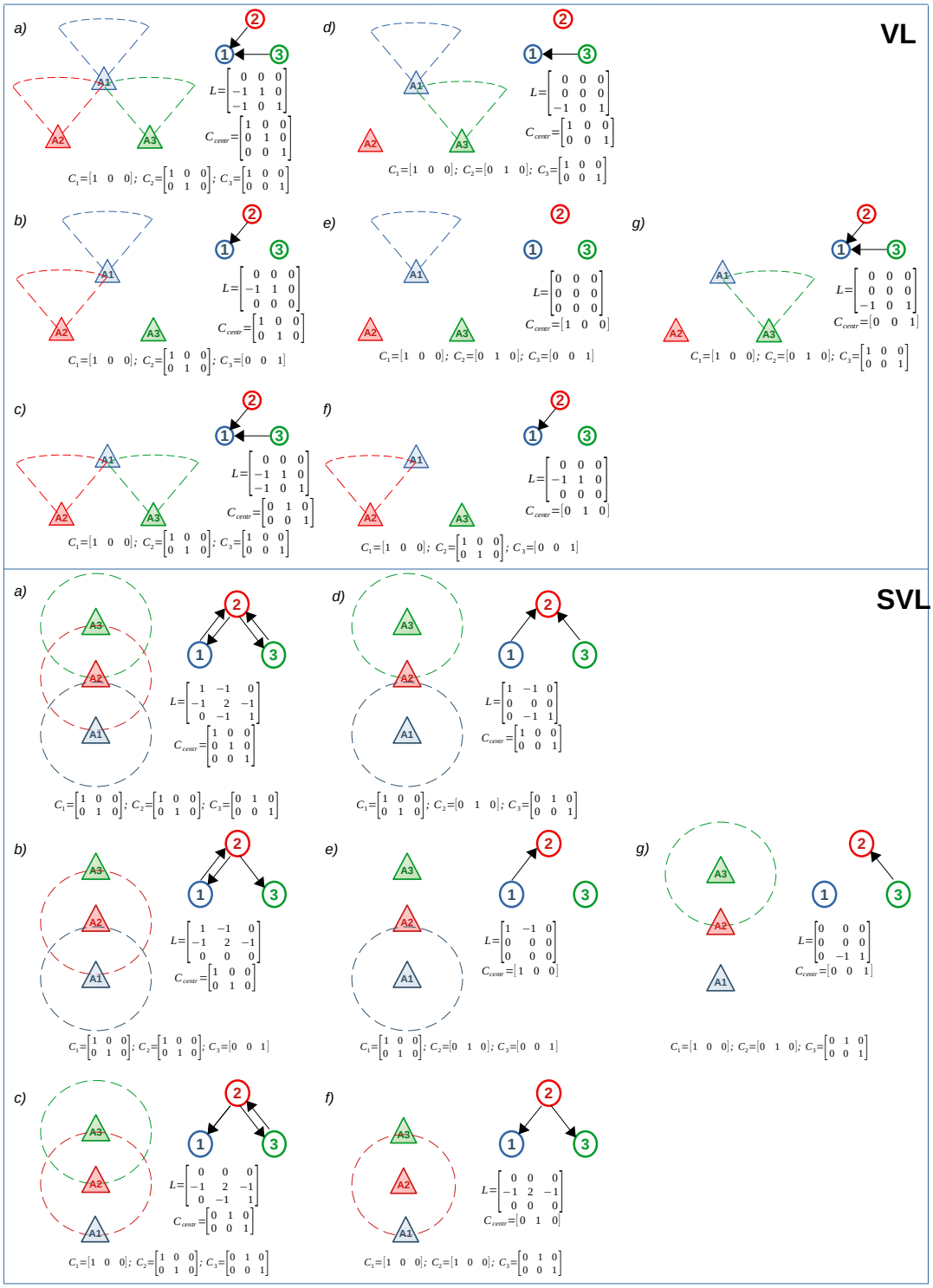


Figure 3.4: Different cases for VL with conical shaped sensor and SVL with circular shaped sensor

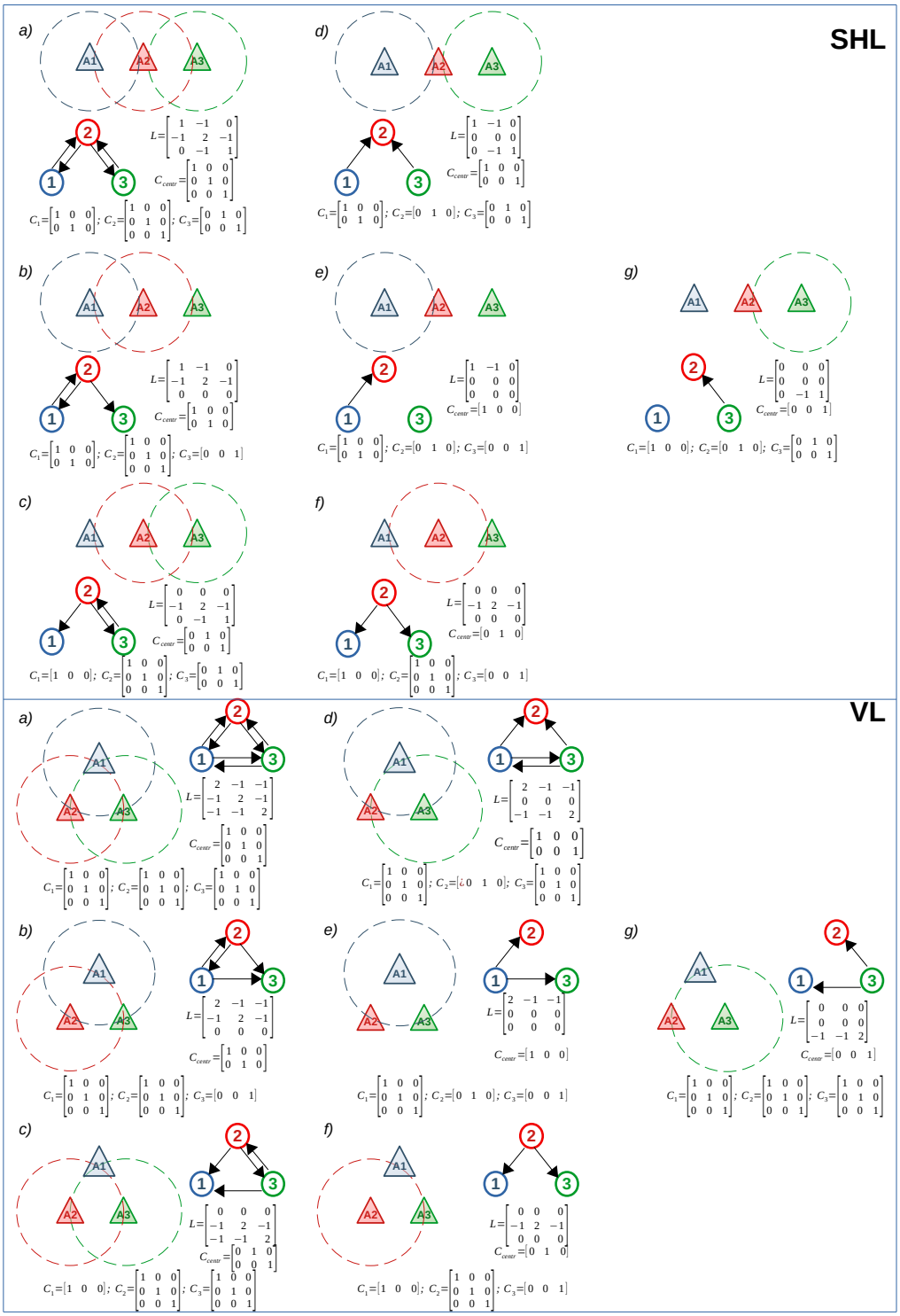


Figure 3.5: Different cases for SHL and VL formations with circular shaped sensor

With this result, the question has been raised whether additional states, for example the inclusion of obstacles along the trajectory, could implement the observability among the cases. The obstacles are considered as non-animated elements (their output is unknown) and they can be only sensed by the agents.
In Figure 3.6, 3.7, 3.8, the three different formation, with conical and circular sensor's shapes only, are represented with additional obstacles. In particular, the following topologies are taken in account:

- **T1**: one obstacle seen only by A1;
- **T2**: one obstacle seen only by A2;
- **T3**: one obstacle seen only by A3;
- **T4**: one obstacle seen by A1 and A2;
- **T5**: one obstacle seen by A2 and A3;
- **T6**:one obstacle seen by A1 and A3;
- **T7**: two obstacles, the first is seen by A1 and A2, the second is seen by A2 and A3;
- **T8**: two obstacles, the first is seen by A1 and A2, the second is seen by A1 and A3;
- **T9**: two obstacles, the first is seen by A1 and A3, the second is seen by A2 and A3;
- **T10**: three obstacles, each one seen by a couple of agents.

Not all the topologies are physically possible for the different formations, in fact only the V-shape formation allows all the ten topologies.
Despite not all the different cases are displayed in the Figure 3.6, 3.7, 3.8, they are considered in the tests.
To have a clear idea, a complete example of T1 topology in the case of SHL formation with a circular sensor's shape follows.

Example: SHL - circular - T1
The states are four: the three agents and one obstacle. The observability matrix is, in the all the cases, the following:

$$\mathcal{O} = \begin{pmatrix} C \\ C \cdot A \\ C \cdot A^2 \\ C \cdot A^3 \end{pmatrix}$$

For each cases the two matrices $\mathcal{L} = -A$ and C are calculated. Case a: all the sensors are correctly working and there is an obstacle that can be seen by the first agent.

$$\mathcal{L} = \begin{bmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}; \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Case b: an error affects the third agent:

$$\mathcal{L} = \begin{bmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}; \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Case c: an error affects the first agent:

$$\mathcal{L} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}; \qquad C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Case d: an error affects the second agent:

$$\mathcal{L} = \begin{bmatrix} 2 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}; \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Case e: an error affects the second and the third agents:

$$\mathcal{L} = \begin{bmatrix} 2 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}; \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}.$$

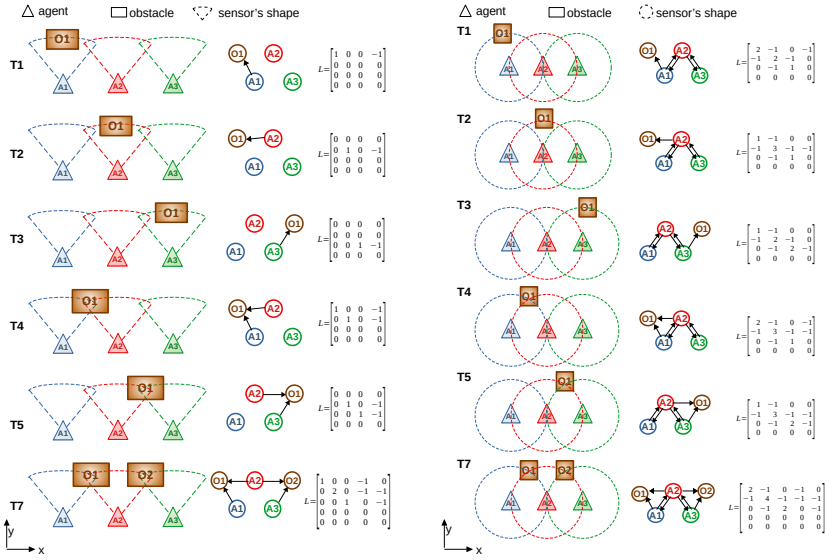
Case f: an error affects the first and the third agents:

$$\mathcal{L} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}; \qquad C = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}.$$

Case g: an error affects the first and the second agents:

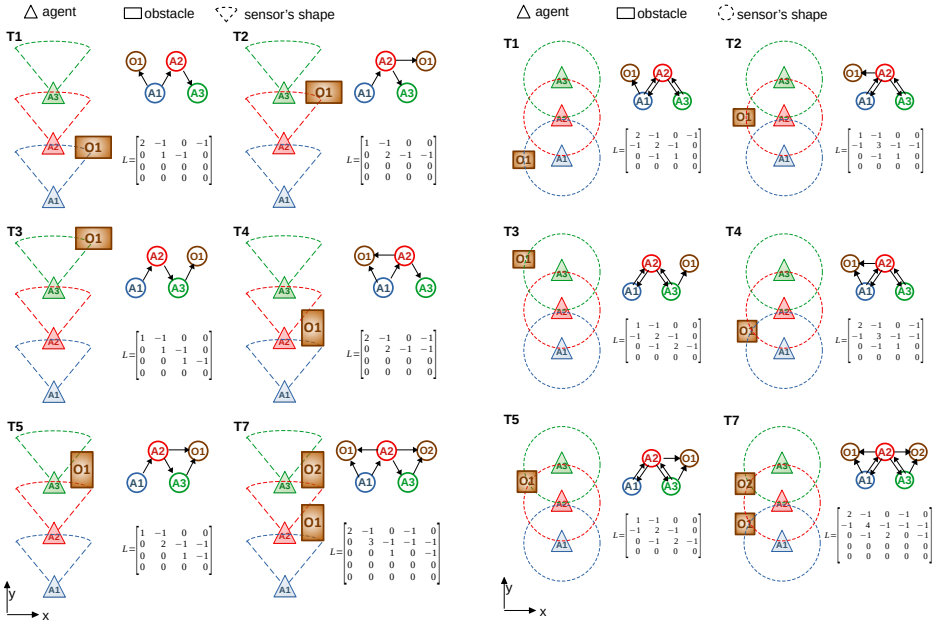
$$\mathcal{L} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}; \qquad C = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}.$$

This kind of procedure is, then, repeated for each topology and the results are collected in the Table 3.3.
In the case in which the observability matrix is full rank and, then, the system is observable, all the states can be observed, thus, resolving the problem of the missing outputs



(a) (b)

Figure 3.6: Configurations with additional obstacles in SHL formations with conical (a) and circular (b) shaped sensors



(a) (b)

Figure 3.7: Configurations with additional obstacles in SVL formations with conical (a) and circular (b) shapes

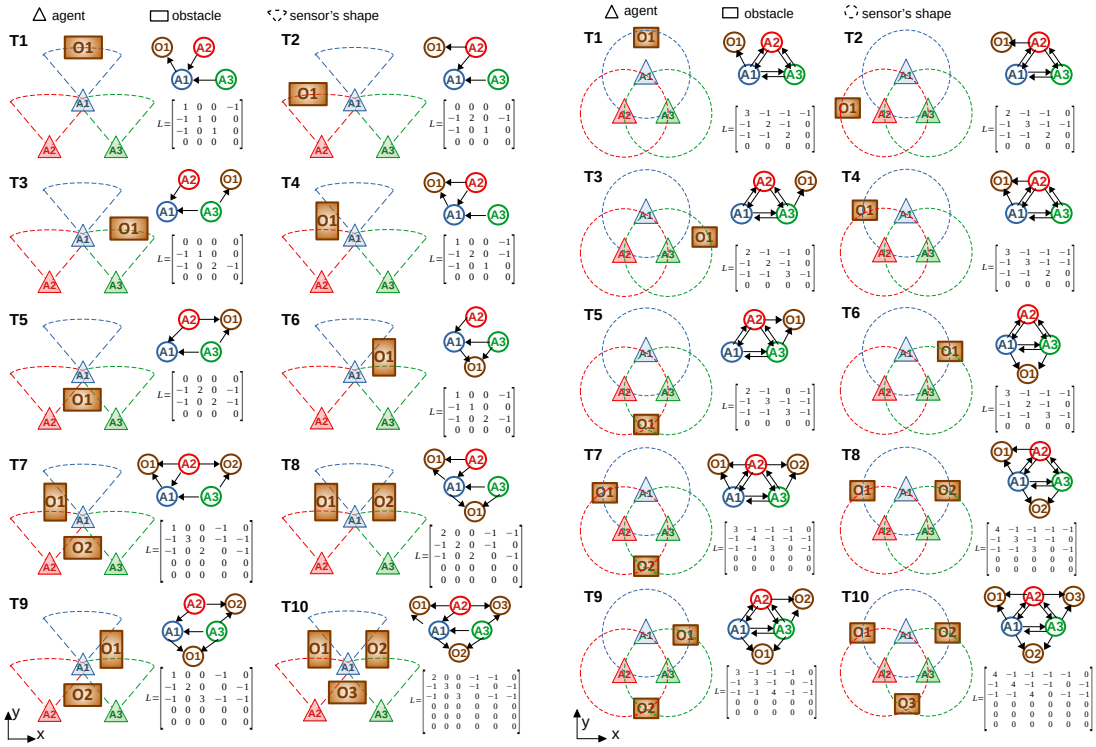


Figure 3.8: Configurations with additional obstacles in VL formations with conical (a) and circular (b) sensor's shape

Sensor & Formation	**T**	***a*:** y_1,y_2,y_3	***b*:** y_1,y_2	***c*:** y_2,y_3	***d*:** y_1,y_3	***e*:** y_1	***f*:** y_2	***g*:** y_3
conical SHL	T1	o						
	T2	o						
	T3	o						
	T4	o						
	T6	o						
	T7	o						
circular SHL	T1	o	o		o			
	T2	o						
	T3	o		o	o			
	T4	o	o		o			
	T6	o		o	o			
	T7	o						

	T1	o	o					
	T2	o						
conical	T3	o			o			
SVL	T4	o						
	T6	o		o				
	T7	o						
	T1	o	o		o			
	T2	o						
circular	T3	o		o	o			
SVL	T4	o	o		o			
	T6	o		o	o			
	T7	o						
	T1	o						
	T2	o		o				
	T3	o		o				
	T4	o		o				
conical	T5	o						
VL	T6	o		o				
	T7	o						
	T8	o						
	T9	o						
	T10	o						
	T1	o	o		o			
	T2	o	o	o				
	T3	o		o	o			
	T4	o		o	o			
circular	T5	o	o					
VL	T6	o	o	o				
	T7	o						
	T8	o						
	T9	o						
	T10	o						

Table 3.3: Observability results in different formations by adding obstacles

Formation	**Input**	***a*:** y_1,y_2,y_3	***b*:** y_1,y_2	***c*:** y_2,y_3	***d*:** y_1,y_3	***e*:** y_1	***f*:** y_2	***g*:** y_3
VL	A1							
	A2							
	A3							
	A1-A2	c		c	c			c
	A2-A3	c	c		c	c		
	A1-A3	c	c	c			c	
SVL	A1	c		c				
	A2							
	A3	c	c					
	A1-A2	c		c	c			c
	A2-A3	c	c		c	c		
	A1-A3	c	c	c			c	
SHL	A1	c		c				
	A2							
	A3	c	c					
	A1-A2	c		c	c			c
	A2-A3	c	c		c	c		
	A1-A3	c	c	c			c	

Table 3.4: Table with the controllability results (L changes with the cases)

By comparing the obtained results, it can be seen how the additional states do not improve but, rather, affect the observability of the system. Note that in table the arc-shaped sensor is not reported.
For the sake of completeness, taking into account only the formation given by the circular shaped sensor, the analysis is extended to the controllability, as the system derived from this topology is observable in more cases and it will be applied in the next simulations. Recalling that a system is said to be controllable if the controllability matrix

$$\mathcal{C} = \begin{pmatrix} B & A \cdot B & \dots & A^2 \cdot B \end{pmatrix} = \begin{pmatrix} B & (-\mathcal{L}) \cdot B & \dots & (-\mathcal{L})^2 \cdot B \end{pmatrix}$$

calculated as above for a three-agent formation, is full rank, the analysis is performed only in the case in which no obstacle is detected and impede the trajectory.
The results are collected in the Table 3.4, showing that, as in the observability analysis, the two SHL and SVL formations are controllable in more cases than in the VL formation.
During the analysis, the question raises whether it is fair to consider a Laplacian

Sensor	**Formation**	***a*:** y_1,y_2,y_3	***b*:** y_1,y_2	***c*:** y_2,y_3	***d*:** y_1,y_3	***e*:** y_1	***f*:** y_2	***g*:** y_3
conical	SVL	o	o		o	o		
	SHL	o				o		
	VL	o		o		o		
circular	SVL	o	o	o	o	o		o
	SHL	o	o	o	o	o		o
	VL	o	o	o	o	o		
arched	SVL	o	o		o	o		
	SHL	o	o	o	o	o		o
	VL	o	o	o	o	o	o	o

Table 3.5: Table with the observability results for conical and circular sensors in SVL, SHL, VL formations, considering different cases

Formation	A_1**:**	A_2**:**	A_3**:**	A_1-A_2**:**	A_2-A_3**:**	A_1-A_3**:**
VL				c	c	c
SVL	c		c	c	c	c
SHL	c		c	c	c	c

Table 3.6: Table with the controllability results (L in the best case)

matrix changing case by case or whether, instead, the Laplacian should reflect the ideal case, in which all the sensors are properly working, and let only the C matrix to represent the failure in one or more outputs.
Again, the study of observability and controllability are repeated in the case in which no obstacles are encountered during the trajectory, resulting in the Tables 3.5 and 3.6, the latter studied again only for the formations with circular shaped sensor, as it gives the best results in terms of observability. From the obtained tables, it can be deduced that the observability results aren't affected with exception of the VL formation, in which the case e becomes observable in every sensor's shape.
The controllability analysis is, instead, streamlined and, overally, improved.
In conclusion, it will be reasonable to consider from now on the Laplacian to always be the ideal one and to include the failures in the model thanks to the C matrix, representing the system's outputs.

3.3 Simulation of Kinematic Consensus Protocol

Afterwards, the kinematic consensus algorithm is tested on a model of three agents developed in MATLAB&Simulink R2019b environment.
In general the simulation is composed by a main script in which the initial positions of the three agents, the simulation parameters and the plotting indications are defined. The main script is associated to a Simulink model in which the topology and the kinetic consensus equation are specified in a function, in order to determine the actual position and velocity of each agent.
The simulations start from the simplest problem statement and incrementally add details to provide a realistic description of the kinetic consensus problem.

Reaching of the CoM

As first instance, the aim is to verify the kinematic consensus algorithm and, accordingly, the convergence of three agents in a unique point, the CoM, starting from different initial positions.
In the simulation, only the case of the spherical sensor is considered, because it is easier to consider the same and constant degree of perception on the x and y axes, such as the systems with the spherical sensor are observable in more cases in comparison to the other sensor's shapes.
The main script is shown below:

```
%--------------------------------------------------------------------------
%                              XY Initial conditions
%--------------------------------------------------------------------------
  x0_1  =  0;    y0_1  =  0;
  x0_2  = 10;    y0_2  =  5;
  x0_3  =  0;    y0_3  = 10;
  xi0   = [ x0_1; y0_1; x0_2; y0_2; x0_3; y0_3];

  cons_time =      0;
  rad2deg   = 180/pi;
  Ts        =   0.05;      %sampling time

%--------------------------------------------------------------------------
%                                SIM Configuration
%--------------------------------------------------------------------------
  delay  =  0;
  Tsim   = 15;
  sim('SimXYkin',Tsim);

%--------------------------------------------------------------------------
%                              TIME of reaching consensus
%--------------------------------------------------------------------------
   for i = 1:length(t)
       if (abs(xi_dot(i,:)) < ones(1,6)*0.01)
           cons_time = t(i);
           i = 1000;
```

```
27              break
28          end
29      end
30      cons_time
31
32  %-------------------------------------------------------------------------
33  %                                   PLOTS
34  %-------------------------------------------------------------------------
35    xi_x = [xi(:,1),xi(:,3),xi(:,5)];
36    xi_y = [xi(:,2),xi(:,4),xi(:,6)];
37    xi_dot_x = [xi_dot(:,1),xi_dot(:,3),xi_dot(:,5)];
38    xi_dot_y = [xi_dot(:,2),xi_dot(:,4),xi_dot(:,6)];
39
40    lw = 1;     t = tout(:,1);
41
42  figure(1)
43      subplot(2,1,1)
44          px = plot(t, xi_x, 'LineWidth', lw);
45          set(px, {'Color'}, {'b';'r';'g'});
46          xlabel('$t$[sec]')
47          ylabel('$\xi_x$ [m]')
48          legend('A_1', 'A_2', 'A_3');
49      subplot(2,1,2)
50          py = plot(t, xi_y, 'LineWidth', lw);
51          set(py, {'Color'}, {'b';'r';'g'});
52          xlabel('$t$[sec]')
53          ylabel('$\xi_y$ [m]')
54          legend('A_1', 'A_2', 'A_3');
55
56  figure(2)
57      subplot(2,1,1)
58          px = plot(t, xi_dot_x, 'LineWidth', lw);
59          set(px, {'Color'}, {'b';'r';'g'});
60          xlabel('$t$[sec]')
61          ylabel('$\xi_dot_x$ [m/s]')
62          legend('A_1', 'A_2', 'A_3');
63      subplot(2,1,2)
64          py = plot(t, xi_dot_y, 'LineWidth', lw);
65          set(py, {'Color'}, {'b';'r';'g'});
66          xlabel('$t$[sec]')
67          ylabel('$\xi_dot_y$ [m/s]')
68          legend('A_1', 'A_2', 'A_3');
69
70  figure(3)
71    px = plot(xi_x, xi_y, 'LineWidth', lw);
72    set(px, {'Color'}, {'b';'r';'g'});
73    hold on;      grid on;
74    a1 = plot(xi_x(length(t),1),xi_y(length(t),1), 'Color','b','Marker','o');
75    a2 = plot(xi_x(length(t),2),xi_y(length(t),2),'Color','r','Marker','o');
76    a3 = plot(xi_x(length(t),3),xi_y(length(t),3),'Color','g','Marker','o');
77    xlabel('$\xi_x$[m]')
78    ylabel('$\xi_y$ [m]')
79    legend('A_1','A_2','A_3');
```

In the line 18 of the main script, the Simulink model in Figure 3.9 is called.
In particular, the model is composed by the function that, applying the kinetic consensus equation (eq.(2.10)), gives as output the velocity of each agent, from the feedback of the actual position.
The Topology function script follows:

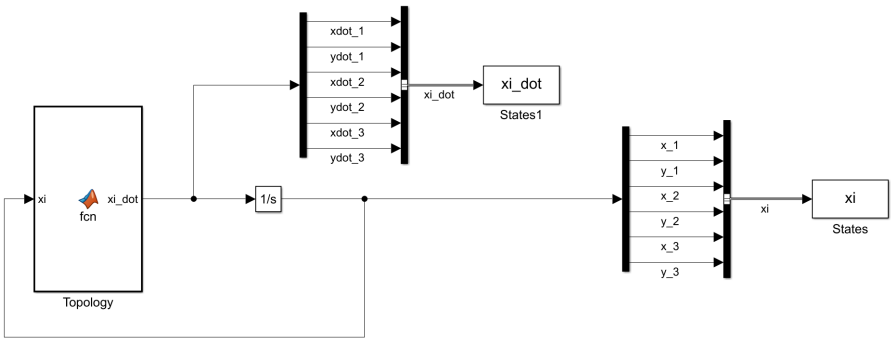


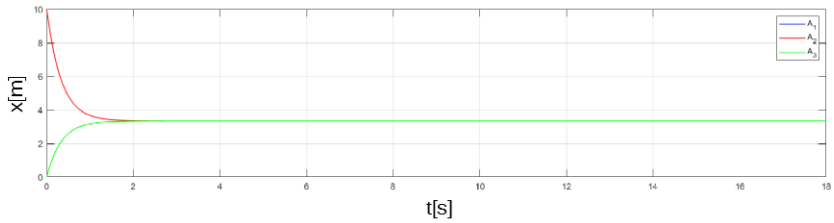
Figure 3.9: Simulink

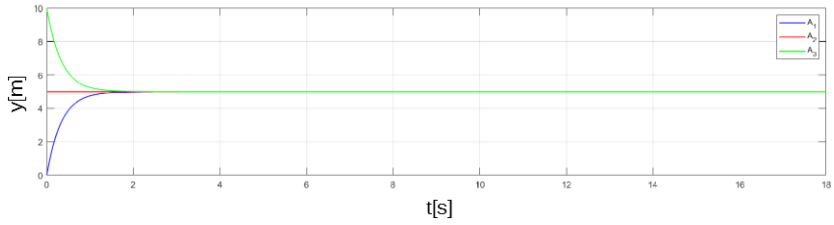
```
1  function xi_dot = fcn(xi)
2  %------------------------------------------------------------------------
3  %                          Definition of Topology
4  %------------------------------------------------------------------------
5  L = [a11 a12 a13; a21 a22 a23; a31 a32 a33];
6
7  %------------------------------------------------------------------------
8  %                        Transformation from 1D to 2D
9  %------------------------------------------------------------------------
10   Im  = [1 0; 0 1];
11   Kp  = diag(ones(3,1));
12   Lm  = kron(Kp*L,Im);        %Kronocker product
13
14 %------------------------------------------------------------------------
15 %                          Convergence at CoG
16 %------------------------------------------------------------------------
17   xi_tilde = xi;
18
19 %------------------------------------------------------------------------
20 %                            Consensus term
21 %------------------------------------------------------------------------
22   xi_dot = -(Lm * xi_tilde);
```

By changing the Laplacian matrix in the line 5 of the Topology script, is possible to experiment the different topologies.

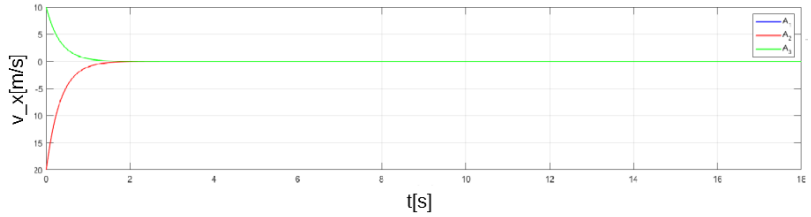
Example of VL formation:
As first example, it could be considered the VL-shaped formation, with an undirected and connected topology as in Figure 3.5, case *a*.
From the simulation the following plots are displayed:

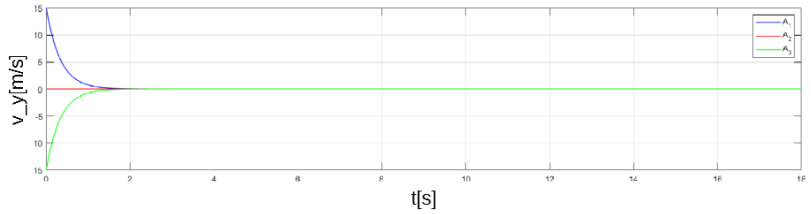
- Figure 1: x and y positions with respect of time (Figure 3.10(a));
- Figure 2: x and y velocities with respect of time (Figure 3.10(b));
- Figure 3: y-position with respect to x-position (Figure 3.10(c)).



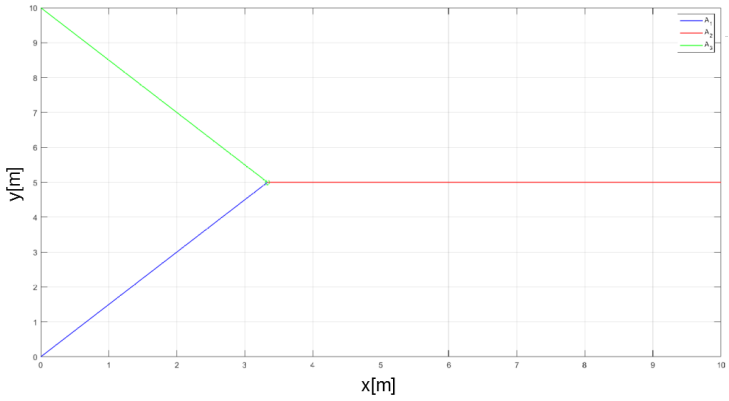


(a)





(b)



(c)

Figure 3.10: CoG reaching without considering an offset in VL formation, case (a)

As said before, the agents converge from the initial positions to a common point, that corresponds to the CoM, in the general case:

$$x_{CoG} = \frac{m_{A1}x_{0_{A1}} + m_{A2}x_{0_{A2}} + m_{A3}x_{0_{A3}}}{m_{A1} + m_{A2} + m_{A3}} = \frac{x_{0_{A1}} + x_{0_{A2}} + x_{0_{A3}}}{3}$$

$$y_{CoG} = \frac{m_{A1}y_{0_{A1}} + m_{A2}y_{0_{A2}} + m_{A3}y_{0_{A3}}}{m_{A1} + m_{A2} + m_{A3}} = \frac{y_{0_{A1}} + y_{0_{A2}} + y_{0_{A3}}}{3}$$

and in the specific example:

$$x_{CoG} = \frac{0 + 10 + 0}{3} \approx 3.33 \qquad y_{CoG} = \frac{0 + 5 + 10}{3} = 5 \tag{3.4}$$

In addition, the moment in which all the velocities reach the zero value, represents the consensus reaching time.
The same experiment was repeated considering the different formations and the seven cases in which the sensors could stop with the usual working operation.
It follows a list resuming the results:

- In Figure 3.11(a), the final aim is the VL formation and A_3 isn't aware of the other two agents and so it is stuck in his initial position. A_1 and A_2, that can sense A_3, will try to reach the consensus by moving together to the A_3;
- In Figure 3.11(b), the final aim is the VL formation and both the sensors of A_2 and A_3 aren't working in a proper way. Instead, A_1 has information about both and it will move to the middle point between the two, corresponding with the CoM of the system;
- In Figure 3.11(c) again, A_3 cannot sense the other two agents but in this case the agents want to reach a SHL formation. The result is the same of Figure 3.11(a), with a similar trajectory;
- In Figure 3.11(d) only the sensor of A_1 is working and it can sense only A_2, according to the SHL formation, so it will reach consensus by moving to A_2, ignoring the existence of A_3.

In conclusion, when agent's sensor is not working correctly, the agent remains stuck to its initial position, the remaining agents try to reach consensus considering only the information they already have, corresponding to the different topologies.
This result is totally coherent to what already cited in the Section 2.7, i.e. the kinetic consensus algorithm doesn't specify the agreement value: in case of isolated agent, the agreement value will be computed only considering the sub-group of communicating agents.
In the light of the results, can be demonstrated the dependency between a strongly

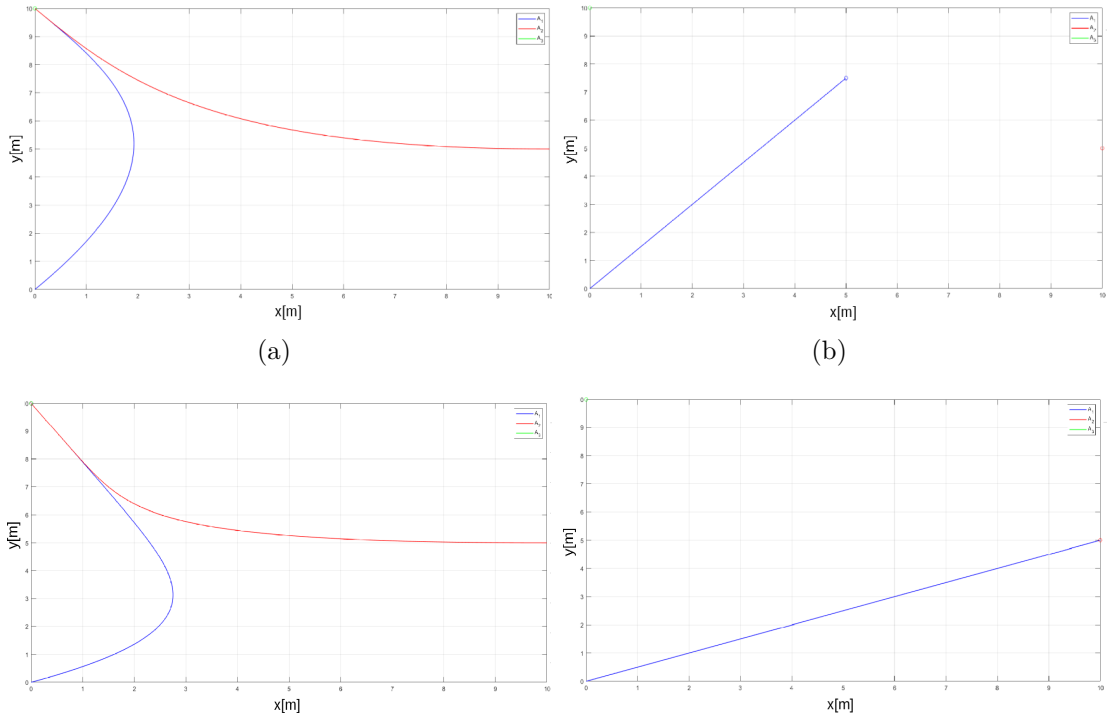


Figure 3.11: CoG reaching without considering an offset of a VL formation topology

connected graph and reaching of consensus.
Recalling that a graph is connected when, considering the set of nodes $(i, j) \in \mathbb{V}$, there is a path between any node i and node j, a graph is said to be *strongly* connected when between the two nodes i, j there is a directed path from i to j and viceversa.
In particular:

- If a graph is strongly connected, then its matrix is irreducible [54];
- Consider the Laplacian matrix of a strongly connected graph, it is irreducible and [55]:
  1. $\mathcal{L}\mathbf{1}_N = 0$;
  2. there is a positive vector $\xi = (\xi_1, \xi_2, ..., \xi_N)^T$, such that $\xi^T \mathcal{L} = 0$;
  3. there exists a positive-definite diagonal matrix $\Xi = diag(\xi_1, \xi_2, ..., \xi_N)$, such that $\hat{\mathcal{L}} = \frac{\Xi\mathcal{L}+\mathcal{L}^T\Xi}{2}$ is a symmetric matrix and $\sum_{j=1}^{N} \hat{l}_{ij} = \sum_{j=1}^{N} \hat{l}_{ji} = 0$ for all $i = 1,2, ..., N$.

Variable	**case *a*:** y_1,y_2,y_3	**case *b*:** y_1,y_2	**case *c*:** y_2,y_3	**case *d*:** y_1,y_3	**case *e*:** y_1	**case *f*:** y_2	**case *g*:** y_3
$t_{cons_{VL}}$	2.54	6.64	6.64	6.92	3.66	3.82	3.66
$t_{cons_{SVL}}$	6.22	15.44	15.44	6.92	6.92	3.82	3.82
$t_{cons_{SHL}}$	6.22	15.44	15.44	6.92	6.92	3.82	3.82
$\lambda_{2_{VL}}$	3	1	1	1	0	0	0
$\lambda_{2_{SVL}}$	1	0.382	0.382	1	0	0	0
$\lambda_{2_{SHL}}$	1	0.382	0.382	1	0	0	0

Table 3.7: Table collecting time needed to reach consensus and the algebraic connectivity in VL, SVL, SHL formations, considering different cases

- For a strongly connected graph with Laplacian matrix $\mathcal{L}$, let's define [13]:

$$a(\mathcal{L}) = \min_{x^T\xi=0, x\neq 0} \frac{x^t\hat{\mathcal{L}}x}{x^T\Xi x} \qquad b(\mathcal{L}) = \max_{x^T\xi=0, x\neq 0} \frac{x^t\hat{\mathcal{L}}x}{x^T\Xi x}$$

  and call $a(\mathcal{L})$ *General Algebraic Connectivity*. $a(\mathcal{L})$ is greater than 0 if and only if the graph is connected[56][57].

If $\Xi = \mu I_N$ and the network is undirected, by calculating the eigenvalues of the Laplacian matrix $\mathcal{L}$ of the graph $\mathcal{G}$, they can be ordinated as:

$$0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n.$$

λ_2, corresponding to the second smallest eigenvalue of $\mathcal{L}$, is called *Algebraic Connectivity* of $\mathcal{G}$, as in this case $a(\mathcal{L}) = \lambda_2$. In addition, $b(\mathcal{L}) = \lambda_N$.
Comparing VL, SVL and SHL-shaped formations, in the best case (all the outputs are known), the corresponding Laplacian matrices are respectively:

$$\mathcal{L}_{VL} = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}; \qquad \mathcal{L}_{SVL} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}; \qquad \mathcal{L}_{SHL} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}.$$

and the diagonal matrices containing the eigenvalues of each Laplacian:

$$\mathcal{D}_{VL} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix}; \qquad \mathcal{L}_{SVL} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}; \qquad \mathcal{L}_{SHL} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}.$$

Such that $\lambda_{2_{VL}} = 3$, $\lambda_{2_{SVL}} = 1$, $\lambda_{2_{SHL}} = 1$.
The same calculation is repeated for every case in the three formation, as well as the measure of the time needed to reach the consensus and the results are collected

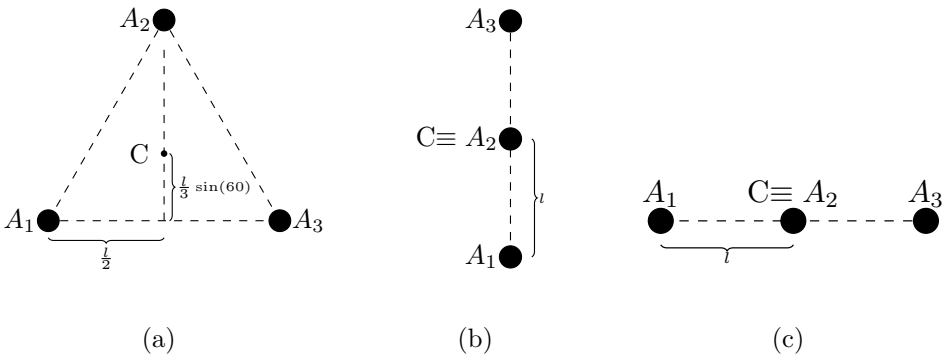


Figure 3.12: Addition of offset in the three formations

in the Table 3.7.
Making a comparison between the gathered data, it's evident that bigger is the algebraic connectivity, that means, more the graph is connected, faster (in a exponential fashion) the consensus is reached [7].
The result is coherent with what cited before: when the graph is not connected, the consensus is reached only for the agents in the sub-connected graph, the not connected nodes will be remain stuck in their original position.

Addition of offset

The second step was to define an offset between the agents, considering that real agents have a physical encumbrance.
In this case, a fixed offset is considered, to emulate the reality, in which the agents will respect a minimum safety distance between each other, to avoid collisions.
Considering the fixed offsets means that the whole formation can be compared to a rigid body. In the future, the interaction between two agents should be modeled as a damped mass-spring system.
In the three different formations, the offsets were defined taking into account the CoM as the origin of the relative system of reference. The final positions of each agent are defined as following:

- VL formation (Figure 3.12(a)):

$$\boldsymbol{p_{A_1}} = \left[-\tfrac{l}{2}, -\tfrac{l}{3}\sin 60\right] \qquad \boldsymbol{p_{A_2}} = \left[0, \tfrac{2l}{3}\sin 60\right]; \qquad \boldsymbol{p_{A_3}} = \left[\tfrac{l}{2}, -\tfrac{l}{3}\sin 60\right]. \tag{3.5}$$

- SVL formation (Figure 3.12(b)):

$$\boldsymbol{p_{A_1}} = [0, -l]; \qquad \boldsymbol{p_{A_2}} = [0,0]; \qquad \boldsymbol{p_{A_3}} = [0, l]. \tag{3.6}$$

- SHL formation (Figure 3.12(c)):

$$\boldsymbol{p_{A_1}} = [-l,0] \qquad \boldsymbol{p_{A_2}} = [0,0]\,; \qquad \boldsymbol{p_{A_3}} = [l,0]\,. \tag{3.7}$$

with l the distance between two agents, as figured in Figure 3.12.

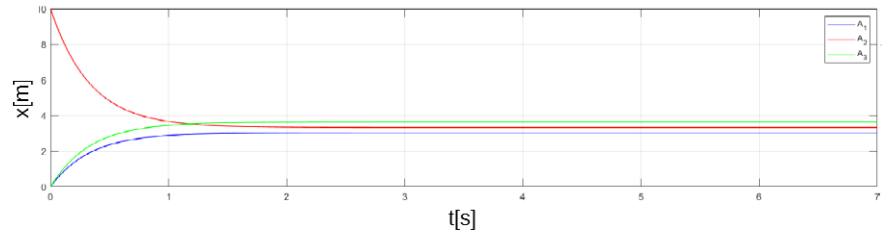
Example of VL formation:
Taking into account the example of a VL formation without offsets, the following lines were added in the Topology function script:

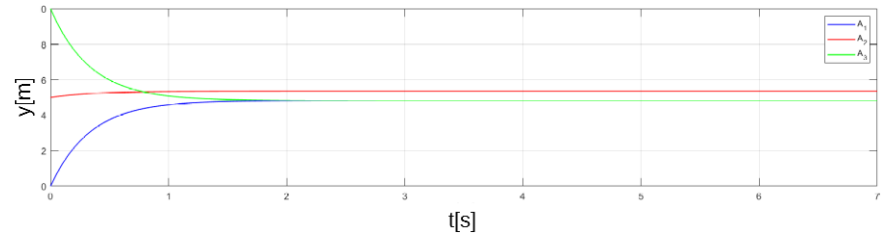
```
14 %------------------------------------------------------------------------
15 %                              Offset addition
16 %------------------------------------------------------------------------
17 L=0.6;
18 offset_x = [0, L/2, -L/2]';
19 offset_y = [(2/3)*L*sin(pi/3), -(1/3)*L*sin(pi/3), -(1/3)*L*sin(pi/3)]';
20 offset = [offset_x(1,1); offset_y(1,1); offset_x(2,1); offset_y(2,1); offset_x
       (3,1); offset_y(3,1)];
21
22 %------------------------------------------------------------------------
23 %                              Convergence at CoG
24 %------------------------------------------------------------------------
25  xi_tilde = xi - offset;
26
27 %------------------------------------------------------------------------
28 %                              Consensus term
29 %------------------------------------------------------------------------
30 xi_dot = -(Lm * xi_tilde);
```

So, firstly the offset between agents is defined, as in the eq,3.5, the offset is subtracted from the actual position and, at the end, the consensus term is calculated.
As in the previous example of VL formation, three different plots are reported:

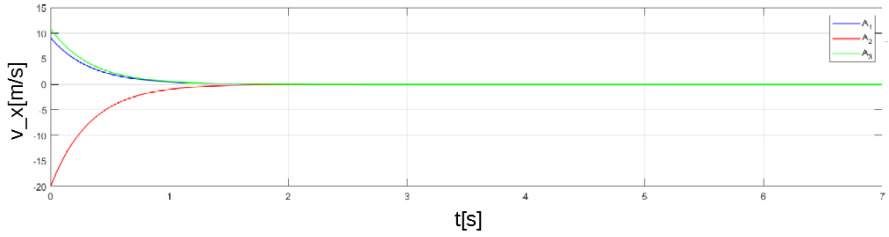
- Figure 1: x and y positions with respect of time (Figure 3.13(a));
- Figure 2: x and y velocities with respect of time (Figure 3.13(b));
- Figure 3: y-position with respect to x-position (Figure 3.13(c)).

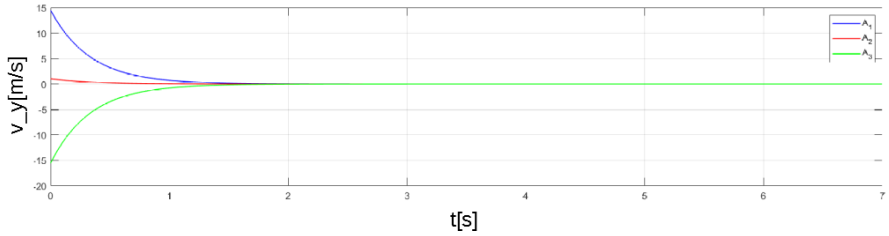
Changing the offset, as the ones in the eq.3.6 and eq.3.7, resulting in the graphs in Figure 3.14.



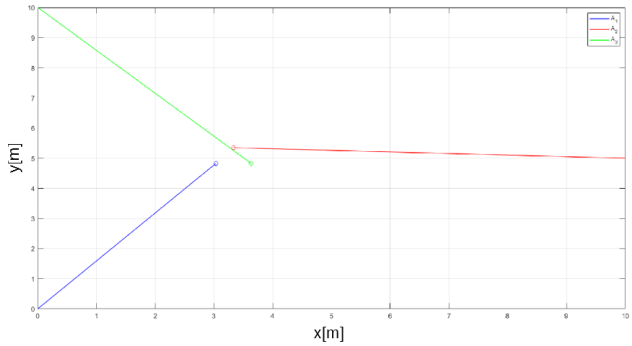


(a)



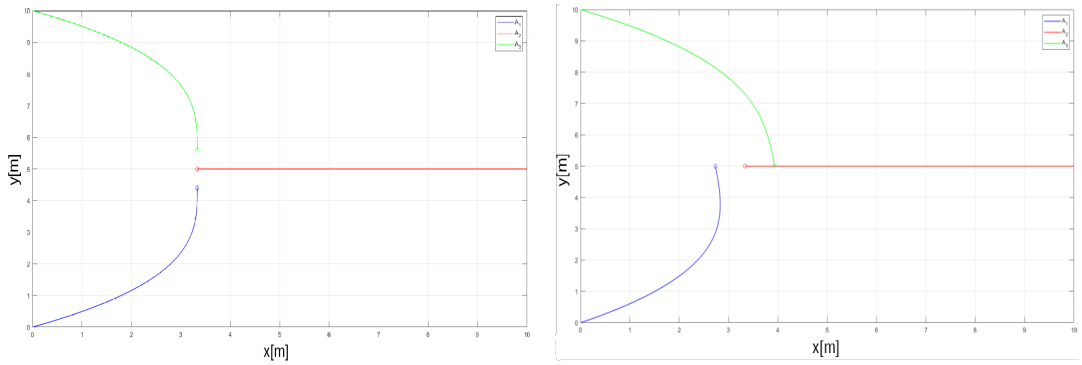


(b)



(c)

Figure 3.13: Examples of CoG reaching considering the offset between the agents, in VL formation



(a) (b)

Figure 3.14: Addition of offset in SVL (a) and SHL (b) formations

Definition of a leader

In this new step, the main objective is to define a leader, that is the only that is aware of the desired trajectory, and let the other agents follow the leader. Until now, the desired trajectory wasn't specified, so a new function block is added to the Simulink model, as in Figure 3.15, that, receiving as input the current time, gives as output the desired position, velocity and acceleration.
The desired coordinates will be sent to the function Topology.
In Desired Trajectory script were defined the desired constant final position (time-invariant desired trajectory) and a desired circular trajectory (time variant desired trajectory). The script is the following:

```
function [hmotiondx,hmotiondy] = fcn(t)
f      =  0.1;          %freq in Hz
radius =    3;      %radius in meters
sel    =    2;
Tsim   =   25;
Ts     = 0.05;
i      =    0;

%-------------------------------------------------------------------------
%                              Desired Trajectory
%-------------------------------------------------------------------------
if(sel==2)                %% constant setpoint
    xd     =  6;
    xdotd  =  0;
    xddotd =  0;

    yd     =  2;
    ydotd  =  0;
    yddotd =  0;
end

```

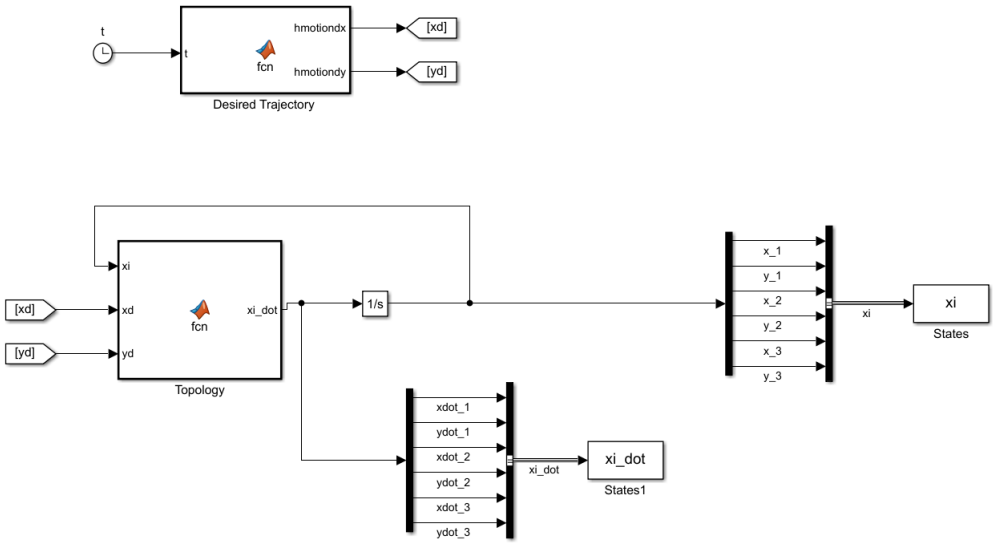


Figure 3.15: Modified version of Simulink model

```
if(sel==3)                %% circular trajectory
    xd      =   radius*cos(2*pi*f*t);
    xdotd   = - radius*2*pi*f*sin(2*pi*f*t);
    xddotd  = - radius*cos(2*pi*f*t)*((2*pi*f)^(2)) ;

    yd      =   radius*sin(2*pi*f*t);
    ydotd   =   radius*2*pi*f*cos(2*pi*f*t);
    yddotd  = - radius*sin(2*pi*f*t)*((2*pi*f)^(2));
end

hmotiondx   =   [xd; xdotd; xddotd];
hmotiondy   =   [yd; ydotd; yddotd];
```

Once that the Desired Trajectory function is created, the desired coordinates are sent to the Topology function and they will influence the consensus term.
In the experiments it was introduced the Γ matrix that stabilizes which agent is aware of the trajectory and, then, is the leader. Assuming that only the first agent has the information about the desired position, the Γ matrix is constructed as:

$$\Gamma = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.8}$$

The square matrix Γ will multiplied by the desired trajectory vector

$$\boldsymbol{x_d} = [x_{1_d};\ y_{1_d};\ x_{2_d};\ y_{2_d};\ x_{3_d};\ y_{3_d}] \tag{3.9}$$

meaning that only the information related to x_{1_d} and y_{1_d} will be taken into account. Moreover, there will be considered two cases, affecting the Laplacian matrix:

- The leader is aware of the other agents;
- The leader isn't aware of the other agents, more realistic in the case of the definition of a leader-slave structure.

Firstly, the consensus term is modified by the addition of the a navigational feedback term[10], giving the knowledge of the desired final position.

$$\begin{cases} \tilde{x_i} = x_i - x_{offset_i}, & i \in \mathbb{V} \\ \dot{x_i} = -\alpha \cdot \mathcal{L} \cdot \tilde{x_i} - K_P \cdot \Gamma \cdot (x_i - x_d) + \Gamma \cdot \dot{x}_d, & i \in \mathbb{V} \end{cases} \tag{3.10}$$

The Topology script is modified adding the following lines:

```
%------------------------------------------------------------------------
%                          Desired trajectory terms
%------------------------------------------------------------------------
x_desired    = [xd(1,1); yd(1,1); xd(1,1); yd(1,1); xd(1,1); yd(1,1)];
xdot_desired = [xd(2,1); yd(2,1); xd(2,1); yd(2,1); xd(2,1); yd(2,1)];

Gamma=[1 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 0 0 0 0 0];

%------------------------------------------------------------------------
%                             Convergence at CoG
%------------------------------------------------------------------------
 xi_tilde     = xi - x_offset;

%------------------------------------------------------------------------
%                             Consensus term
%------------------------------------------------------------------------
xi_dot = - (Lm * xi_tilde) - Gamma*(xi - x_desired) + Gamma*xdot_desired;
```

In general, let's define the desired trajectory as:

$$\boldsymbol{p_d} = [x_d,\ y_d] \qquad \boldsymbol{\dot{p}_d} = [\dot{x}_d,\ \dot{y}_d] \qquad \boldsymbol{\ddot{p}_d} = [\ddot{x}_d,\ \ddot{y}_d]$$

And for the specific cases:

- Desired fixed position:

$$\boldsymbol{p_d} = [x_d,\ y_d] \qquad \boldsymbol{\dot{p}_d} = [0,\ 0] \qquad \boldsymbol{\ddot{p}_d} = [0,\ 0]$$

- Desired time-varying position:

$$\begin{aligned} \boldsymbol{p_d} &= [R \cdot \cos(2\pi f \cdot t),\ R \cdot \sin(2\pi f \cdot t)] \\ \boldsymbol{\dot{p}_d} &= [-R \cdot 2\pi f \sin(2\pi f \cdot t),\ R \cdot 2\pi f \cos(2\pi f \cdot t)] \\ \boldsymbol{\ddot{p}_d} &= [-R \cdot (2\pi f)^2 \cos(2\pi f \cdot t),\ -R \cdot (2\pi f)^2 \sin(2\pi f \cdot t)] \end{aligned}$$

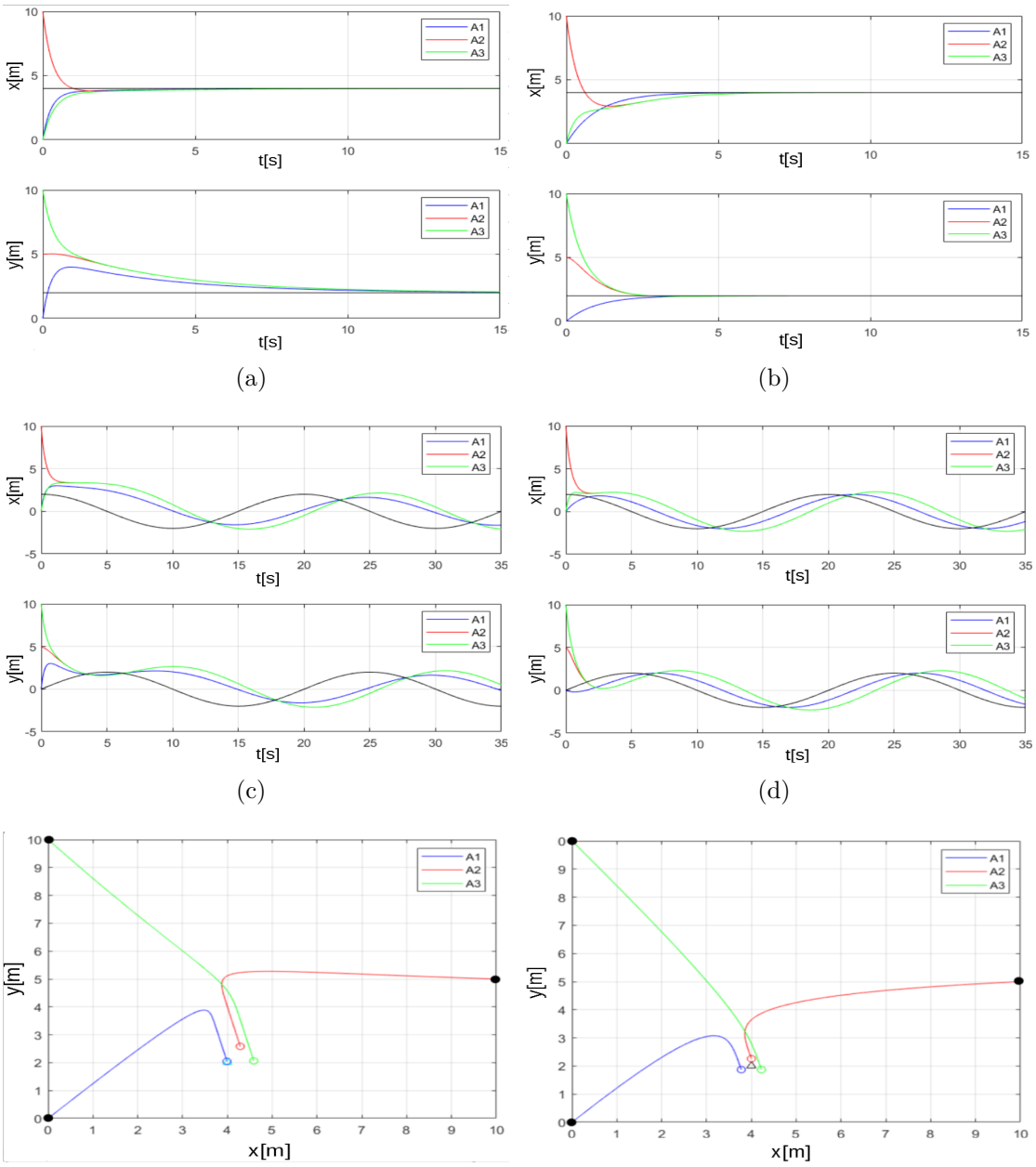


Figure 3.16: a) Fixed trajectory - A_1 is aware of the other agents, b) Fixed trajectory - A_1 isn't aware of the other agents, c) Varying trajectory - A_1 is aware of the other agents, d) Varying trajectory - A_1 isn't aware of the other agents, e) Fixed trajectory - only one leader (A_1), f) Fixed trajectory - all the agents are aware of the desired final position.

Results:

- Fixed desired trajectory - A_1 is aware of the slaves (Figure 3.16(a)): the consensus is reached in a short time, corresponding to the desired position. At first time, A_1 gets closer to the other agents and later tries to follow the desired trajectory;
- Fixed desired trajectory - A_1 is not aware of the slaves (Figure 3.16(b)): the consensus is reached but in a shorter time than the one needed in the case in which the leader is aware of the slaves. In fact, A_1 quickly tries to follow the desired trajectory and the other agents follow it as a consequence;
- Time-varying desired trajectory - A_1 is aware of the slaves (Figure 3.16(c)): as in the case of a fixed final position, the leader, that can perceive the other agents, firstly tries to get closer to them and later to follow the specified trajectory;
- Time-varying desired trajectory - A_1 is not aware of the slaves (Figure 3.16(d)): the trajectory is followed with a better approximation since A_1 directly follows the desired trajectory.

Important assumptions have to be done: the topology is considered fixed and the agents regulate their position and orientation basing on the knowledge of the Laplacian matrix[11]. Until now it was considered a three-agents formation but the same problem can be extended to a huge number of agents: the complexity could be reduced by defining a neighbourhood of agents, in which the topology is fixed. Notice that, when the desired trajectory is known by a single agent, the leader will reach the desired final position and the other agents will settle based on the leader position (Figure 3.16(e)), meaning that the final consensus term won't correspond to the desired position. Instead, in the case in which all the agents are aware of the desired final position, this one will correspond to the final consensus term (Figure 3.16(f)).

3.4 Simulation of Dynamic Consensus Protocol

The Simulink model for the dynamic consensus algorithm was developed starting from the model of the kinematic.
All the scripts are maintained: the *main.c* script defines the initial positions of the three agents, the simulation parameters and the plotting indications,the *Topology* script characterizes the topology and the dynamic consensus equation, the *Desired Trajectory* script is dedicated to the definition of the trajectory to follow.
Before detailing the various steps, some important assumptions have to be made:

- The algebraic connectivity, measure of the connectivity of the topologies, has to be large enough[50];
- the velocity of the agents is a time-varying variable resulting in a non-linear system;
- the communication between agents cannot be guaranteed all the whole time, due to errors in physical devices and external disturbances.

Achievement of velocity consensus

In the first step, the easiest dynamic consensus algorithm is considered, described by the following equation:

$$\begin{cases} \tilde{\boldsymbol{x}} = \boldsymbol{x} \\ \dot{\tilde{\boldsymbol{x}}} = \tilde{\boldsymbol{v}} \\ \dot{\boldsymbol{v}} = -\alpha \cdot \mathcal{L} \cdot \tilde{\boldsymbol{x}} - \beta \cdot \mathcal{L} \cdot \boldsymbol{v} \end{cases} \tag{3.11}$$

with α and β as weighting factors considered as equal to one, $\mathcal{L}$ the Laplacian matrix describing the topology. The Topology script is modified as follow:

```
function xi_dotdot   = fcn(xi_dot, xi)
i=0;

%------------------------------------------------------------------------------
%                                  Laplacian matrix
%------------------------------------------------------------------------------
L = [2 -1 -1; -1 2 -1; -1 -1 2];

%------------------------------------------------------------------------------
%                                     1D to 2D
%------------------------------------------------------------------------------
 Im = [1 0; 0 1];
 kp=1;
 Kp=diag([kp;kp;kp]) ;
 Lm = kron(Kp*L,Im);

%------------------------------------------------------------------------------
%                                  Convergence at CoG
%------------------------------------------------------------------------------
gamma=[1 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 0 0 0 0 0];
 xi_tilde = xi;
 xi_dot_tilde = xi_dot;

%------------------------------------------------------------------------------
%                                  Consensus term
%------------------------------------------------------------------------------
xi_dotdot = - Lm*xi_tilde - Lm*xi_dot_tilde;
```

In particular, from the function is outputting the acceleration $\ddot{\boldsymbol{x}}$.
From this model, the agreement value corresponds to the average velocity of the

initial velocities of the three agents, as shown in the Figure 3.17(a). In Figure 3.17(b), instead, the y-position is plotted with respect to the x-position: the position is changing in time as the agreement value corresponds to a constant velocity.

Definition of desired trajectory

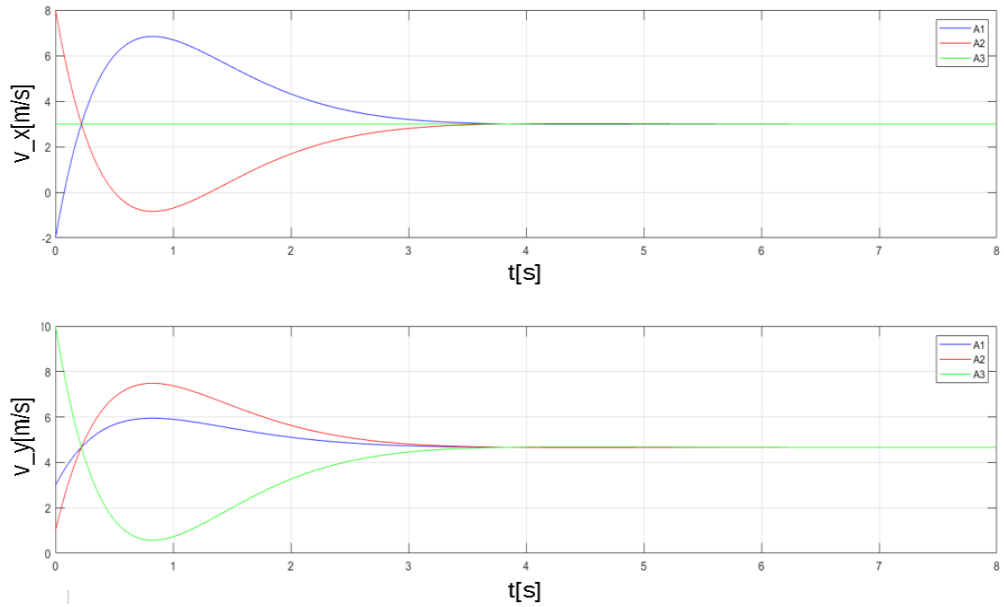
As before, in the definition of kinetic consensus algorithm, the next step consists in the specification of the desired trajectory.

The final equation, including the dynamic consensus term, the offsets between agents and the desired trajectory, is:

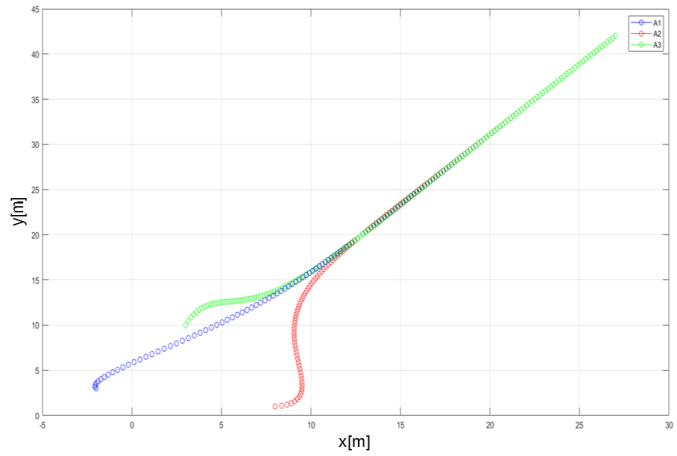
$$\begin{cases} \tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x_{offset}} \\ \dot{\tilde{\boldsymbol{x}}} = \tilde{\boldsymbol{v}} \\ \dot{\boldsymbol{v}} = -\alpha \cdot \mathcal{L} \cdot \tilde{\boldsymbol{x}} - K_P \cdot \Gamma(\boldsymbol{x} - \boldsymbol{x_d}) - \beta \cdot \mathcal{L} \cdot \boldsymbol{v} - K_D \cdot \Gamma\left(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_{\boldsymbol{d}}\right) + \Gamma \cdot \dot{\boldsymbol{x}}_{\boldsymbol{d}} \end{cases} \tag{3.12}$$

With, as before, α and β weighting factors equal to 1, $\mathcal{L}$ the Laplacian matrix describing the topology, Γ a matrix defining the leader, K_P and K_D the gains of the PD controller. The Topology script is modified as follow:

```
function xi_dotdot   = fcn(xi_dot, xi, xd, yd)
i=0;

%------------------------------------------------------------------------
%                              Laplacian matrix
%------------------------------------------------------------------------
L = [2 -1 -1; -1 2 -1; -1 -1 2];

%------------------------------------------------------------------------
%                                 1D to 2D
%------------------------------------------------------------------------
 Im = [1 0; 0 1];
 kp=1;
 Kp=diag([kp;kp;kp]) ;
 Lm = kron(Kp*L,Im);

%------------------------------------------------------------------------
%                            Desired coordinates
%------------------------------------------------------------------------
 p_desired=[xd(1,1); yd(1,1); xd(1,1); yd(1,1); xd(1,1); yd(1,1)];
 v_desired=[xd(2,1); yd(2,1); xd(2,1); yd(2,1); xd(2,1); yd(2,1)];
 a_desired=[xd(3,1); yd(3,1); xd(3,1); yd(3,1); xd(3,1); yd(3,1)];

%------------------------------------------------------------------------
%                            Convergence at CoG
%------------------------------------------------------------------------
 gamma=[1 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 0 0 0 0 0];
 xi_tilde = xi;
 xi_dot_tilde = xi_dot;

%------------------------------------------------------------------------
%                              Consensus term
%------------------------------------------------------------------------
```



(a)

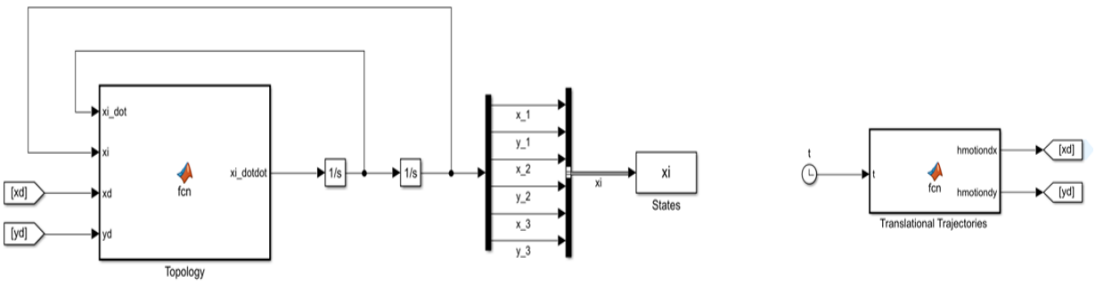


(b)

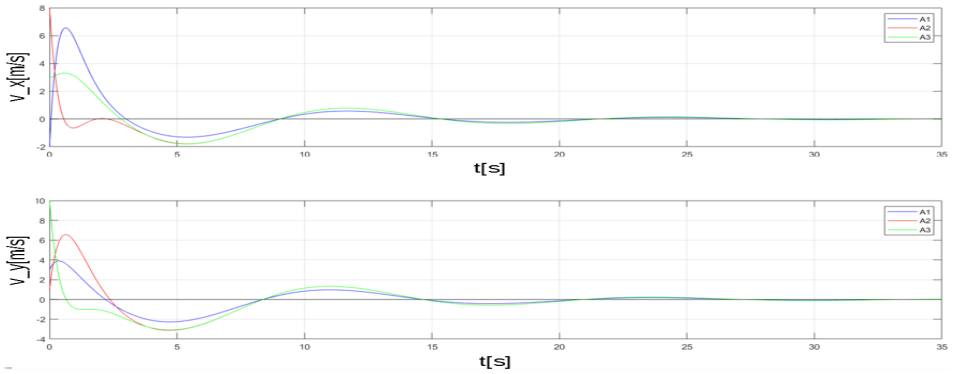
Figure 3.17: Dynamic consensus protocol: a) x and y-velocities with respect to time, b) y-position with respect to x-position

```
34 xi_dotdot = - Lm*xi_tilde - Lm*xi_dot_tilde -gamma*(xi_tilde-p_desired) - gamma*(
       xi_dot_tilde-v_desired) + gamma*a_desired;
```

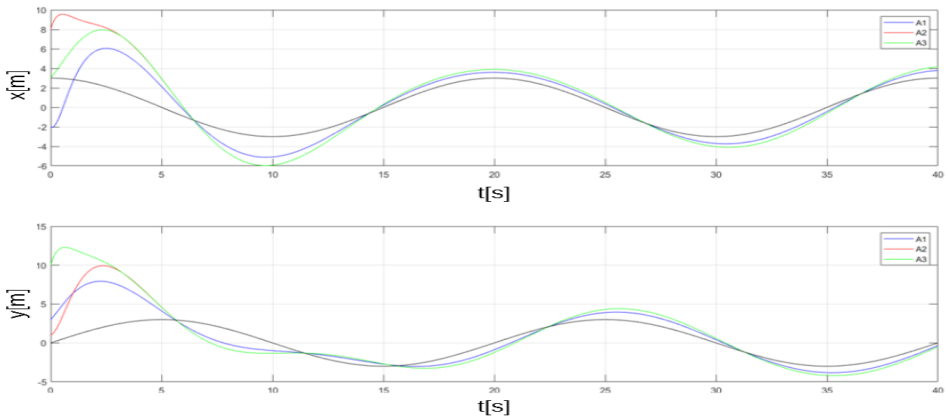
The final Simulink model and the related results are displayed in the Figure 3.18(a).



(a)



(b)



(c)

Figure 3.18: (a) Simulink model implementing the dynamic consensus protocol when the desired trajectory is specified, (b) x and y-velocities with respect to time in the case of fixed desired trajectory, (c) x and y-velocities with respect to time in the case of time-varying desired trajectory

3.5 Observer

Before the observer's design, the theory of State-Feedback is recalled: this technique allows to place the closed-loop poles of the plant in already-determined locations in the s-plane, in order to obtain the desired system response.
Consider the first-order system

$$\begin{cases} \dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} \\ \boldsymbol{y} = C\boldsymbol{x} \end{cases}$$

and impose the input vector as $\boldsymbol{u} = -K\boldsymbol{x}$, the new state-space representation is:

$$\begin{cases} \dot{\boldsymbol{x}} = (A - B \cdot K)\,\boldsymbol{x} \\ \boldsymbol{y} = C\boldsymbol{x} \end{cases}$$

By solving $det\,[sI - (A - BK)] = 0$, the new poles, characterizing the state-feedback system, can be found.
At this point, the observer can started to be constructed, as crucial in the model, in the case of missing states. The design starts from the first order state-space representation and it is later expanded to the second order one, as the latter is implemented in the model.

First order linear system

Starting from the time-invariant continuous-time system:

$$\begin{cases} \dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} \\ \boldsymbol{y} = C\boldsymbol{x} \end{cases}$$

The observed time-invariant physical continuous-time system is described as:

$$\begin{cases} \dot{\hat{\boldsymbol{x}}} = A\hat{\boldsymbol{x}} + B\boldsymbol{u} + L(\boldsymbol{y} - \hat{\boldsymbol{y}}) \\ \hat{\boldsymbol{y}} = C\hat{\boldsymbol{x}} \end{cases}$$

The observer is called *symptotically stable* if the observer error $\boldsymbol{e} = \boldsymbol{x} - \hat{\boldsymbol{x}}$ converges to 0 when going to infinity. Substituting the equations of the observer model in the equations of the original system, the following equation describing the derivative error is obtained:

$$\dot{\boldsymbol{e}} = \dot{\hat{\boldsymbol{x}}} - \dot{\boldsymbol{x}} = [A\hat{\boldsymbol{x}} + L\,(\boldsymbol{y} - \hat{\boldsymbol{y}}) + B\boldsymbol{u}] - [A\boldsymbol{x} + B\boldsymbol{u}] =$$
$$A\hat{\boldsymbol{x}} + LC\,(\boldsymbol{x} - \hat{\boldsymbol{x}}) + B\boldsymbol{u} - A\boldsymbol{x} - B\boldsymbol{u} = A\,(\hat{\boldsymbol{x}} - \boldsymbol{x}) - LC\,(\hat{\boldsymbol{x}} - \boldsymbol{x}) = [A - LC]\,\boldsymbol{e}$$

resuming:

$$\dot{\boldsymbol{e}} = [A - LC]\,\boldsymbol{e} \tag{3.13}$$

The eigenvalues of the matrix $A - LC$ can be chosen by tuning the observer gain L.
In general, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{u} \in \mathbb{R}^{n_u}$ $\boldsymbol{y} \in \mathbb{R}^{n_y}$.
The matrices are: $A \in \mathbb{R}^{n\times n}$,$B \in \mathbb{R}^{n\times n_u}$, $C \in \mathbb{R}^{n_y\times n}$, $L \in \mathbb{R}^{n\times n_y}$.

Second order linear system

An observer model in a second-order linear system can be described as:

$$\begin{cases} \boldsymbol{v} = \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{v}} = A_1\boldsymbol{x} + A_2\dot{\boldsymbol{x}} + B\boldsymbol{u} \\ \boldsymbol{y} = C\boldsymbol{x} \end{cases}$$

and, in matrix-representation:

$$\begin{cases} \begin{bmatrix} \dot{\boldsymbol{x}} \\ \ddot{\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} 0_{n\times n} & \mathbb{I}_{n\times n} \\ A_1 & A_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{bmatrix} + \begin{bmatrix} 0_{n\times 1} \\ B \end{bmatrix} \boldsymbol{u} \\ \boldsymbol{y} = \begin{bmatrix} C & 0_{1\times n} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{bmatrix} \end{cases}$$

As before, the observed time-invariant system is found as:

$$\begin{cases} \begin{bmatrix} \dot{\hat{\boldsymbol{x}}} \\ \ddot{\hat{\boldsymbol{x}}} \end{bmatrix} = \begin{bmatrix} 0_{n\times n} & \mathbb{I}_{n\times n} \\ A_1 & A_2 \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}} \\ \dot{\hat{\boldsymbol{x}}} \end{bmatrix} + \begin{bmatrix} 0_{n\times 1} \\ B \end{bmatrix} \boldsymbol{u} + \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \begin{bmatrix} C & 0_{1\times n} \end{bmatrix} (\boldsymbol{y} - \hat{\boldsymbol{y}}) \\ \hat{\boldsymbol{y}} = \begin{bmatrix} C & 0_{1\times n} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}} \\ \dot{\hat{\boldsymbol{x}}} \end{bmatrix} \end{cases}$$

resulting in the follow final equation:

$$\begin{bmatrix} \dot{\boldsymbol{e}}_{\boldsymbol{1}} \\ \dot{\boldsymbol{e}}_{\boldsymbol{2}} \end{bmatrix} = \left(\begin{bmatrix} 0_{n\times n} & \mathbb{I}_{n\times n} \\ A_1 & A_2 \end{bmatrix} - \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \begin{bmatrix} C & 0_{1\times n} \end{bmatrix} \right) \begin{bmatrix} \boldsymbol{e}_{\boldsymbol{1}} \\ \boldsymbol{e}_{\boldsymbol{2}} \end{bmatrix} =$$
$$\left(\begin{bmatrix} 0_{n\times n} & \mathbb{I}_{n\times n} \\ A_1 & A_2 \end{bmatrix} - \begin{bmatrix} L_1C & 0_{n\times n} \\ L_2C & 0_{n\times n} \end{bmatrix} \right) \begin{bmatrix} \boldsymbol{e}_{\boldsymbol{1}} \\ \boldsymbol{e}_{\boldsymbol{2}} \end{bmatrix} = \begin{bmatrix} -L_1C & \mathbb{I}_{n\times n} \\ A_1 - L_2C & A_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{e}_{\boldsymbol{1}} \\ \boldsymbol{e}_{\boldsymbol{2}} \end{bmatrix}$$

Then, the eigenvalues can be calculated by solving the following equation:

$$det \left| \begin{bmatrix} -L_1C & \mathbb{I}_{n\times n} \\ A_1 - L_2C & A_2 \end{bmatrix} - \begin{bmatrix} \Lambda_{n\times n} & 0_{n\times n} \\ 0_{n\times n} & \Lambda_{n\times n} \end{bmatrix} \right| =$$
$$\Lambda^2 + \Lambda\,(L_1C - A_2) + (L_2C - L_1CA_2 - L_2C)$$

By imposing the desired λ eigenvalues, L_1 and L_2 can be calculated.

Consensus algorithm

At this stage, the state-space representation can be modified by including the dynamic consensus algorithm in it:

$$\begin{cases} \boldsymbol{v} = \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{v}} = -\mathcal{L}\boldsymbol{x} - \mathcal{L}\dot{\boldsymbol{x}} - \Gamma\left(\boldsymbol{x} - \boldsymbol{x_d}\right) - \Gamma\left(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_{\boldsymbol{d}}\right) + \Gamma\ddot{\boldsymbol{x}}_{\boldsymbol{d}} \\ \boldsymbol{y} = C\boldsymbol{x} \end{cases}$$

In the specific, the first two equations, in matrix-representation, are:

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ \ddot{\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} 0_{n\times n} & \mathbb{I}_{n\times n} \\ -\mathcal{L} & -\mathcal{L} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{bmatrix} - \begin{bmatrix} 0_{n\times n} & 0_{n\times n} \\ \Gamma & \Gamma \end{bmatrix} \begin{bmatrix} \boldsymbol{x} - \boldsymbol{x_d} \\ \dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_{\boldsymbol{d}} \end{bmatrix} + \begin{bmatrix} 0_{n\times n} & 0_{n\times n} \\ 0_{n\times n} & \Gamma \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}}_{\boldsymbol{d}} \\ \ddot{\boldsymbol{x}}_{\boldsymbol{d}} \end{bmatrix}$$

and, then,

$$\begin{bmatrix} \mathbb{I}_{n\times n} & 0_{n\times n} \\ 0_{n\times n} & \mathbb{I}_{n\times n} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}} \\ \ddot{\boldsymbol{x}} \end{bmatrix} - \begin{bmatrix} 0_{n\times n} & 0_{n\times n} \\ 0_{n\times n} & \Gamma \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}}_{\boldsymbol{d}} \\ \ddot{\boldsymbol{x}}_{\boldsymbol{d}} \end{bmatrix} = \begin{bmatrix} 0_{n\times n} & \mathbb{I}_{n\times n} \\ -\mathcal{L} - \Gamma & -\mathcal{L} - \Gamma \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{bmatrix} - \begin{bmatrix} 0_{n\times n} & 0_{n\times n} \\ -\Gamma & -\Gamma \end{bmatrix} \begin{bmatrix} \boldsymbol{x_d} \\ \dot{\boldsymbol{x}}_{\boldsymbol{d}} \end{bmatrix}$$

Resulting in a Simulink model like the one displayed in the Figure 3.19.
The observed time-invariant system is modeled as it follows:

$$\begin{bmatrix} \dot{\hat{\boldsymbol{x}}} \\ \ddot{\hat{\boldsymbol{x}}} \end{bmatrix} = \begin{bmatrix} 0_{n\times n} & \mathbb{I}_{n\times n} \\ -\mathcal{L} & -\mathcal{L} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}} \\ \dot{\hat{\boldsymbol{x}}} \end{bmatrix} - \begin{bmatrix} 0_{n\times n} & 0_{n\times n} \\ \Gamma & \Gamma \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}} - \boldsymbol{x_d} \\ \dot{\hat{\boldsymbol{x}}} - \dot{\boldsymbol{x}}_{\boldsymbol{d}} \end{bmatrix} + \begin{bmatrix} 0_{n\times n} & 0_{n\times n} \\ 0_{n\times n} & \Gamma \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}}_{\boldsymbol{d}} \\ \ddot{\boldsymbol{x}}_{\boldsymbol{d}} \end{bmatrix} + L \begin{bmatrix} C & 0_{1\times n} \end{bmatrix} (\boldsymbol{y} - \hat{\boldsymbol{y}})$$

The error equation results:

$$\begin{bmatrix} \dot{\boldsymbol{e}}_{\boldsymbol{1}} \\ \dot{\boldsymbol{e}}_{\boldsymbol{2}} \end{bmatrix} = \begin{bmatrix} \dot{\hat{\boldsymbol{x}}} \\ \ddot{\hat{\boldsymbol{x}}} \end{bmatrix} - \begin{bmatrix} \dot{\boldsymbol{x}} \\ \ddot{\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} 0_{n\times n} & \mathbb{I}_{n\times n} \\ -\mathcal{L} - \Gamma & -\mathcal{L} - \Gamma \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}} \\ \dot{\hat{\boldsymbol{x}}} \end{bmatrix}$$
$$+ \begin{bmatrix} 0_{n\times n} & 0_{n\times n} \\ \Gamma & \Gamma \end{bmatrix} \begin{bmatrix} \boldsymbol{x_d} \\ \dot{\boldsymbol{x}}_{\boldsymbol{d}} \end{bmatrix} + \begin{bmatrix} 0_{n\times n} & 0_{n\times n} \\ 0_{n\times n} & \Gamma \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}}_{\boldsymbol{d}} \\ \ddot{\boldsymbol{x}}_{\boldsymbol{d}} \end{bmatrix} + L \begin{bmatrix} C & 0_{1\times n} \end{bmatrix} \left(\begin{bmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{bmatrix} - \begin{bmatrix} \hat{\boldsymbol{x}} \\ \dot{\hat{\boldsymbol{x}}} \end{bmatrix} \right) -$$
$$\left(\begin{bmatrix} 0_{n\times n} & \mathbb{I}_{n\times n} \\ -\mathcal{L} - \Gamma & -\mathcal{L} - \Gamma \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{bmatrix} + \begin{bmatrix} 0_{n\times n} & 0_{n\times n} \\ \Gamma & \Gamma \end{bmatrix} \begin{bmatrix} \boldsymbol{x_d} \\ \dot{\boldsymbol{x}}_{\boldsymbol{d}} \end{bmatrix} + \begin{bmatrix} 0_{n\times n} & 0_{n\times n} \\ 0_{n\times n} & \Gamma \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}}_{\boldsymbol{d}} \\ \ddot{\boldsymbol{x}}_{\boldsymbol{d}} \end{bmatrix} \right)$$
$$= \begin{bmatrix} 0_{n\times n} & \mathbb{I}_{n\times n} \\ -\mathcal{L} - \Gamma & -\mathcal{L} - \Gamma \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}} - \boldsymbol{x} \\ \dot{\hat{\boldsymbol{x}}} - \dot{\boldsymbol{x}} \end{bmatrix} - \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \begin{bmatrix} C & 0_{1\times n} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}} - \boldsymbol{x} \\ \dot{\hat{\boldsymbol{x}}} - \dot{\boldsymbol{x}} \end{bmatrix}$$

At the end, the final equation is:

$$\begin{bmatrix} \dot{\boldsymbol{e}} \\ \ddot{\boldsymbol{e}} \end{bmatrix} = \begin{bmatrix} -L_1 C & \mathbb{I}_{n\times n} \\ -\mathcal{L} - \Gamma - L_2 C & -\mathcal{L} - \Gamma \end{bmatrix} \begin{bmatrix} \boldsymbol{e} \\ \dot{\boldsymbol{e}} \end{bmatrix} \tag{3.14}$$

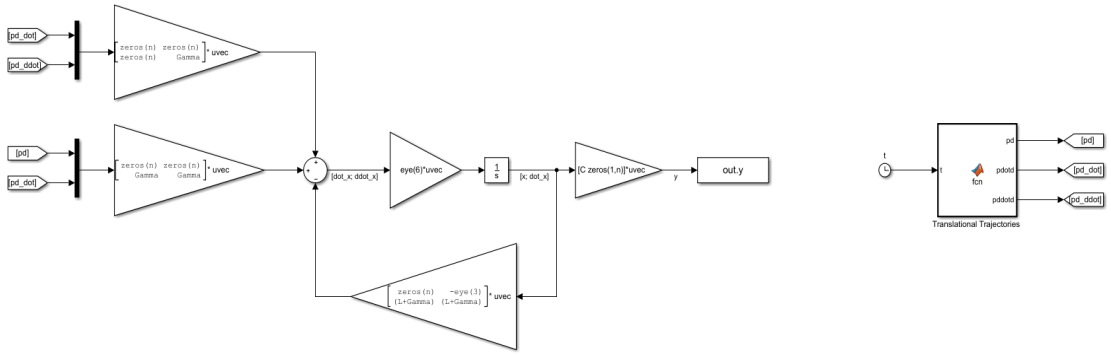


Figure 3.19: Simulink model implementing the dynamic consensus protocol

As before, from the desired eigenvalues is it possible to calculate the two L_1 and L_2 observer gains, starting form the calculation of the eigenvalues

$$
\begin{aligned}
det\left|\left[\begin{array}{cc} -L_1C & \mathbb{I}_{n\times n} \\ \mathcal{L}-\Gamma-L_2C & -\mathcal{L}-\Gamma \end{array}\right]-\left[\begin{array}{cc} \Lambda_{n\times n} & 0_{n\times n} \\ 0_{n\times n} & \Lambda_{n\times n} \end{array}\right]\right| = \\
det\left|\left[\begin{array}{cc} -L_1C-\Lambda_{n\times n} & \mathbb{I}_{n\times n} \\ \mathcal{L}-\Gamma-L_2C & -\mathcal{L}-\Gamma-\Lambda_{n\times n} \end{array}\right]\right| = \\
= \Lambda^2+\Lambda\left(L_1C+\mathcal{L}+\Gamma\right)+\left(L_1C\mathcal{L}+L_1C\Gamma+\mathcal{L}+\Gamma+L_2C\right)
\end{aligned}
$$

and imposing the equation above to be equal to

$$s^2+2\xi\omega_0 s+\omega_0^2$$

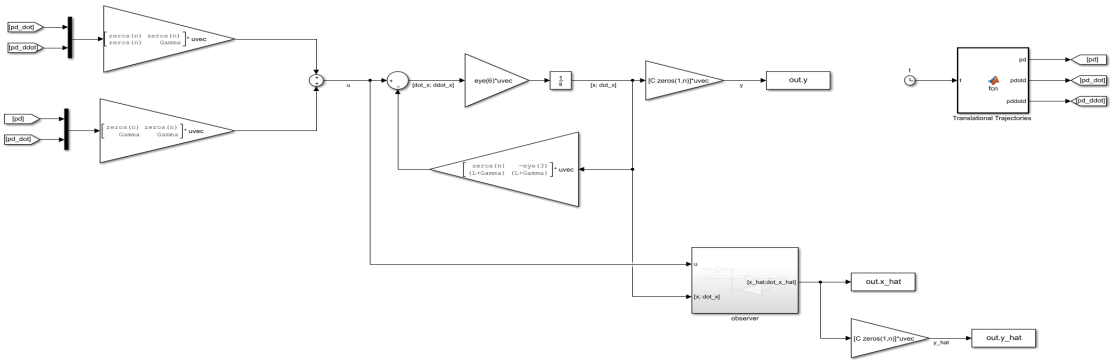
With $\xi=1$ and $\omega_0=\frac{5}{t_{r_{cons}}}=\frac{5}{2}=2.5$, the two gains result:

$$
\begin{cases} L_1C\mathcal{L}+L_1C\Gamma+\mathcal{L}+\Gamma+L_2C=\frac{25}{2.5^2} \\ L_1C+\mathcal{L}+\Gamma=\frac{10}{2.5} \end{cases}
$$

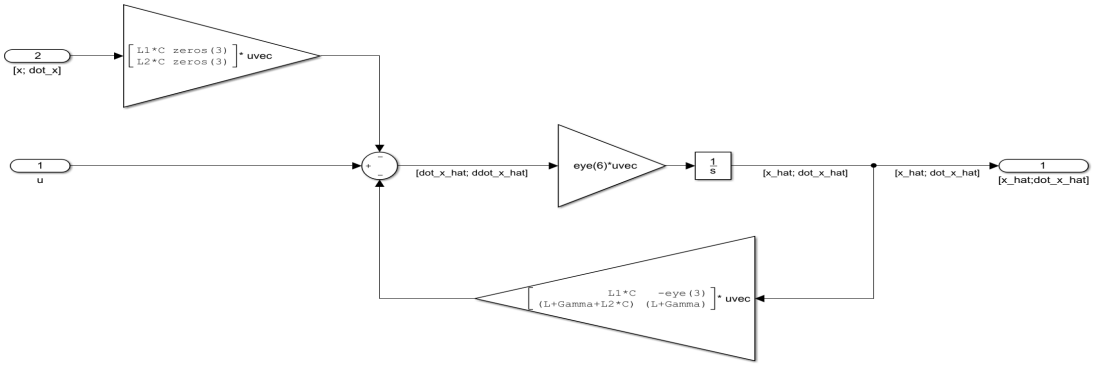
$$
\begin{cases} L_1=\left(\frac{10}{2.5}-\mathcal{L}-\Gamma\right)\cdot\frac{1}{C} \\ L_2=\left[\frac{25}{2.5^2}-L_1C\left(\mathcal{L}+\Gamma\right)-\mathcal{L}-\Gamma\right] \end{cases}
$$

For example, consider the VL formation is taken into account as it can be demonstrated to make the system both observable and controllable. The matrices characterizing the system are:

$$
\mathcal{L}=\left[\begin{array}{ccc} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{array}\right],\quad B=\left[\begin{array}{c} 1 \\ 0 \\ 0 \end{array}\right]\quad C=\left[\begin{array}{ccc} 1 & 0 & 0 \end{array}\right],\quad \Gamma=\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right]
$$



(a)



(b)

Figure 3.20: (a) Simulink model implementing the dynamic consensus protocol provided by an observer and (b) internal structure of the observer

The observer gain L results:

$$L = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 0 & -1.75 & -1 & 0 \end{bmatrix}'$$

In the research several observer models (e.g. Figure 3.20(a),3.20(b)) were simulated but none of them with satisfying results. So, next step will be to ultimate the model of an observer and to expand it by the addition of a PID controller.

Chapter 4

Experimental Stage

The experimental part was conducted on real quadrotors (Crazyflie 2.1x, manufactured by Bircraze) with fascinating properties, as the small frame, the light structure, the easiness of the assembly, the reusability and the good performances, given by the two on-board MCUs (accountable for handling the communication, the state estimation and the control) and even to the spiking of expansion decks (e.g. Flow Deck and Z-Ranger Deck) and an external motion capture (e.g. Optitrack). Despite the manufacturing ensures the download of Crazyflie Client application through which a single quadrotor can be piloted in a preliminary approach,. the real potentiality of this quadrotor lies, in an advanced development, in a open-source and fully programmable project, by using ROS (see Appendix C), that allows the customization of functionalities regarding our needs.
Crazyflie is, then, the ideal platform when dealing with research and education environment, especially about indoor flight in dense formations.
This chapter aims to retrace the steps followed to configure the quadrotor and its environment and to report the results obtained during the tests, taking inspiration from the work of Hönig [58],[59]. The chapter is outlined as it follows:

- Datasheet;
- Unpacking and Assembly;
- Crazyflie PC Client Interface Setup;
- Client Interface;
- Crazyflie Workspace Configuration;
- Expansion Decks;
- Experimental Tests.

4.1 Datasheet

The features and some specifications of Crazyflie 2.1 follows, as they are presented by the manufacturing in the datasheet.

Mechanical specifications:

- Takeoff weight: 27g;
- Size (WxHxD): 92x92x29mm (motor-to-motor and including motor mount feet).

Radio specifications:

- 2.4GHz ISM band radio;
- Increased range with 20 dBm radio amplifier, tested to $>$ 1 km range LOS with Crazyradio PA (environmentally dependent);
- Bluetooth Low Energy support with iOS and Android clients available;
- Dual antenna support with both on board chip antenna and U.FL connector.

Microcontrollers:
Crazyflie 2.1 relies on two microcontrollers, STM32 (the master) and NRF51 (the slave), that communicate thanks the *syslink protocol*,

- STM32F405 main application MCU (Cortex-M4, 168MHz, 192kb SRAM, 1Mb flash), it is used for state-estimation, control and handling of extensions:
  - Sensor reading and motor control;
  - Flight control;
  - Telemetry (including the battery voltage);
  - Additional user development.
- nRF51822 radio and power management MCU (Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash), that manages the flight control and everything else:
  - ON/OFF logic, supporting also the bootloading when the ON/OFF button is hold pressed for few seconds;
  - Enabling power to the rest of the system (STM32, sensors and expansion board);
  - Battery charging management and voltage measurement;
  - Master radio bootloader;

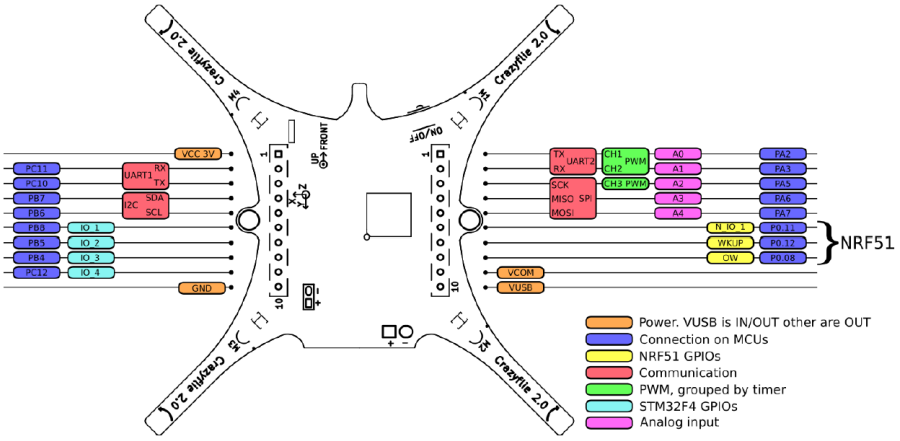


Figure 4.1: Expansion connector multiplexing

  - Radio and BLE communication;
  - Detect and check installed expansion boards.

IMU:

- 3 axis accelerometer / gyroscope (BMI088);
- high precision pressure sensor (BMP388).

Flight specifications:

- Flight time with stock battery: 7 minutes;
- Charging time with stock battery: 40 minutes;
- Max recommended payload weight: 15 g.

Supported clients/controllers:

- Win/Linux/OSX python client;
- The gamepads used by the Xbox 360 and the Playstation 3 are used as reference controllers, but any gamepad/controller with at least 4 analog axes can be used;
- Android mobile device;
- iOS mobile device.

4.2 Unpacking and assembly

The main advantage of Crazyflie corresponds to the easiness of assembly, as no soldering is required.
The following components are included in the package:

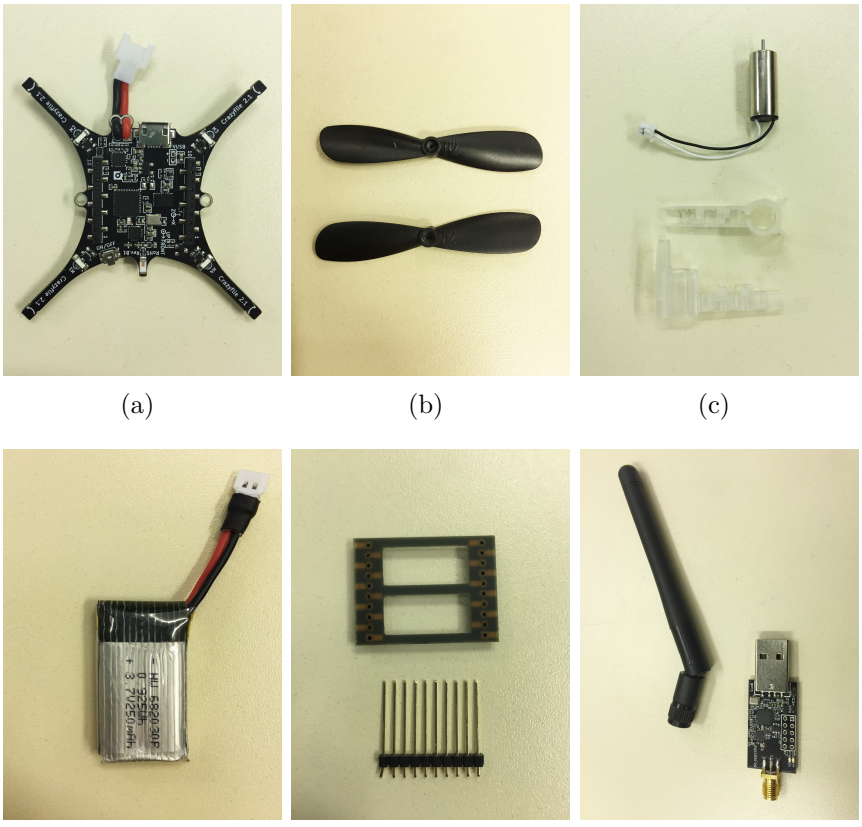
- 1 x Crazyflie 2.1 control board (Figure 4.2(a));
- 2 x CCW propellers (B) (Figure 4.2(b), above);
- 2 x CW propellers (A) (Figure 4.2(b), below);
- 4 x Coreless DC motors (Figure 4.2(c), above);
- 4 x Motor mounts (Figure 4.2(c), below);
- 1 x Foam battery pad;
- 1 x LiPo battery (240mAh) (Figure 4.2(d));
- 1 x Battery holder expansion board (Figure 4.2(e), above);
- 2 x Short/long expansion connector pins (2mm spacing, 8/14 mm long) (see Figure 4.2(e), below);
- 1 x USB cable;
- 1 x Crazyradio PA with USB dongle (Figure 4.2(f)).

Preliminary test

Before the assembly, a self-test should be performed to be sure that the control board correctly works: connect the Crazyflie board to a USB power source: if the M4 LED blinks green five time fast, the test is passed, insted, if M1 LED blinks red five time fast, the test is not passed.

Assembly

- Twist the wires of the four motors to better fit the motor's mounts and to reduce disturbances;
- Embed the motor and the wires in the motor's mounts (Figure 4.3(a));
- Insert the motor's mounts in the four branches of the control board and connect the motor connectors to the Crazyflie(Figure 4.3(b));

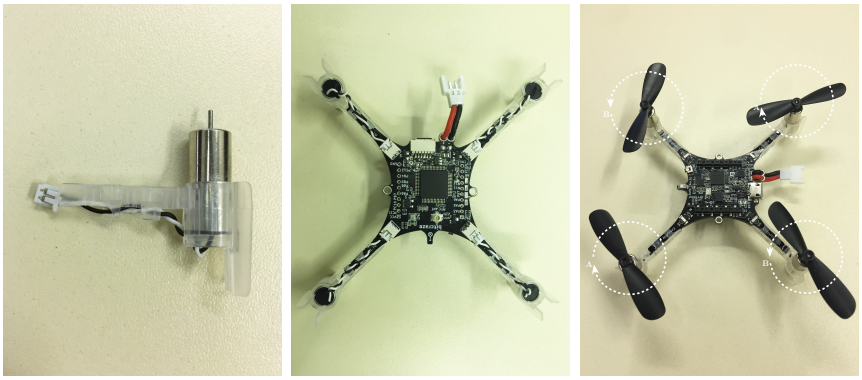


(a) (b) (c)

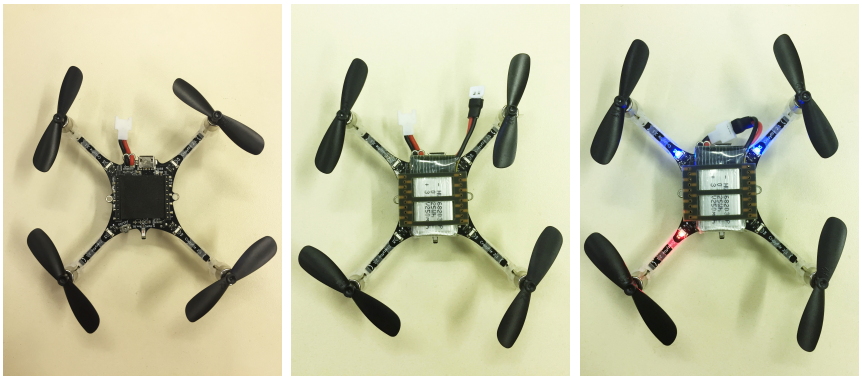
(d) (e) (f)

Figure 4.2: Components of Bitcraze Crazyflie 2.1

- Attach the propellers, taking account that two adjacent branches have not the same rotation direction. In fact, a propeller marked with an A1 or A2 has a clockwise (CW) sense of rotation, instead one marked with B1 or B2 has a counter clockwise (CCW) sense of rotation. In Figure 4.3(c) are represented the rotation directions for the propellers. To prevent the break of the propeller, avoid to push it until the point of contact with the motor;
- Attach the rubber pad on the top of the Crazyflie to keep better the battery in position (Figure 4.3(d));
- Complete the assembly by adding the headers in the expansion connectors and, after positioning the battery on the rubber, positioning the battery holder (Figure 4.3(e)).
- Power on: the Crazyflie is provided with a power-push button and it will automatically power on when connecting the battery (Figure 4.3(f)). Firstly a



(a) (b) (c)



(d) (e) (f)

Figure 4.3: Mounting steps

self-test will be run, to check if the hardware correctly works, and the sensor will be calibrated.

4.3 Crazyflie PC Client Interface Setup

As mentioned before, the Crazyflie PC Client Interface permits the Crazyflie flight through the utilization of a joystick. The steps required to install the application and a introduction to the major functionalities of the interface follow.
Assuming that the application can be easily installed in mobile devices or in Windows computers, the installation procedure is directed to the Linux operating system (in this case Ubuntu 18.04).
The Crazyflie PC Client Interface can be downloaded from source. As very first step is necessary download *cflib*, an API written in Python required for the communication between the client software and the quadcopter.

Install Python3, pip and pyqt5:

```
sudo apt-get install python3 python3-pip python3-pyqt5 python3-
    pyqt5.qtsvg
sudo pip3 install --user -e
```

Creation of a directory divided in two parts, the first one refers to $crazyflie - client - python$ ($cflib$):

```
mkdir ~/crazyflie
cd ~/crazyflie
git clone https://github.com/bitcraze/crazyflie-lib-python.git
cd crazyflie-lib-python
pip3 install --user -e
```

The second part consists in the $crazyflie - client - python$, that uses $cflib$ to generate the graphical user interface:

```
cd ~/crazyflie
git clone https://github.com/bitcraze/crazyflie-clients-python.
    git
cd crazyflie-clients-python
pip3 install --user -e .
```

On Linux, to use the Crazyradio, it is necessary to set udev permissions as follows:

```
sudo groupadd plugdev
sudo usermod -a -G plugdev $USER
sudo touch /etc/udev/rules.d/99-crazyradio.rules
sudo nano /etc/udev/rules.d/99-crazyradio.rules
```

with the last command line ($sudonano/etc/udev/rules.d/99 - crazyradio.rules$) it is possible to open the newly created $99 - crazyradio.rules$ and there write:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="1915", ATTRS{idProduct}==
    7777", MODE="0664", GROUP="plugdev"
```

and load the new rules, by writing in the command tab:

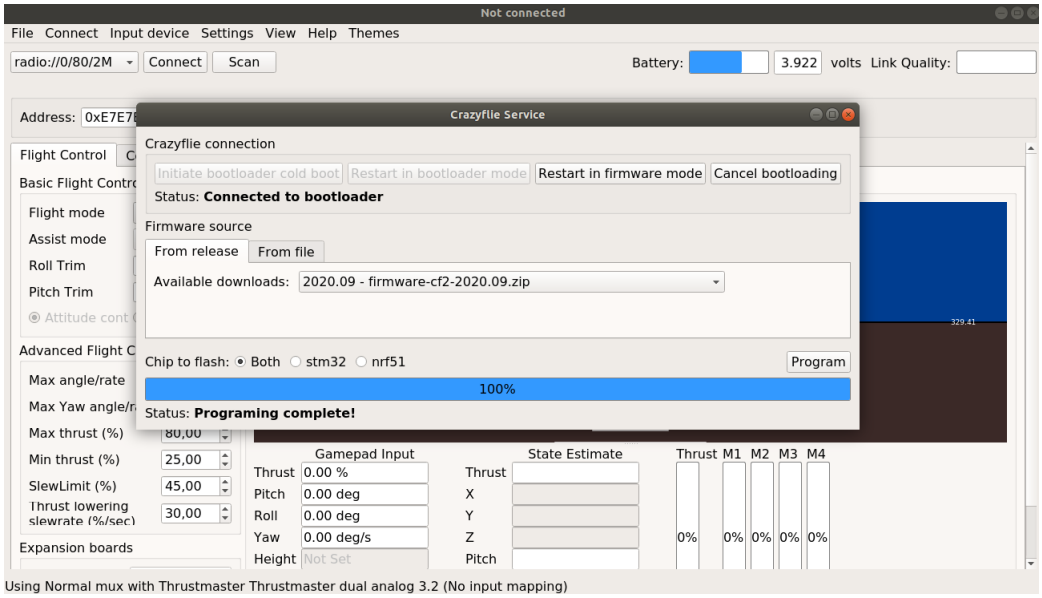


Figure 4.4: Firmware Upgrading

```
sudo udevadm control --reload-rules
sudo udevadm trigger
```

Now it is finally possible to start the Crazyflie PC Client with the following command lines:

```
cd ~/crazyflie/crazyflie-clients-python
sudo python3 bin/cfclient
```

4.3.1 Firmware upgrade

At the first use is crucial to update the firmware to the latest version (it can be downloaded going to: releases.
The update of the firmware can be done by following the listed instructions below:

- Select in the menu the voices *Connect→Bootloader*: a new window, Crazyflie Service, will be opened;
- Turn off the Crazyflie and maintain the power-on button pressed for few seconds, the M_2 and M_3 blue LEDs will start to blink, meaning that the quadrotor is in bootloader mode. If not already done, insert the CrazyRadio;
- Click on the button *Initiate bootloader cold boot*;

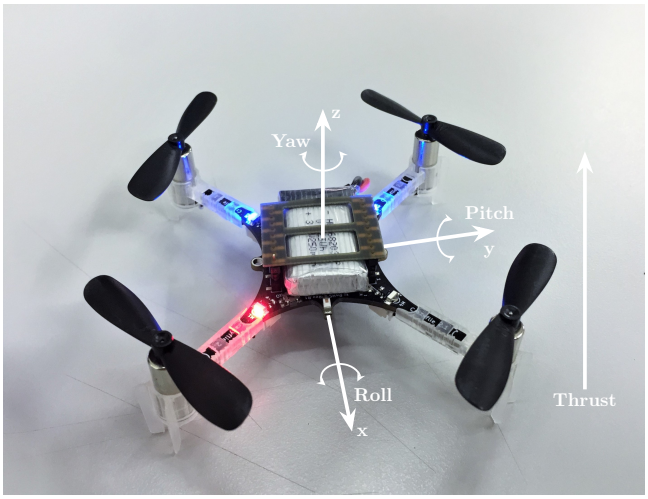


Figure 4.5: 3D coordinated system of Crazyflie

- Select the newest version between the *Available downloads* and, after assuring that *Both* option is selected, click on *Program*. In this case, both the MCUs, $STM32F405$ and $nRF51822$ will be upgraded;

- When the *Status* is completed (Figure 4.4), click on *Restart in firmware mode*.

The firmware version can be checked when the Crazyflie is connected, by looking at the output in the Console tab.

4.4 Crazyflie PC Client

Crazyflie PC Client is the client interface provided by the manufacturing, needed to control the Crazyflie, update the firmware, set parameters and log data.
To begin started to the Crazyflie's flight, it is crucial to introduce the concept of Thrust, the force that is applied to adjusts the altitude, and to define the quadrotor principal axes in a three-dimensional (3D) (Figure 4.5). Once the three axes are defined, the rotations around them are:

- Roll: rotation around x-axis. This permits the Crazyflie to move to its left or to its right.

- Pitch: the rotation around y-axis. This moves the Crazyflie forwards or backwards.

- Yaw: rotation around z-axis. Yaw is used to change flying direction.

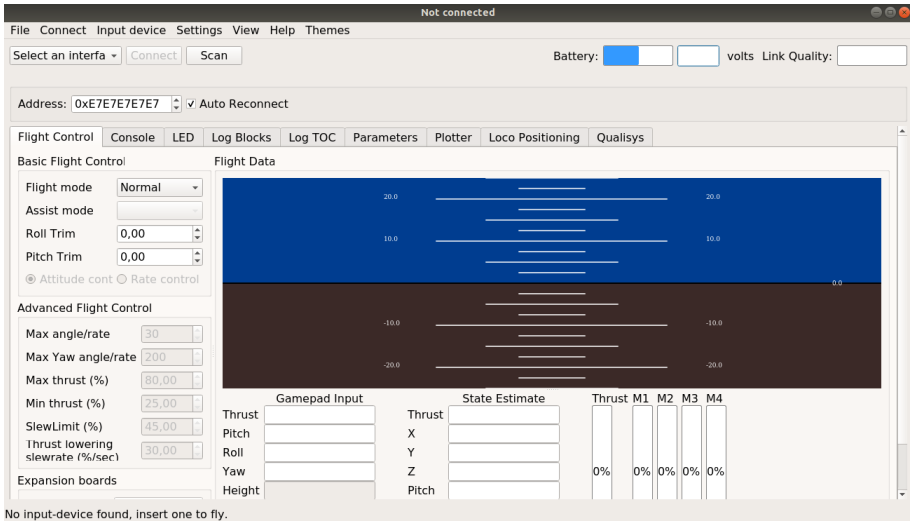


Figure 4.6: Crazyflie PC Client Interface

These four data will be used in the control of the Crazyflie flight.
Thanks to the PC Client Interface (Figure 4.6), it is possible to connect the Crazyflie and display Roll, Pitch, Yaw and Thrust, in the *Flight Control* tab.

Connection and input-device selection

At the first use, the Crazyflie have, by default, the address 0xE7E7E7E7E7E7, and, by clicking on the button *Scan*, it will be possible to *Connect* when it will be detected by the radio. The connection status can be seen above in the interface, and, so, the battery status.
In the menu, *Input device* allows to choose between different operating modes:

- *Normal*: normal mode, in which one single controller manages a quadrotor;
- *Teacher (RP)*: two joysticks, one (teacher) that manages roll and pitch, the second (student) controls all the others functionalities (but roll and pitch as well, thanks to the Mux-switch option);
- *Teacher (RPYT)*: two joysticks, one (teacher) controls roll, pitch, yaw and thrust, the other (student) is aimed at managing the other functionalities (but also roll, pitch, yaw and thrust thanks to Mux-switch option).

In easiest case, the *Normal* mode is used and the joystick is selecting, by clicking on *Input device*→*Device*→*Input map*→"your-device" (see in Figure 4.7).

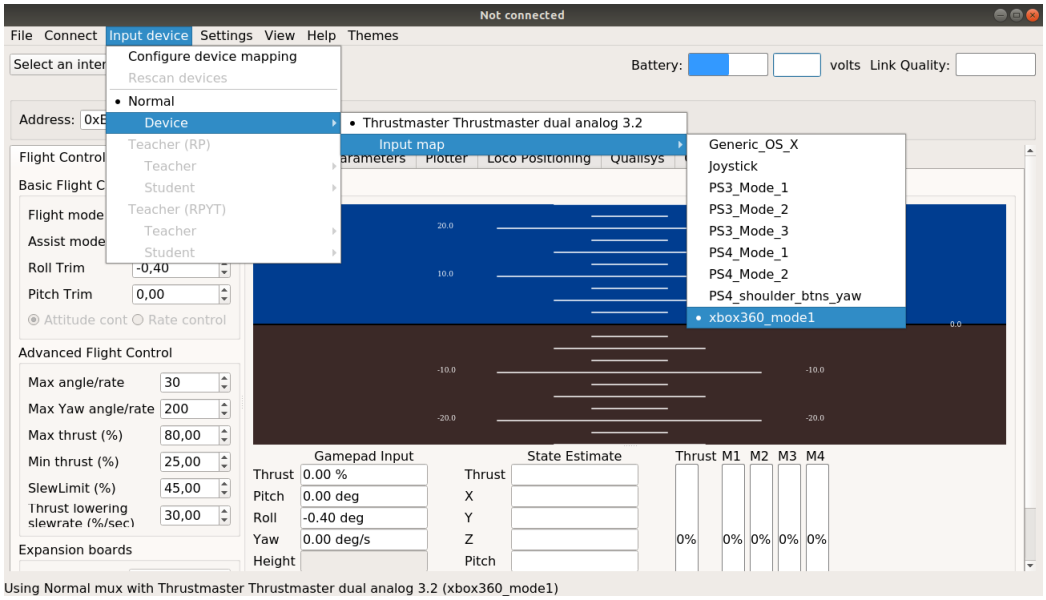


Figure 4.7: Selection of the Input Device

Configuration of Crazyflie

Once the Crazyflie is connected, the address can be easily changed: in the menu select *Connect→Configure 2.X*, it will be possible to set a new address, to select the radio bandwidth (250k, 1M or 2M) or the radio channel (from 0 to 125), pitch and roll trims. To save the modifications, click on the *Write* button and restart the quadrotor.

Configuration of joystick

Any type of input device, with four analog axis (roll, pitch, yaw and thrust), can be used, but it will referred to a joystick since it was used in the experiments.
The default configuration in the client interface can be changed, by going to *Input device → Configure device mapping* in the menu. By selecting the input device to be configured, the axis can be configured through the *Detect* button. Only for roll, pitch, yaw and thrust detections, it will compare a window in which it is possible to choose the *Combined Axis Detection* that permits to map the functionality to two axis.

Tabs

The principal interface displays many tabs that can be shown or hidden going to *View → Tabs*. An overview on the tabs function follows:

- Flight Control: thrust, pitch, roll and yaw joystick inputs control, estimation of the actual inputs, actual motors' output, basic and advanced flight controls;
- Log TOC: real-time log variables list. A log configuration can be created and saved in the *Log configuration* window, by clicking on *Settings→Logging configurations*;
- Parameters: detailed list of parameters, that are real-time variables stored in the Crazyflie. The variables are aggregated in groups and the name, the type and the accessability are specified for each variable (RW: data can be read and written from the client, RO: data can be only read);
- Log Blocks: list of all log configurations saved, start/stop tracking and write to file choices;
- Plotter: plot of the selected log configuration, it is possible to specify the number of samples (x-axis) and to settle the scale of the y-axis;
- Console: Firmware's output while the Crazyflie is in operation;
- LED tab: LED Ring expansion deck screening and control;
- Loco Positioning Tab: information from Loco Positioning System, see details in Sec. 4.6.2;
- Qualisys Tab: information from Qualisys motion capture system, based on high-resolution cameras that are able to detect markers positioned on Crazyflie's body.

4.5 Crazyflie Workspace

Assuming that the operating system used is Ubuntu 18.04.5 LTS with ROS Melodic Morenia already installed on it (if it is not, see the Appendix B and C for an introduction to ROS), the installation of the official software packages of the Crazyflie can begin.
The *crazyflie_ros* stack allows to publish ROS standard messages on on-board sensors, to use multiple Crazyflie with a single Crazyradio or to implement an external motion capture system.
The first step, to let the Crazyflie operate in the ROS environment, consists in the creation of a new ROS workspace and to clone all the crazyflie's packages:

```
1 mkdir -p ~/crazyflie_ws/src
2 cd ~/crazyflie_ws/src
```

```
catkin_init_workspace
git clone https://github.com/whoenig/crazyflie_ros.git
cd ~/crazyflie_ros
git submodule init
git submodule update
cd ~/crazyflie_ws
catkin_make
```

To use the new workspace, open your ∼/.bashrc with the following command:

```
sudo nano ~/.bashrc
```

and add at the end of the file the following line:

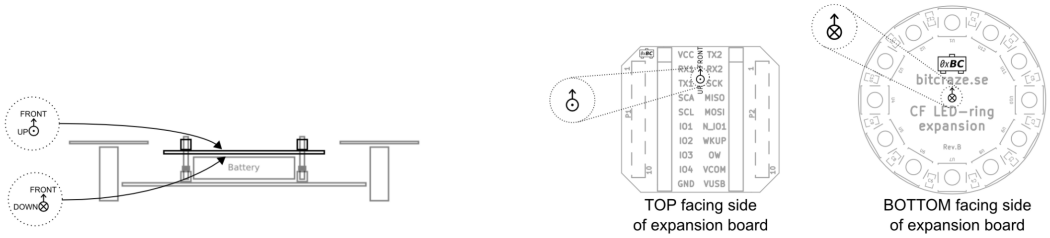
```
source ~/crazyflie_ws/devel/setup.bash
```

Instead, if you are already in the

```
source devel/setup.bash
```

At this point, the Crazyflie's workspace has been created with all its packages, that are:

- crazyflie_controller: it contains the code of a PID controller to let the quadrotors fly in a space provided by an external location feedback (e.g. VICON);
- crazyflie_cpp: this package is made by a C++ library for the Crazyradio and Crazyflie. It is used by crazyflie_ros and crazyflie_tools but it can be utilized independently of ROS too;
- crazyflie_demo: it contains many examples of .launch files (teleoperation, hovering, commanded position) and nodes in the Python format;
- crazyflie_description: it is composed by 3D-models of the Crazyflie in URDF format, inteded to be used in Rviz representation;
- crazyflie_driver: the driver is constituted by crazyflie_server, that allows to support multiple Crazyflies with a single Crazyradio, and by crazyflie_add, needed to add the quadrotors and let them communicate with the server;
- crazyflie_tools: it contains advanced operation tools.



(a) (b)

Figure 4.8: Expansion decks positioning

4.6 Expansion decks

Crazyflie quadrotor is provided by additional expansion decks that are useful to implement a reliable and predictable flight. In particular, the expansion decks actually available are:

- LED-ring expansion deck;
- Buzzer expansion deck;
- Qi inductive charging expansion deck;
- Qi 1.2 compatible wireless charging deck;
- Prototype expansion deck;
- Breakout expansion deck;
- BigQuad expansion deck;
- Micro SD expansion deck;
- Z-ranger expansion deck;
- Bosch Sensortec expansion deck;
- Lighthouse positioning deck;
- Motion capture marker deck.

Before entering in details of the expansions used in the experimental work, the decks' mechanical positioning will be introduced. Expansion boards can be installed on top, bottom, or both top and bottom of the Crazyflie, thanks to the female pass-through connectors that can be fitted with male pins.
Referring to Figure 4.8, the symbols indicate the bottom or the upper face and the orientation with which the expansion board has to be mounted.

4.6.1 FlowDeck v2

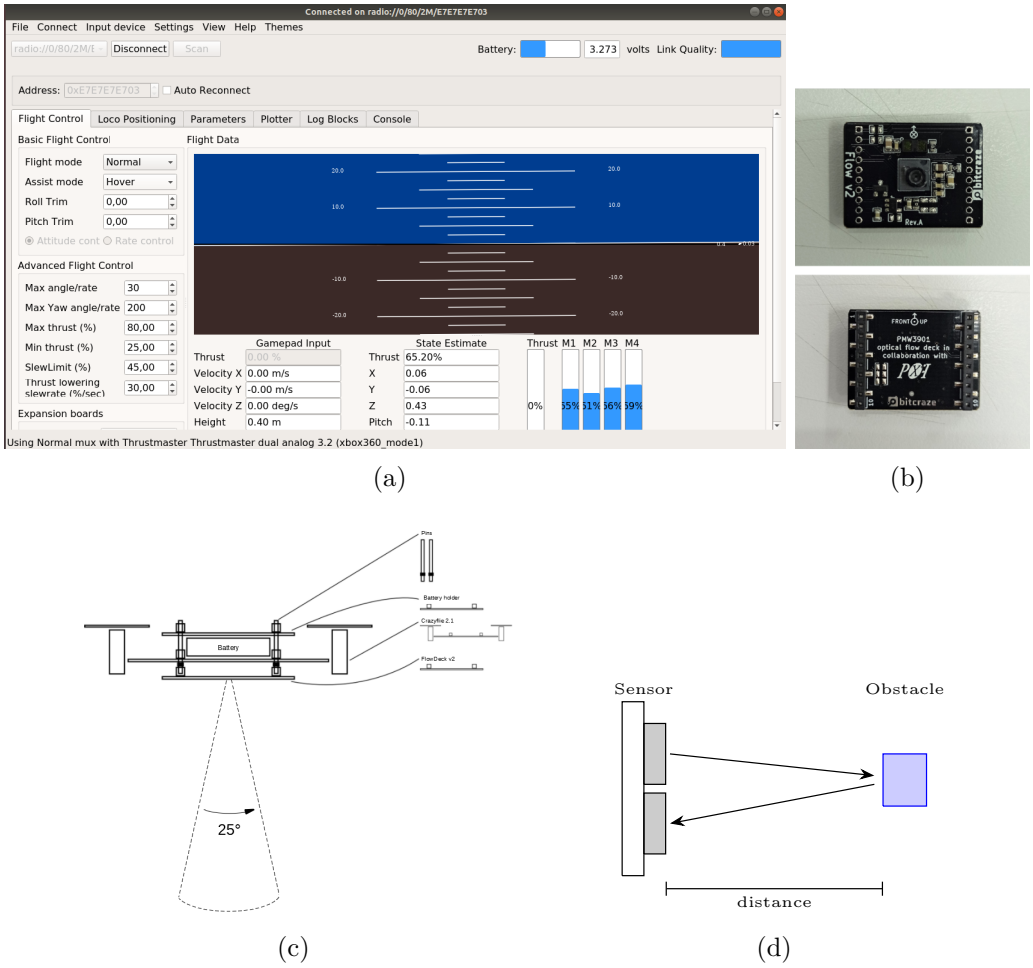
The Flow deck (Figure 4.9) makes possible to position the Crazyflie thanks to a optical motion detection.
The joint action of $VL35L1x$ and $PMW3901$ sensors allows the Crazyflie to detect the horizontal motion and to accurately estimate distance ranging until a maximum of 4 meters. Moreover, in the normal operation with the CF Client Interface, it's possible to let the quadrotor fly in assist mode: by selecting the *Hover* mode between the Assist modes, in the *Flight Control* tab (see Figure 4.9(a)), configuring the assisted control button on the joystick and holding the last mentioned during the flight, the Crazyflie will take off and hover at a previously fixed height (40cm by default).
In particular:

- VL53L1x ToF sensor, to measure distance from the ground with a precision of few millimeters, basing on the time for emitted light to be reflected (Figure 4.9(d)). The precision depends on surface and light conditions.
- PMW3901 optical flow sensor (Figure 4.9(c)), it measures movements with respect to the ground (works best on matt surfaces). Thanks to a camera, the relative motion between the quadrotor and the scene is sensed, meaning that a background full of details could improve the efficiency of the optical flow sensor.

4.6.2 Loco Positioning system

The Crazyflie's positioning in space given by the on board sensors (gyroscope and accelerometers) is not enough to obtain an accurate positioning. The joint utilization of Flow Deck v2 (Sec.4.6.1) and of an external positioning system could increase the accuracy.
The Loco Positioning system (LPS) is similar to a GPS system, able to calculate the current position of the quadrotor: a set of *Anchors* positioned in the room (reference system) receives short high frequency radio signals from the *Tags*, positioned directly on the Crazyflies. By measuring the latency between the moment in which the message is sent and the moment in which the message is received, the system is able to perform on-board calculation of the Crazyflies' position.
The Loco Positioning system is composed, then, by:

- Loco Positioning Deck(Figure 4.10(a),4.10(c)): it is mounted on the Crazyflie and it takes care of the calculation of the distance to the Anchors;
- Loco Position Node (Figure 4.10(b),4.10(d)): it can act either as an Anchor or Tag: in the first case, it would be part of the reference system, in the second case it is mounted on the quadrotor.



(a) (b)

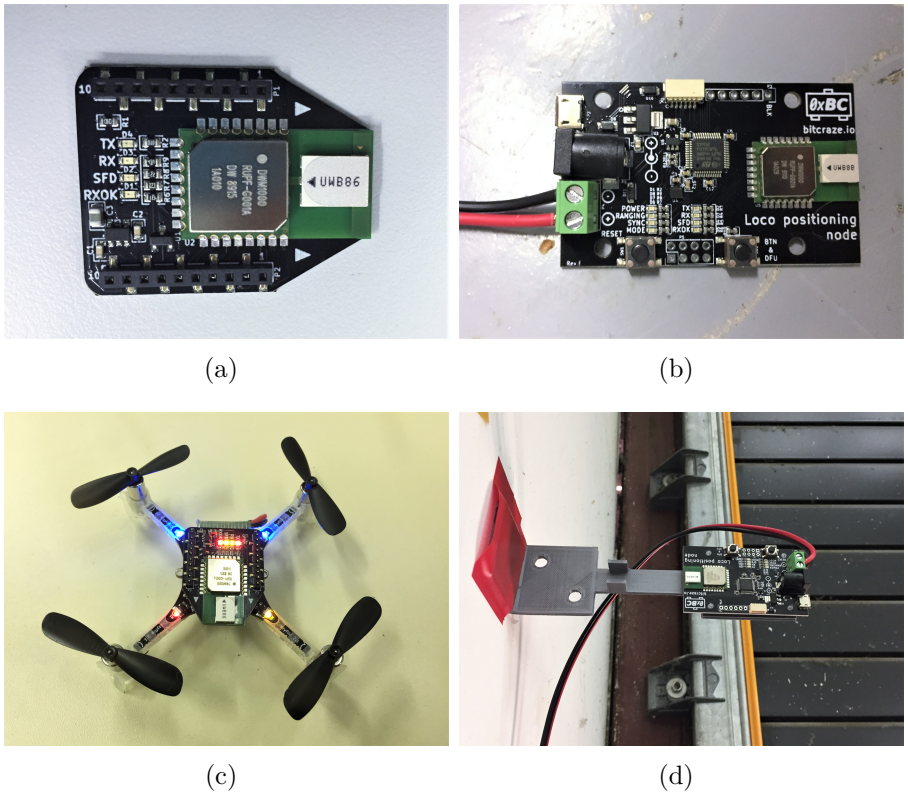
(c) (d)

Figure 4.9: a) Hover mode in Client Interface, b) Flow Deck v2, c) Crazyflie with the Flow Deck graphic, d) ToF (Time of Flight) sensor behavior

The system's ranging accuracy is around ±10cm and the maximum tested range corresponds to 10m.
The Loco Positioning system can operate in three different modes:

- *Two Way Ranging (TWR)*: the tags send messages to the anchors in sequence. TWR corresponds to the most accurate mode but only on tag can be utilized, with a maximum of 8 anchors, and it works also when the tag exit the space delimited by the anchors;

- *Time Difference of Arrival 2 (TDoA2)*: the anchors send messages continuously, the tag calculate the relative distance between the anchors from the delay between the messages. In this mode a lot of Crazyflies can be used, since



(a) (b)

(c) (d)

Figure 4.10: a) LPS deck, b) LPS node, c) Crazyflie with LPS deck, d) LPS node positioning on the room

the tag is listening to the messages and, so, the system doesn't risk to be overloaded. In TDoA2 it is recommended that the Crazyflie doesn't leave the space sorrounded by the anchors (maximum 8) to avoid information lost;

- *Time Difference of Arrival 3 (TDoA3)*:the anchors send messages in a randomized fashion, supporting the addition of more anchors, such that a larger operating space can be considered.

Loco Positioning system setting

Assuming that the Crazyflie is updated to the latest firmware and by updating the firmware and configuring each node, it's possible to start with the set up of Loco Positioning system.
Firstly, to update nodes, download the LPS configuration tool, in a new terminal:

```
cd ~/crazyflie_ws/src
git clone https://github.com/bitcraze/lps-tools.git
cd ~/crazyflie_ws
catkin_make
```

To flash the node, it's required to set the udev permissions as follows:

```
sudo touch /etc/udev/rules.d/99-lps.rules
sudo nano /etc/udev/rules.d/99-lps.rules
```

Once the the $99 - lps.rules$ is open, write:

```
# cat > /etc/udev/rules.d/99-lps.rules << EOF
SUBSYSTEM=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="
    df11", MODE="0664", GROUP="plugdev"
EOF
```

and load the new rules, by writing in the command tab:

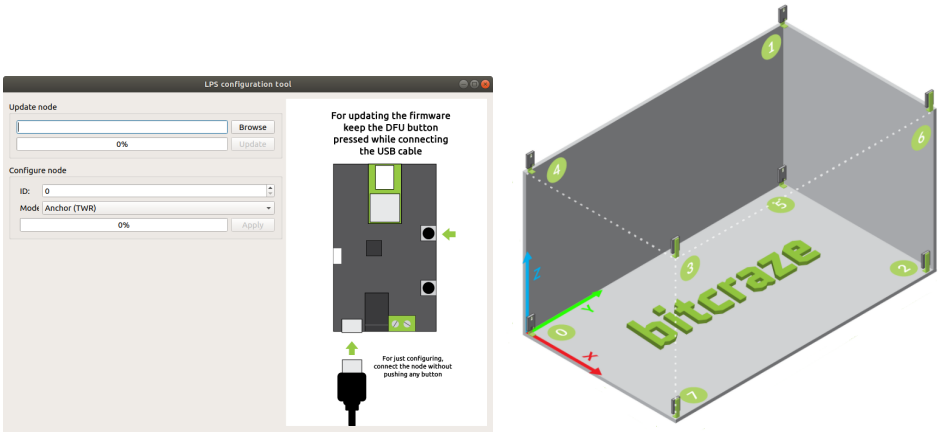
```
sudo udevadm control --reload-rules %reload
sudo udevadm trigger
```

Now it's possible to open the LPS configuration tool, starting from home, as:

```
cd crazyflie_ws/src/lps-tools/
python3 -m lpstools
```

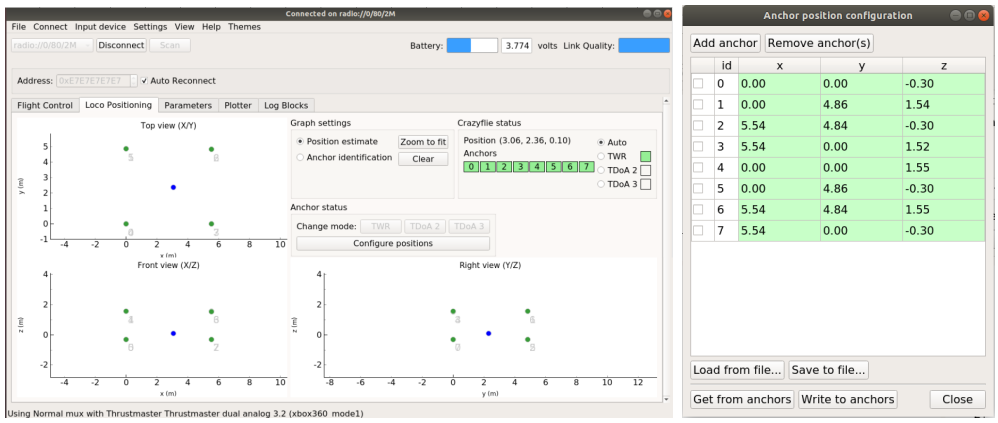
Once the LPS configuration tool is opened, keep the node's DFU button hold while connecting the node to the computer via USB (in zFigure 4.11(a) the LPS configuration tool interface and the instruction to enter in DFU mode), and execute the following instructions (in TWR mode, the one used during the experimental work):

- Browse the last firmware, previously downloaded here: firmware;
- Update the node;
- disconnect the node, press the reset button (the other button on the node) and reconnect it;
- Select the ID (all the nodes have to be numbered with a different ID, it will be applied to correctly position the nodes in the room);



(a) (b)

Figure 4.11: a) LPS configuration tool interface, b) Loco Position Nodes arrangement in a room



(a) (b)

Figure 4.12: a) Loco Positioning Tab b) Anchors' actual positioning

- make sure that the *Anchor (TWR)* mode is selected;
- click on Apply button.

Now the anchors are ready to be positioned in the room.

Anchor positioning and power-on

In the flying arena, 8 anchors were utilized, so only that setting case will be considered.

To get an acceptable result, the anchors should be distributed in the room at least 2m apart and placed 15cm far from the wall, ceiling, metal object, to avoid any reflection interference (make use of the following 3D-stamped support: anchor-stand).
Once the anchors were disposed as displayed in Figure 4.11(b), they can be powered on in three different ways: through micro USB, barrel jack or screw terminal, with 5-12V and providing, at least, a current of 150mA.

LPS Tab

By connecting the Crazyflie through the CF client, it is possible to visualize the Loco Positioning System operation in the LPS tab.
In the tab, firstly check the anchor status, by looking at the color of the boxes: if one the boxes is red, the corresponding anchor is not communicating with the quadrotor. It follows, if not already done, the configuration of the anchor positions. A new window will be opened and the actual anchor positions will be stored (Figure 4.12(b)). In the Loco Positioning tab (Figure 4.12(a) are shown three $2D$-graphs, that are able to give a $3D$ idea of the positioning:

- Top view (X/Y);
- Front view (X/Z);
- Right view (Y/Z).

4.6.3 Optitrack

Optitrack is a motion capture system, capable to offer high-quality performances and affordable prices. It is based on a set of cameras, positioned in the corners of the flying-area and a set of markers that will be arranged on the rigid body to be tracked.
Starting from the hardware setup, it is essential to guarantee a clean and free-reflections area, in order to avoid to spoil the tracking. Therefore, it is suggested to cover every sunlight source and to cover with rubber the reflective areas.
Optitrack uses the Motive software platform to control the motion capture system and to track the bodies and process the recorded 3D data, together with the live-stream of the data.
At the first usage of Motive, the client interface of Optitrack, it is asked to upload a calibration layout, a file with the information to completely restore positions and orientations of each camera, lens distortion parameters and the camera settings. It is supposed that the calibration is already provided and it uploaded by clicking on *File*→*Open* and browse the file.
After this passage, the interface will be as the one in Figure 4.13, in which the four

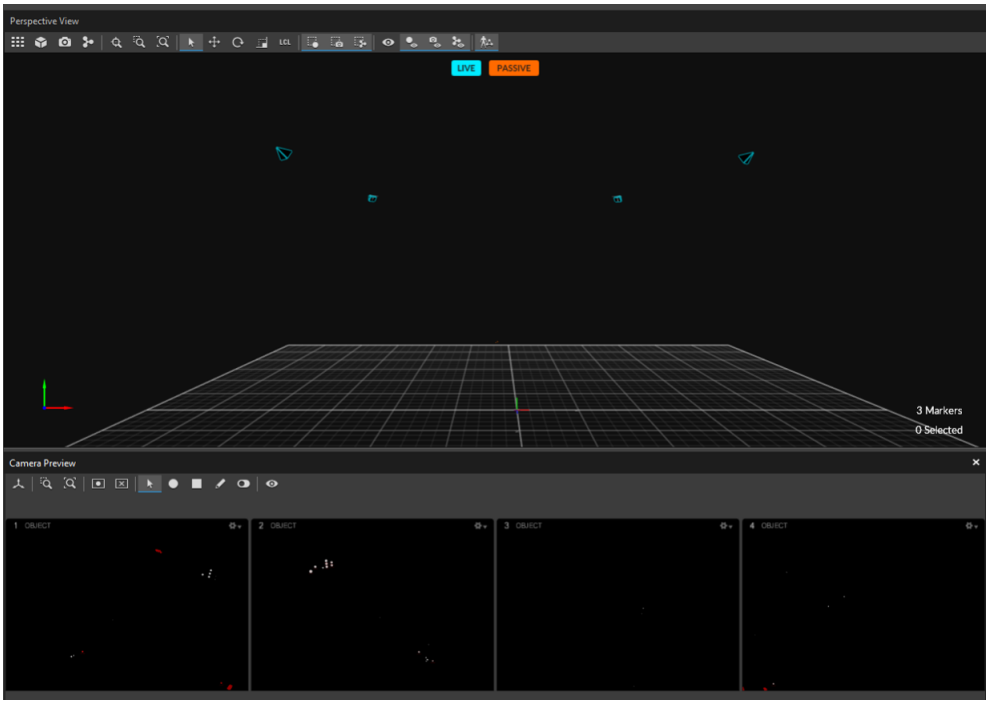
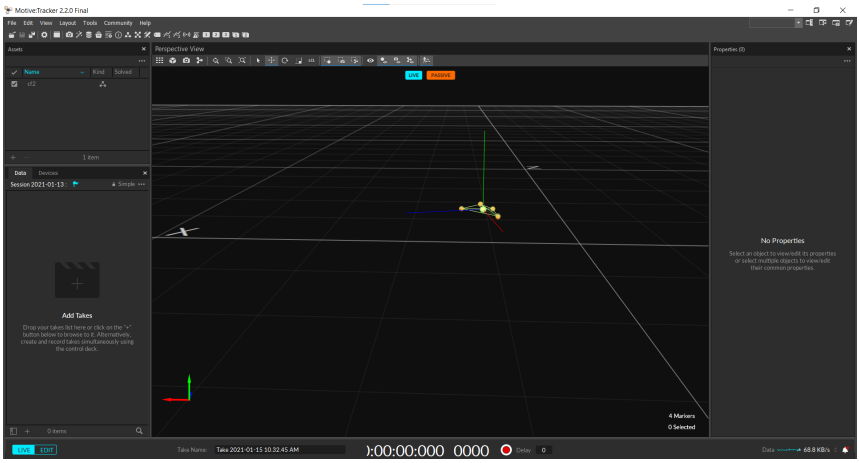


Figure 4.13: Optitrack Interface

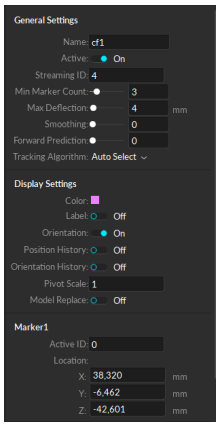
cameras are figured in the exact position in a 3D viewport but also the view from each camera in a 2D viewport (bottom part in Figure 4.13). Now, it follows the identification of the body, performed in few steps:

- place four or more markes on the body in a asymmetrical fashion (Figure 4.14(b));
- Position the mobile robot in the Optitrack's observable area;
- the markes should be identified and figured in the *Perspective view*. At this point, select the markers by clicking and dragging the mouse over all the markers;
- right click on one of the markers and select *Rigid Body→Create from selected markers*;
- modify the name of the newly created rigid body (Figure 4.14(a)) in the *Properties Pane* (see Figure 4.14(c),4.14(d));
- verify the tracking of the rigid body by moving your body in the Optitrack's observable area and looking at the interface.



(a)

(b)



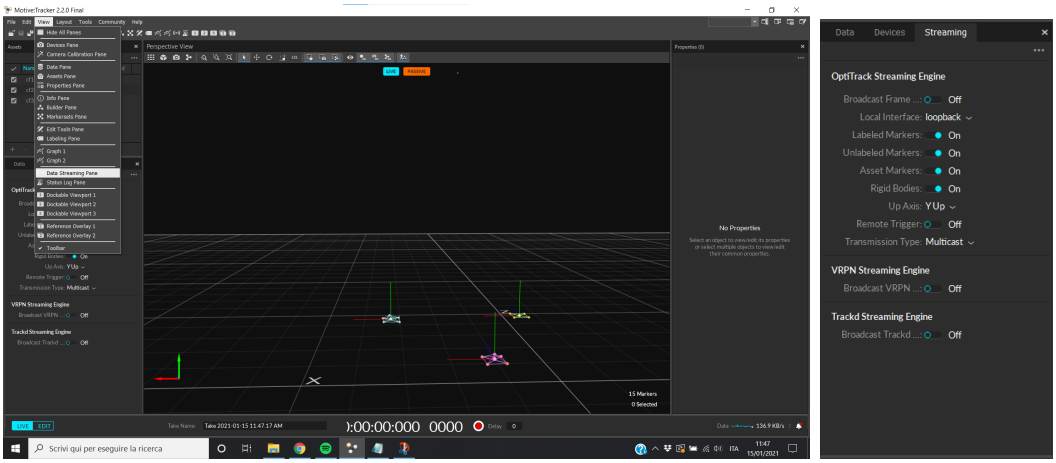
(c)

(d)

Figure 4.14: (a) Perspective view, (b) markers placement on the Crazyflie and (c),(d) Properties pane

After the definition of all the rigid bodies in the area, the capture can start, by pressing the red record button. If the red button is pressed again, the capture will be ended and a capture file (TAK extension) will saved, a file containing all the data to reconstruct the capture.
In the case of Crazyflie controlled thanks to ROS, the Data should be streamed in real-time: Motive offers several options, in this case the VRPN mode was deployed. At the beginning connect the two laptops (one deployed for Motive, the second for ROS) to the same network, used to stream the data, and follow the steps:

- click on *View→Data Streaming Pane* (Figure 4.15(a));
- set everything as shown in Figure 4.15(b):



(a) (b)

Figure 4.15: Streaming of the tracking Data

- make sure that the *Broadcast Frame Data* and the *VRPN Data* are on ON;
- Set the *Local Interface* to the correct IP address, corresponding to the one used to broadcast the information;
- Set *Up Axis* to Z Up;
- select the VPRN Broadcast.

On the laptop on which ROS is running, it is necessary to install the vrpn-client-ros package:

```
sudo apt install ros-melodic-vprn-client-ros
```

and import the *vrpn.launch* file in your *Launch* folder:

```
<?xml version="1.0" ?>
<launch>

    <arg name="server" default="192.168.0.138"/>

    <node pkg="vrpn_client_ros" type="vrpn_client_node" name="
    vrpn_client_node" output="screen">
        <rosparam subst_value="true">
            server: $(arg server)$
            port: 3883
```

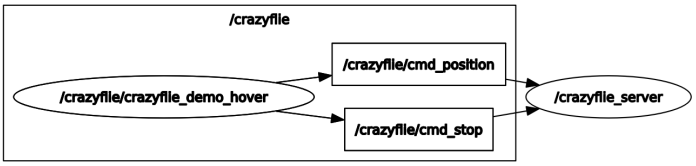


Figure 4.16: rqt graph representing the mod_position.launch file

```
            update_frequency:100.0
            frame_id: world

            # Use the VRPN server's time, or the client's ROS time.
            use_server_time: false
            broadcast_tf: true

            # Must either specify refresh frequency > 0.0, or a list
    of trackers to create
            refresh_trakcer_frequency: 1.0

            trackers:
                - cf1

        </rosparam>
    </node>

</launch>
```

In the script can be found the same parameters that were fixed on Motive: the Local Interface, the VRPN Broadcast Port and the name of the trackers. When the *.launch* file is launched, the connections between nodes as showed the Figure 4.16 are created. At this point everything is set to correctly track the motion of the Crazyflie.

4.7 Experimental Tests

In the following section there are reported few examples of experimental tests conducted on one or more Crazyflies, giving a general idea of the tasks carried out, how to replicate them, and delineating the steps to follow in order to create a new workspace and new nodes.

4.7.1 Position holding - Hovering

In the first easiest tests, the scripts already provided in the Crazyflie Workspace were modified and applied. First task was to perform, in order, the take off, the hovering and the land of a Crazyflie. It follows the *position.launch* file, the related node in Python code script and a brief explanation:

```
## position.launch
<?xml version="1.0"?>
<launch>
  <arg name="uri" default="radio://0/80/2M/E7E7E7E703" />

  <include file="$(find crazyflie_driver)/launch/crazyflie_server.
    launch">
  </include>

  <group ns="crazyflie">
    <include file="$(find crazyflie_driver)/launch/crazyflie_add.
    launch">
        <arg name="uri" value="$(arg uri)" />
        <arg name="tf_prefix" value="crazyflie" />
        <arg name="enable_logging_imu" value="True" />
        <arg name="enable_logging_pose" value="True" />
    </include>

    <node name="crazyflie_demo_hover" pkg="crazyflie_demo" type="
    mod_Position.py" output="screen">
    </node>

    <node name="plotjuggler_with_layout2"
          pkg="plotjuggler"
          type="PlotJuggler"
          args="--layout $(find crazyflie_controller)/config/
    z_position.xml"
    />
    </group>
</launch>
```

The launch code is the one that will activate the nodes (the Python code, the related graphic tool to plot the data), that creates the connections (Figure 4.16) between the Crazyflie and the server by reading the appropriate URI, and enables the logging of the data (in this case IMU and pose).
In detail the IMU comprises the angular velocity $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ coming from the gyroscope, the angular velocity covariance, the linear acceleration $(\ddot{x}, \ddot{y}, \ddot{z})$, linear acceleration covariance, orientation (w, x, y, z) and orientation covariance. Instead, the pose includes: the orientation (ϕ, θ, ψ) and position (x, y, z).

```
## mod_Position.py
#!/usr/bin/env python

import rospy
import tf
from crazyflie_driver.msg import Position
from std_msgs.msg import Empty
from crazyflie_driver.srv import UpdateParams

if __name__ == '__main__':
    rospy.init_node('position', anonymous=True)
    worldFrame = rospy.get_param("~worldFrame", "/world")
    rate = rospy.Rate(10) # 10 hz
    name = "cmd_position"
    msg = Position()
    msg.header.seq = 0
    msg.header.stamp = rospy.Time.now()
    msg.header.frame_id = worldFrame
    msg.x = 0.0
    msg.y = 0.0
    msg.z = 0.0
    msg.yaw = 0.0

    pub = rospy.Publisher(name, Position, queue_size=1)
    stop_pub = rospy.Publisher("cmd_stop", Empty, queue_size=1)
    stop_msg = Empty()

    rospy.wait_for_service('update_params')
    rospy.loginfo("found update_params service")
    update_params = rospy.ServiceProxy('update_params', UpdateParams)

    rospy.set_param("kalman/resetEstimation", 1)
    update_params(["kalman/resetEstimation"])
    rospy.sleep(0.1)
    rospy.set_param("kalman/resetEstimation", 0)
    update_params(["kalman/resetEstimation"])
    rospy.sleep(0.5)

    # take off
    while not rospy.is_shutdown():
        for y in range(10):
            msg.x = 0.0
            msg.y = 0.0
            msg.yaw = 0.0
            msg.z = y / 25.0
            now = rospy.get_time()
            msg.header.seq += 1
            msg.header.stamp = rospy.Time.now()
```

```
            rospy.loginfo("sending...")
            rospy.loginfo(msg.x)
            rospy.loginfo(msg.y)
            rospy.loginfo(msg.z)
            rospy.loginfo(msg.yaw)
            # rospy.loginfo(now)
            pub.publish(msg)
            rate.sleep()
        for y in range(350):
            msg.x = 0.0
            msg.y = 0.0
            msg.yaw = 0.0
            msg.z = 0.4
            msg.header.seq += 1
            msg.header.stamp = rospy.Time.now()
            rospy.loginfo("sending...")
            rospy.loginfo(msg.x)
            rospy.loginfo(msg.y)
            rospy.loginfo(msg.z)
            rospy.loginfo(msg.yaw)
            # rospy.loginfo(now)
            pub.publish(msg)
            rate.sleep()
        break

    # land, spend 1 secs
    start = rospy.get_time()
    while not rospy.is_shutdown():
        msg.x = 0.0
        msg.y = 0.0
        msg.z = 0.0
        msg.yaw = 0.0
        now = rospy.get_time()
        if (now - start > 1.0):
            break
        msg.header.seq += 1
        msg.header.stamp = rospy.Time.now()
        rospy.loginfo("sending...")
        rospy.loginfo(msg.x)
        rospy.loginfo(msg.y)
        rospy.loginfo(msg.z)
        rospy.loginfo(msg.yaw)
        pub.publish(msg)
        rate.sleep()

    stop_pub.publish(stop_msg)
```

In the mod_Position.py script is possible to find all the commands relative to the take off and the land of the quadrotor.

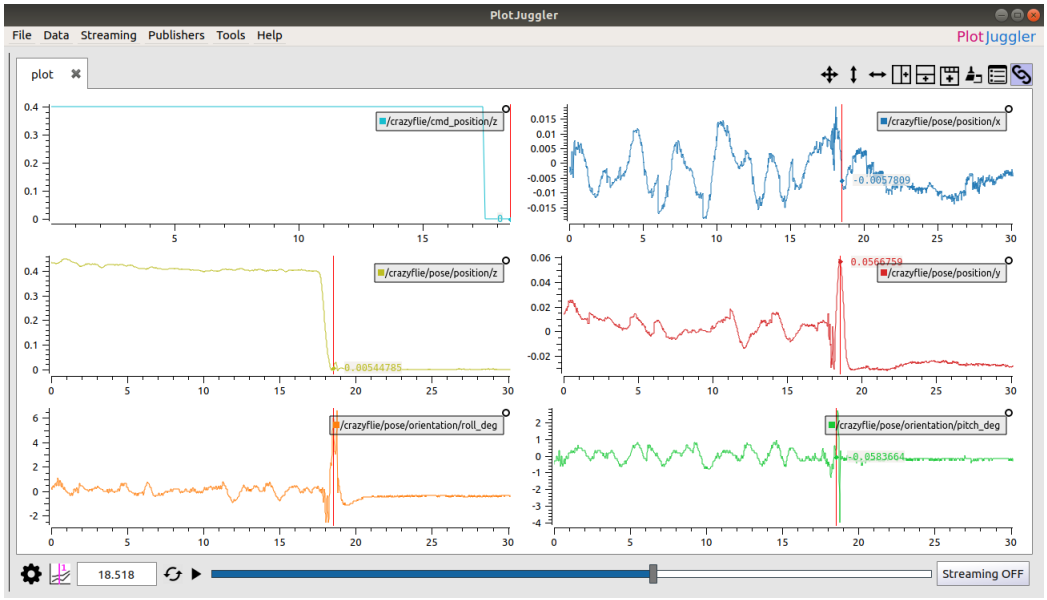


Figure 4.17: Plot of Data coming from IMU and pose while the quadrotor performs Take Off and Land flight manoeuvres

This easy script permits the Crazyflie to take off to a fixed height of 0.4 meters and to maintain it for several seconds (hovering). A video of the test can be found at the following link: **1cf-hovering**. In Figure 4.17 the data relative to the cmd_position (only along the z-axis), the actual position (x, y, z), the orientation (only pitch and roll angles) are displayed thanks to PlotJuggler.
The same manoeuvres can be commanded easily to more than one Crazyflie, by modifying the *position.launch* file: more groups can be added, each one representing one of the Crazyflies, each one with a specified URI (there is the possibility to connect all the quadrotors to the same radio or to several radios). In the test of three Crazyflies (link: **3cf-hovering**), the cmd_position corresponds to a step function: starting from an altitude of 0.6 meters, it is decreased by a step of 0.2 meters every 2.5 seconds, until the land. The resulting plots are shown in Figure 4.18.

4.7.2 Take off - Go To - Land

The same script was modified in order to include additional steps between the Take Off and the Land: from the initial position, stored at the Crazyflie's switching on, relative displacements were defined in the xy-plane.
By considering the case of the single Crazyflie, the mod_position is modified by adding the following general part of code:

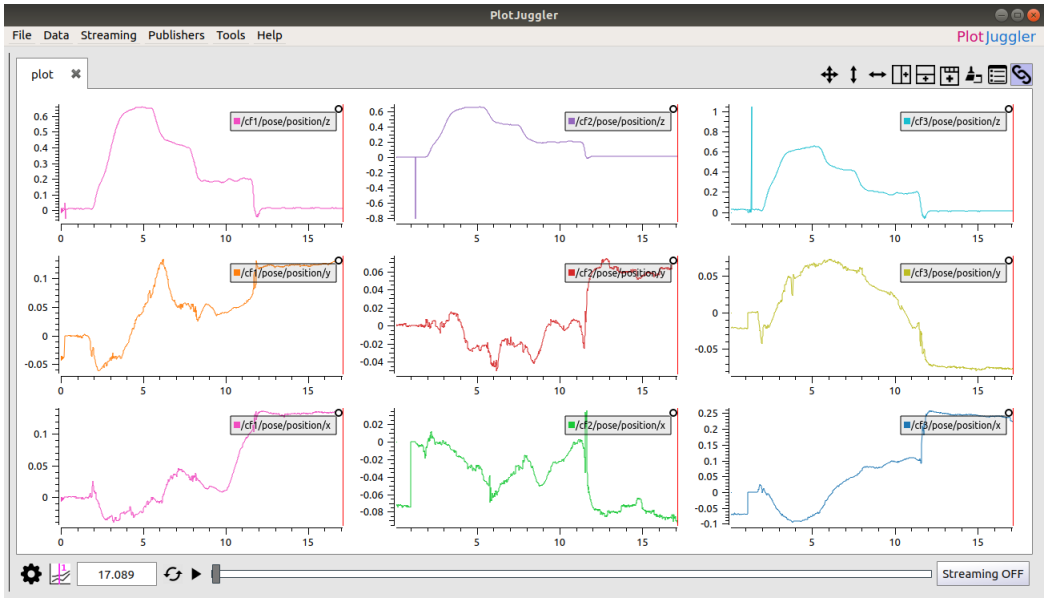


Figure 4.18: Plot of data coming from IMU and pose while three quadrotors perform hovering manoeuvres

```
start = rospy.get_time()
    while not rospy.is_shutdown():
        msg.x = x
        msg.y = y
        msg.yaw = yaw
        msg.z = 0.4
        now = rospy.get_time()
        if (now - start > 2.0):
            break
        msg.header.seq += 1
        msg.header.stamp = rospy.Time.now()
        rospy.loginfo("sending...")
        rospy.loginfo(msg.x)
        rospy.loginfo(msg.y)
        rospy.loginfo(msg.z)
        rospy.loginfo(msg.yaw)
        pub.publish(msg)
        rate.sleep()
```

and, step by step, by specifying the desired x, y and ψ.
In the tests, there were fixed different point in the trajectory, corresponding to the angles of a poster, such that the Crazyflie, after the taking off, follows the

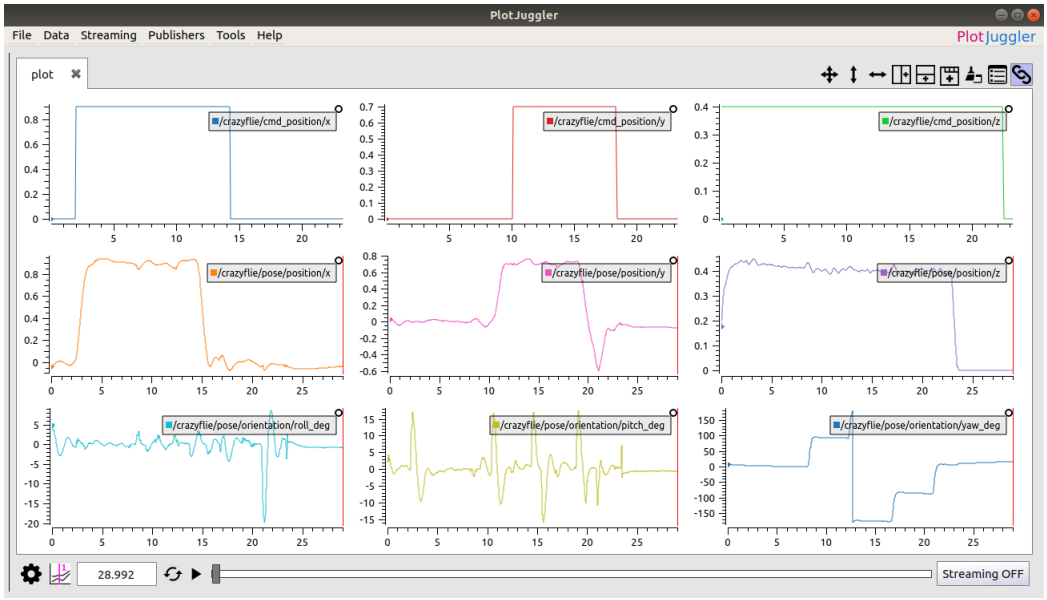


Figure 4.19: Test of a Crazyflie following the perimeter of a poster

perimeter of the poster until it reaches the starting position and it lands (link: **1cf-perimeter-following**). The corresponding plots are displayed in the Figure 4.19. The same test is expanded to two Crazyflies (link: **2cf-perimeter-following**).

4.7.3 Creation of my workspace

Until now, a script already provided in the Crazyflie workspace was applied and the quadrotor motion was performed by stages.
So, next step was the creation of a new workspace with a new Python code in which specify the publishers and published variables and to which one to subscribe, define a continuous trajectory. Starting from scratch, the following steps were followed to create the new package *crazyswarm*:

- Create the package: in a new terminal write

```
1    cd crazyflie_ws/src
2    catkin_create_pkg mycrazyswarm rospy roscpp std_msgs
     sensor_msgs geometry_msgs
3    cd ..
4    catkin_make
5
```

- Create a new C++ script corresponding to the new node *crazyswarm_node*, in *crazyflie_ws/src/mycrazyswarm/src/*:
  - Include libraries and messages;
  - Copy the main function from a basic example;
  - Define the function to subscribe to the topic and publish it;
  - define the global and the local variables and initialize them;
  - define your trajectory.
- add in the workspace the folder containing the messages and .srv files, taken from the Crazyflie workspace;
- modify the CMakeList:
  - add *message_generation* and *crazyflie_cpp* in *find_package*();
  - uncomment *add_service_file* and add all the *.srv* file in *add_service_files*() function as well as all *.msg* files in the *add_message_files*();
  - uncomment dependences *std_msgs* and *geometry_msgs*;
  - uncomment:

```
catkin_package(
INCLUDE_DIRS include
CATKIN_DEPENDS geometry_msgs roscpp rospy sensor_msgs
std_msgs
DEPENDS system_lib )
```

  - uncomment:

```
add_executable(${PROJECT_NAME}_node src/
crazyswarm_node.cpp)

set_target_properties(${PROJECT_NAME}_node PROPERTIES
OUTPUT_NAME node PREFIX "")

add_dependencies(${PROJECT_NAME}_node ${${
PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS
})

target_link_libraries(${PROJECT_NAME}_node
${catkin_LIBRARIES}
)
```

At this point, the *crazyswarm_node* will be fully created, it follows the complete script:

```
#include <ros/ros.h>
#include <math.h>
#include <time.h>
#include "geometry_msgs/Twist.h"
#include "geometry_msgs/PointStamped.h"
#include "geometry_msgs/PoseStamped.h"
#include "mycrazyswarm/Position.h"
#include "sensor_msgs/Imu.h"
#include "sensor_msgs/Temperature.h"
#include "sensor_msgs/MagneticField.h"
#include "std_msgs/Float32.h"

// global variables
double rad2deg = 180.0/M_PI;
double start_time;
double current_time;
struct timespec gettime_now;

// cfx variables
class Crazyflie {
  public :
  double position_x, position_y, position_z;
  double roll, pitch, yaw;
  double init_position_x, init_position_y, init_position_z;
  int once = 0;
  double radius = 0.4;
  double omega =  2;
  mycrazyswarm::Position cmd_msg;
};
Crazyflie cf1;

void poseCallback1(const geometry_msgs::PoseStamped::ConstPtr& msg)
{
    cf1.position_x= msg->pose.position.x;
    cf1.position_y= msg->pose.position.y;
    cf1.position_z= msg->pose.position.z;
    if(!cf1.once){
      cf1.init_position_x = cf1.position_x;
      cf1.init_position_y = cf1.position_y;
      cf1.init_position_z = cf1.position_z;
      cf1.once = 1;
    }

```

```
44      cf1.roll = rad2deg*atan2(2.0*(msg->pose.orientation.y*msg->pose.
    orientation.z + msg->pose.orientation.w*msg->pose.orientation.x),
    msg->pose.orientation.w*msg->pose.orientation.w - msg->pose.
    orientation.x*msg->pose.orientation.x - msg->pose.orientation.y*
    msg->pose.orientation.y + msg->pose.orientation.z*msg->pose.
    orientation.z);
45      cf1.pitch = rad2deg*asin(-2.0*(msg->pose.orientation.x*msg->pose.
    orientation.z - msg->pose.orientation.w*msg->pose.orientation.y));
46      cf1.yaw = rad2deg*atan2(2.0*(msg->pose.orientation.x*msg->pose.
    orientation.y + msg->pose.orientation.w*msg->pose.orientation.z),
    msg->pose.orientation.w*msg->pose.orientation.w + msg->pose.
    orientation.x*msg->pose.orientation.x - msg->pose.orientation.y*
    msg->pose.orientation.y - msg->pose.orientation.z*msg->pose.
    orientation.z);
47
48  int main(int argc, char **argv)
49  {
50      ros::init(argc, argv, "mycrazyswarm");
51      ros::NodeHandle n;
52
53      ros::Publisher pos_pub1 = n.advertise<mycrazyswarm::Position>("/
    cf1/cmd_position", 10);
54      ros::Subscriber sub1 = n.subscribe("/vrpn_client_node/cf1/pose",
    1000, poseCallback1);
55
56      //initialise every vrpn_client_nodevariable
57       cf1.cmd_msg.header.seq = 0;
58       cf1.cmd_msg.header.stamp = ros::Time::now();
59       cf1.cmd_msg.x = 0;
60       cf1.cmd_msg.y = 0;
61       cf1.cmd_msg.z = 0;
62       cf1.cmd_msg.yaw = 0;
63
64      ros::Rate loop_rate(10);
65      int count = 0;
66
67      //start time
68      clock_gettime(CLOCK_REALTIME, &gettime_now);
69      start_time =  gettime_now.tv_sec + 0.001*gettime_now.tv_nsec
    /1000000.0;
70      ROS_INFO("Start Position Controller");
71
72      while (ros::ok())
73      {
74        clock_gettime(CLOCK_REALTIME, &gettime_now);
75         current_time = gettime_now.tv_sec + 0.001*gettime_now.tv_nsec
    /1000000.0 - start_time;
76         if(cf1.once){
77           cf1.cmd_msg.header.seq = count ;
```

```
78          cf1.cmd_msg.header.stamp = ros::Time::now();
79          cf1.cmd_msg.x = cf1.init_position_x;
80          cf1.cmd_msg.y = cf1.init_position_y;
81          if(0.05*current_time + cf1.init_position_z +0.35 < 1){
82            cf1.cmd_msg.z = 0.05*current_time + cf1.init_position_z
    +0.35 ;
83          }
84          else {
85            cf1.cmd_msg.z = 1;
86          }
87          cf1.cmd_msg.yaw = 0;
88       pos_pub1.publish(cf1.cmd_msg);
89     }
90           ros::spinOnce();
91           loop_rate.sleep();
92          ++count;
93         }
94    return 0;
95       }
```

In this script, the Data about the actual position and orientation are coming from the Optitrack system. The launch file is similar to the ones already showed in the previous subsections. The recorded test on Motive can be seen at the link: **optitrack-record**.
As mentioned before, with this new node is possible to define a continuous trajectory and, as before, several crazyflie can be activated at the same time. For example, the code of *crazyswarm_node* was modify in order to let two Crazyflies perform a spiral trajectory: starting from the addition of the second Crazyflie to the class, defining the local variables, duplicate the function to subscribe to the topic and initialize all the variables.
The aim was, positioning the two Crazyflies in the extremities of the diameter of a circumference, to let them follow the perimeter always maintaining an angular distance of 180° between them. The equations describing the trajectory are:

$$\begin{cases} x_1 = -R\sin\omega t + x_C; \\ y_1 = R\cos\omega t + y_C. \end{cases} \qquad \begin{cases} x_2 = R\sin\omega t + x_C; \\ y_2 = -R\cos\omega t + y_C. \end{cases}$$

with

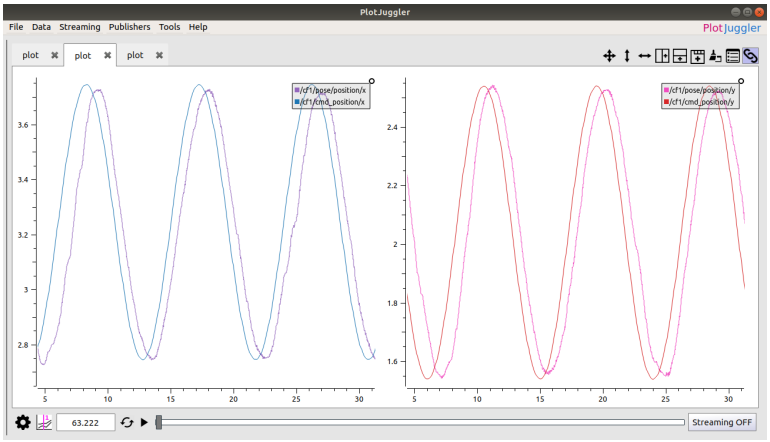
$$C = (x_C, y_C)\,(x_{10}, y_{10} - R) \equiv (x_{20}, y_{20} + R)$$

and, on the *crazyswarm_node* as:

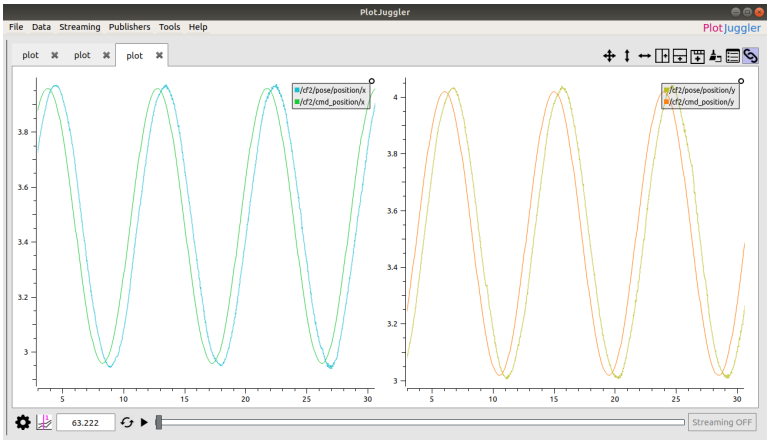
```
76 if(cf1.once && cf2.once){
77         \\cf1 trajectory
78           cf1.cmd_msg.header.seq = count ;
```

```
79          cf1.cmd_msg.header.stamp = ros::Time::now();
80          cf1.cmd_msg.x = -cf1.radius*sin(cf1.omega*current_time) +
    cf1.init_position_x;
81          cf1.cmd_msg.y = cf1.radius*cos(cf1.omega*current_time) +
    cf1.init_position_y - cf1.radius;
82          if(0.05*current_time + cf1.init_position_z +0.2 < 1.5){
83            cf1.cmd_msg.z = 0.05*current_time + cf1.init_position_z
    +0.2 ;
84          }
85          else {
86            cf1.cmd_msg.z = 1.5;
87          }
88          cf1.cmd_msg.yaw = 0;
89
90        //cf2 trajectory
91            cf2.cmd_msg.header.seq = count ;
92            cf2.cmd_msg.header.stamp = ros::Time::now();
93            cf2.cmd_msg.x = cf2.radius*cos(M_PI/2.0 - cf2.omega*
    current_time) + cf2.init_position_x;
94            cf2.cmd_msg.y = -cf2.radius*sin(M_PI/2.0 - cf2.omega*
    current_time) + cf2.init_position_y + cf2.radius;
95            if(0.05*current_time + cf2.init_position_z + 0.2< 1.5){
96               cf2.cmd_msg.z = 0.05*current_time + cf2.
    init_position_z + 0.2;
97             }
98            else {
99              cf2.cmd_msg.z = 1.5;
100            }
101           cf2.cmd_msg.yaw = 0;
```

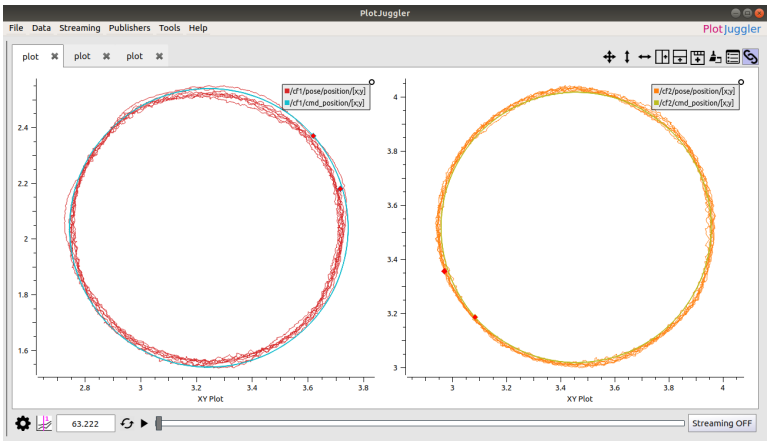
The resulting plots of the test are showed in the Figure 4.20: in detail, in Figure 4.20(a) can be found a comparison between desired and actual x-positions with respect of the time, in Figure 4.20(b), instead, desired and actual y-positions with respect of time are compared and, at the end, in Figure 4.20(c) the actual trajectory overlapped to the desired one in the xy-plane. The test can be seen at the link: **2cf-spiral-trajectory**.



(a)



(b)



(c)

Figure 4.20: Plot of two Crazyflies following a spiral trajectory

Chapter 5

Concluding Remarks and Perspectives

For the sake of clarity, let's recall the main goals of this thesis, (i) to identify the best formation regarding consensus performance, (ii) to verify the position and velocity matching consensus algorithms, (iii) to propose a control model of a three-agent formation and (iv) to develop the test environment envisioning future implementations.
Initially, the cross-linking relationship between agents spatial configuration and sensory module perception profile is attested, regarding information flow and thus the underlying consensus stability (formation). In the actual thesis, it is assumed a constant topology implying a specific perception profile (radius of sensitivity area, angular overture, ...) and considering constant inter-agent distance. Likewise, for modeling and simulation purposes, it is considered static topologies and linear kinematics and dynamics as well. Moreover, as a result of the conducted controllability and observability analysis, formations based on undirected topologies results to be observable in most cases. The latter allowed us to shape the consensus algorithm structure by considering the ideal Laplacian matrix (the topology that mirrors the best communication flow) as the theoretical state-space model, whereas, the specific cases (communication/perception anomalies) are considered in the output equation of the state-space vector. Moreover, it is investigated whether obstacles should be considered as agents, resulting in the extension of state-space vector, might enhance observability condition. Conversely, a performance degradation is faced, rather than an improvement. In fact, including states coming from obstacles degrades the graph connectivity degree and, as demonstrated on prior chapters, connectivity is strictly and directly related to the consensus convergence rate, i.e. the more connected the graph is, the more faster consensus is reached. Within the simulation study, only the formations considering circular-shaped perception

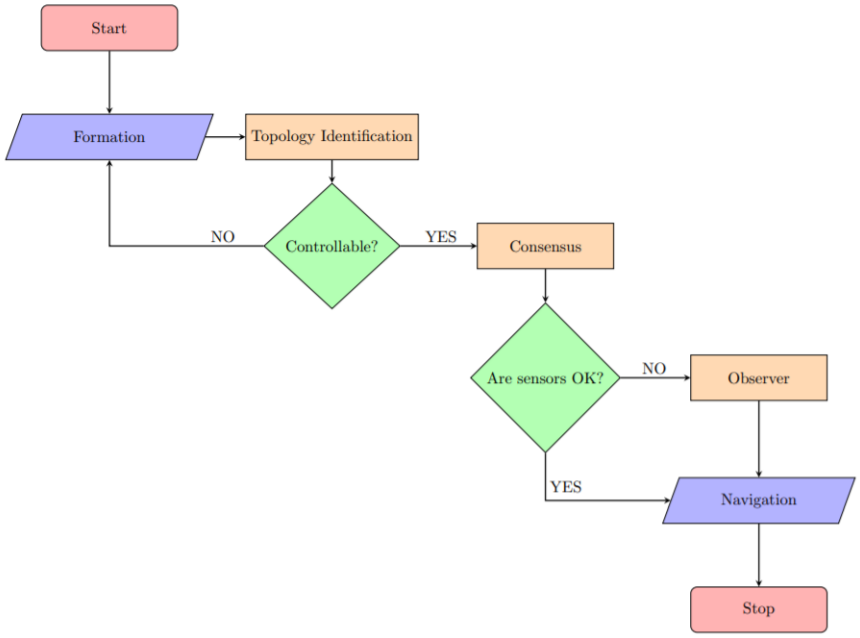


Figure 5.1: Logic flow of the model simulation

sensors, since it stood out as the most suitable case meeting observability and controllability properties.
For pedagogical purposes, these ideas are sketched on the flow chart depicted by Figure 5.1. It starts from the topology, the immediate step is to verify the controllability and, whether the system is controllable, the consensus algorithm can be applied.
It is essential to verify if the sensors are correctly working, in the negative case an observer has to be applied in order to recover the missing states. The final output corresponds to the navigation.

The preliminary experimental stage is based on the Crazyflie with interesting outcomes. The main advantages of this quadrotor are the mechanical simplicity leading to high reliability and low maintenance cost, as well as the flexibility on the implementation. In fact, the working environment based on the ROS framework is defined and implemented with an operational Crazyflie controller.
In spite of the appealing features of the benchmark, it is worth to highlight experimental issues/tips that should be revisited for further project stages:

- On-board sensor can be easily damaged
- Communication link between the workstation (master) and the quadrotors (clients) can not be assured for long periods.

- Even if the reduced size of the quadrotor is an advantage, it is, on the other hand, quite sensitive to external forces. In our case, the Optitrack markers disturbed significantly the stabilized-flight of the vehicle.
- Experimental area must be free of reflective surfaces that will degrade the sensing of the quadrotor on-board Optitrack markers.

The actual thesis lies within the initial phase of an undergoing project of the XLIM robotics department. The thesis provides a theoretical and experimental basis regarding the next stage of the project addressed in this work. Forthcoming research includes:

- Navigation consensus based on time-varying topology
- Robust obstacle avoidance based on formation morphing while preserving, via the observer, undirected topology, e.g. navigating trough cluttered environments as corridors, windows.
- Distributed observer-based estimation of external disturbances, e.g. wind gusts

Appendix A

Matrix Theory

A brief introduction to matrix theory is given in this subsection: starting from the definition, a matrix $A \in \mathbb{R}^{n \times m}$, is defined as a rectangular array ordered in rows, n, and columns, m, such that the element a_{ij} corresponds to the matrix's element in the i-th row and in the j-th column.
The main operations that will be used in this research are:

- Addition/Subtraction: it is performed by adding or subtracting the elements in the same ith row and j-th column.

$$A + B = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{21} & b_{31} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{13} + b_{13} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} \\ a_{31} + b_{31} & a_{32} + b_{32} & a_{33} + b_{33} \end{bmatrix}$$

  The addition/subtraction operation is commutative and associative:

$$A + B = B + A \qquad (A + B) - C = A + (B - C)$$

- Scalar Multiplication/Division: It is performed by multiplicating/dividing each matrix element by the scalar term:

$$c \cdot A = c \cdot \begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} c \cdot a_{11} & c \cdot a_{12} \\ c \cdot a_{21} & c \cdot a_{22} \end{bmatrix}$$

- Matrix Multiplication: when executing the multiplication between matrices,

the elements of the rows in the first matrix are multiplied with the corresponding columns in the second matrix, as follow:

$$A \cdot B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{21} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

- Determinant: it can be calculated only in the case of a square matrix and it is calculated with different ways depending on the dimension.
  Firstly consider a square matrix of dimension $n = 2$, the determinant is calculated as:
  $$det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = (a \cdot d) - (b \cdot c)$$
  The determinant of a square matrix $A \in \mathbb{R}^{n \times n}$ with $n > 2$, it is more complex and is calculated from the definition of determinant of a 2×2 matrix, as follow:
  $$det\,(A) = \sum_{j=1}^{n} \left[a_{ij}\,(-1)^{i+j}\,det\,(A_{ij}) \right]$$
  and, so, for a matrix with $n = 3$:
  $$det \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} =$$
  $$a_{11}\,(a_{22}a_{33} - a_{23}a_{32}) - a_{21}\,(a_{12}a_{33} - a_{13}a_{32}) + a_{31}\,(a_{12}a_{23} - a_{13}a_{22})$$

Appendix B

Installation of ROS Melodic Morenia

In this section are recovered the steps to download and initialize ROS Melodic Morenia on Ubuntu 18.04.

- Set permission to support software form packages.ros.org:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(
    lsb_release -sc) main" > /etc/apt/sources.list.d/ros-
    latest.list'
```

- Set up keys:

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80
    ' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

- Install Ros Melodic:

```
sudo apt install ros-melodic-desktop-full

```

- Add ROS environment variables to bash:

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc

```

- Install and initialize additional tools useful to create and manage a ROS workspace:

```
sudo apt install python-rosdep python-rosinstall python-
    rosinstall-generator python-wstool build-essential
sudo rosdep init
rosdep update

```

- Create your catkin workspace:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make

```

- Add the new workspace to the bash:

```
sudo nano ~/.bashrc
```

and add at the end of the file the following line:

```
source ~/catkin_ws/devel/setup.bash
```

Now all the new ROS Melodic environment is set.

Appendix C

ROS Basics

At this moment, some basic and essential concepts[60][61] are given to start using correctly the ROS environment

Master

The ROS Master manages the communication between the node, by tracking publishers and subscribers to topics as well as services. The following command starts the master, without the need of configuration:

```
roscore
```

Once the master is activated, it should be remain active for the whole period of operation, so the work has to continue in others terminals.

Nodes

The node is defined as a running instance on ROS. In fact, the software is designed as a set of almost independents programs (nodes) that communicate and all run at the same time.
The basic command to run a node is:

```
rosrun package-name executable-name
```

To obtain a list of the nodes running presently:

```
rosnode list
```

Note that *rosout* node is always present and it is automatically started by *roscore*, it corresponds to a standard output.
To get some information from a specific node (if the node is a publisher or a subscriber, the services offered by the node, ...), use the following command:

```
rosnode info node-name
```

In conclusion, to kill the node:

```
rosnode kill node-name
```

Packages

The packages are the main blocks composing the ROS software, consisting in a collection of files, the nodes and the executable program.
The following command gives the list of all the installed ROS packages:

```
rospack list
```

Instead, to visualize the files in the package directory, use:

```
rosls package-name
```

Subscriber and Publisher

A node that sends information to another is called Publisher, the node that receives the information is the Subscriber.

Topics and Messages

The communication between nodes is based on an exchanging of messages, that are grouped in topics.
It follows a list of useful commands:

```
rostopic list                    %check the list of the active
   topics
```

```
rostopic echo topic-name        %see the actual  messages that
   are being published on a topic

rostopic info topic-name        %have more information about a
   topic (message type, publisher and subscribers)

rosmsg show message-type-name   %details about a message type
```

Services

In ROS many publishers and many subscribers can share the same topic, so ROS provides a mechanism (the services) to implement one-to-one communication between the nodes. Services are defined in srv files.

It is important to remark the difference between

```
rosrun package-name node-name
```

and

```
roslaunch package_name file.launch
```

The first one allows to execute directly a node, the second permits to execute many nodes together. These two commands will be used often in the experimental work.

Example:
While running a ROS program, it's possible to graphically screen the nodes and the exchange of messages. If it isn't already installed, open a new terminal and write

```
sudo apt-get install ros-melodic-rqt

sudo apt-get install ros-melodic-rqt-common-plugins
```

To open the *rqt_graph*, use the command:

```
rosrun rqt_graph rqt_graph
```

and a new window will be opened, as the one in the Figure C.1. In this example, it's easy identify that */crazyflie/crazyflie_demo_hover* is the publisher node,

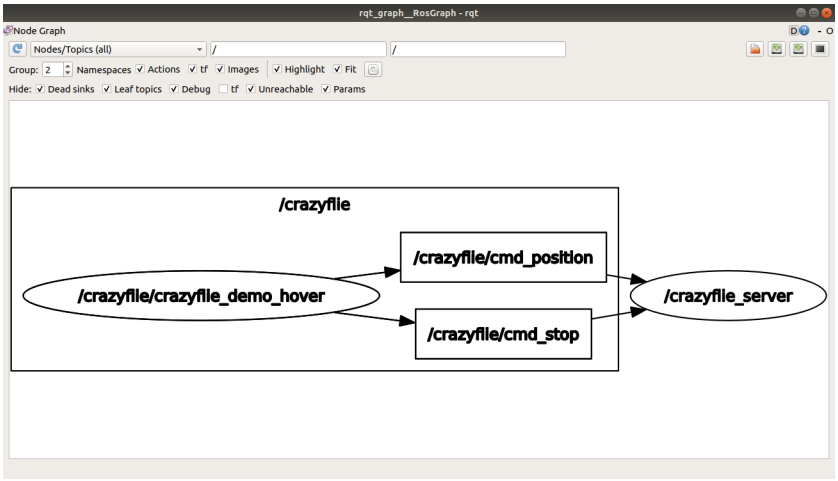


Figure C.1: Example of a rqt graph

/crazyflie_server subscribes to the */crazyflie/cmd_position* and */crazyflie/cmd_stop* topics.

Bibliography

[1] C.W. Reynolds. *Flocks, Herds, and Schools: A Distributed Behavioral Model*. Computer Graphics, 21, 1987, pp. 25–34 (cit. on pp. 1, 15).

[2] R. Olfati-Saber. «Flocking for multi-agent dynamic systems: algorithms and theory». In: *IEEE Transactions on Automatic Control* vol.51, no.3 (2006), pp. 401–420. DOI: 10.1109/TAC.2005.864190 (cit. on p. 2).

[3] T. Balch and R.C. Arkin. «Behavior-based formation control for multirobot teams». In: *IEEE Transactions on Robotics and Automation* vol. 14, no.6 (1998), pp. 926–939. DOI: 10.1109/70.736776 (cit. on p. 2).

[4] T. Vicsek, A. Czirok, E. Ben-Jacob, I. Cohen, and O. Schochet. «Novel type of phase transitions in a system of self-driven particles». In: *Physical Review Letters* vol. 75, no. 6 (1995), pp. 1226–1229 (cit. on p. 2).

[5] A. Jadbabaie, J. Lin, and A.S. Morse. «Coordination of groups of mobile autonomous agents using nearest neighbor rules». In: *IEEE Trans. Autom. Control* vol.48, no.6 (2003), pp. 985–1001 (cit. on pp. 2, 18).

[6] R. Olfati-Saber, J.A. Fax, and R.M. Murray. *Consensus and Cooperation in Net-worked Multi-Agent Systems*. Proceedings of the IEEE 95.1, 2007, pp. 215–233 (cit. on pp. 2, 18).

[7] R. Olfati-Saber and R.M. Murray. *Consensus Problems in Networks of Agents With Switching Topology and Time-Delays*. Vol. 49, no.9. IEEE Transactions on automatic control, 2004 (cit. on pp. 2, 18, 43).

[8] J.A. Fax and R.M. Murray. *Information flow and cooperative control of vehicle formations*. Vol. 49, no. 9. IEEE Trans. Automat. Contr., 2004, pp. 1465–1476 (cit. on pp. 2, 18).

[9] G. Albi, M. Herty, and L. Pareschi. «Kinetic description of optimal control problems and applications to opinion consensus». In: *Communications in mathematical sciences* (2014). DOI: 10.4310/CMS.2015.v13.n6.a3 (cit. on p. 2).

[10] W. Ren, R. W. Beard, and E. M. Atkins. *Information Consensus in Multivehicule Cooperative Control*. Vol. 27, n.2. IEEE Control Systems Magazine, 2007 (cit. on pp. 2, 19, 48).

[11] H.G. Tanner, A. Jadbabaie, and G.J. Pappas. *Stable Flocking of Mobile Agents, Part I: Fixed Topology*. Proc. of the 42nd IEEE Conference on Decision and Control, Maui, 2003 (cit. on pp. 2, 50).

[12] H.G. Tanner, A. Jadbabaie, and G.J. Pappas. *Stable Flocking of Mobile Agents, Part II: Dynamic Topology*. Proc. of the 42nd IEEE Conference on Decision and Control, Maui, 2003 (cit. on p. 2).

[13] G. Wen, Z. Duan, W. Yu, and G. Chen. «Consensus in Multi-agent System with communication constraints». In: *Int. J. Robust. Nonlinear Control*. 2011 (cit. on pp. 2, 20, 42).

[14] W. Ren and R.W. Beard. «Consensus seeking in multiagent systems under dynamically changing interaction topologies». In: *IEEE Trans. Autom. Control* vol.50, no.5 (2005), pp. 655–661 (cit. on p. 2).

[15] X. Li, T.T. Han, and S. S. Ge. «Consensus for Multi-agent System with Bounded Control in Limited Communication». In: *3rd IFAC International Conference on Intelligent Control and Automation Science*. Chengdu, China, 2013 (cit. on p. 2).

[16] H. Li, X. Liao, T. Dong, and L. Xiao. *Second-order consensus seeking in directed networks of multi-agent dynamical systems via generalized linear local interaction protocols*. Vol. 70. Nonlinear Dynamics, 2012, pp. 2213–2226 (cit. on p. 2).

[17] L. Moreau. «Stability of multi-agent systems with time-dependent communication links». In: *IEEE Trans. Autom. Control* vol.50, no.2 (2005), pp. 169–182 (cit. on p. 3).

[18] N.E. Leonard and E. Fiorelli. «Virtual Leaders, Artificial Potentials and Coordinated Control of Groups 1». In: 2001 (cit. on p. 3).

[19] R. Lozano, M. W. Spong, J. A. Guerrero, and N. Chopra. «Controllability and observability of leader-based multi-agent systems». In: *47th IEEE Conference on Decision and Control*. 2008, pp. 3713–3718. DOI: 10.1109/CDC.2008.4739071 (cit. on p. 3).

[20] M. Ji, A. Muhammad, and M. Egerstedt. «Leader-based multi-agent coordination: controllability and optimal control». In: *American Control Conference*. 2006. DOI: 10.1109/ACC.2006.1656406 (cit. on p. 3).

[21] M. Ji and M. Egerstedt. «Observability and estimation in distributed sensor networks». In: *46th IEEE Conference on Decision and Control*. 2007, pp. 4221–4226. DOI: 10.1109/CDC.2007.4434659 (cit. on p. 3).

[22] P. Andrasfai. *Introductory Graph Theory*. The Institute of Physics, 1978 (cit. on p. 5).

[23] F. Bullo. *Lecture on Network Systems*. Kindle Direct Publishing, 2019, pp. 33–55 (cit. on p. 5).

[24] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications.* Springer–Verlag Mu, 2002 (cit. on p. 6).

[25] R.B. Bapat. *Graphs and Matrices*. Springer, 2010 (cit. on p. 7).

[26] R. E. Kalman. «On the General Theory of Control Systems». In: *Proc. 1st Int. Cong. of IFAC*. Butterworth, London, 1961 (cit. on p. 9).

[27] P. G. Balaji and D. Srinivasan. *An Introduction to Multi-Agent systems*. Vol. vol. 310. Berlin Heidelberg: SpringerVerlag, 2010. Chap. 1 (cit. on p. 9).

[28] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003. Chap. 2 (cit. on p. 9).

[29] N. Kasabov and R. Kozma. «Introduction: Hybrid Intelligent adaptive systems». In: *International Journal of Intelligent Systems* (1998) (cit. on p. 9).

[30] A. Dorri, S. S. Kanhere, and R. Jurdak. «Multi-Agent Systems: A survey». In: *IEEE Access* vol. 6 (2018) (cit. on p. 10).

[31] L. Ma, H. Min, S.Wang, Y. Liu, and S. Liao. «An overview of research in distributed attitude coordination control». In: *IEEE/CAA Journal of Automatica Sinica* vol. 2.2 (2015) (cit. on p. 10).

[32] B. Batt. «Why do migratory birds fly in a V-formation?» In: *Scientific American* (Oct. 2007) (cit. on p. 10).

[33] C.Hsia Tien and Michael Soderstrand. *Development of a micro robot system for playing soccer games*. In Proceedings of the Micro-Robot World Cup Soccer Tournament, 1996, pp. 149–152 (cit. on p. 10).

[34] P.G. Balaji and D. Srinivasan. «Distributed multi-agent type-2 fuzzy architecture for urban traffic signal control». In: *IEEE International Conference on Fuzzy Systems*. 2009, pp. 1624–1632 (cit. on p. 10).

[35] A. Damba and S. Watanabe. «Hierarchical control in a multiagent system». In: *International Journal of innovative computing, Information and Control* vol. 4, no. 2 (2008), pp. 3091–3100 (cit. on p. 10).

[36] A. Esmaeili, N. Mozayani, M. R. J. Motlagh, and E. T. Matson. «The impact of diversity on performance of holonic multi-agent systems». In: *Engineering Applications of Artificial Intelligence* vol. 55 (2016), pp. 186–201 (cit. on p. 10).

[37] J.T. Emlen. *Flocking behaviour in birds*. Auk 69, 1952, pp. 160–170 (cit. on p. 13).

[38] G. Roberts. *Why individual vigilance declines as group size increases*. Anim. Behav. 5, 1996, pp. 1077–1086 (cit. on p. 13).

[39] D.V. Radakov. «Schooling in the Echology of Fish». In: *Israeli Scientific Translation Series*. Wiley, New York, 1973 (cit. on p. 13).

[40] T. Pitcher, A. Magurran, and I. Winfield. *Fish in larger shoals find food faster*. Behav. Ecol. Sociobiol. 10, 1982, pp. 149–151 (cit. on p. 13).

[41] B. Bertram. *Vigilance and group size in ostriches*. Anim. Behav. 28, 1980, pp. 278–286 (cit. on p. 13).

[42] K.J. Benoit-Bird. *Cooperative prey herding by the pelagic dolphin, Stenella longirostris*. The Journal of the Acoustical Society of America 125, 2009, pp. 125–137 (cit. on p. 13).

[43] R. Bouffanais. *Design and Control of Swarm Dynamics*. SpringerBriefs in Complexity (First ed.). Springer, 2016, pp. 125–137 (cit. on p. 13).

[44] B.L. Partridge. *The Structure and Function of Fish Schools*. Scientific American, 1982, pp. 114–123 (cit. on pp. 14, 16).

[45] J.F.V. Vincent. *Biomimetics: its practice and theory*. Journal of the Royal Society Interface. 3, 2006, pp. 471–482 (cit. on p. 14).

[46] R.M. Murray. *Recent Research in Cooperative control of Multi-Vehicle Systems*. ASME Journal of Dynamic Systems, Measurement, Control, Special issue on the "Analysis, and Control of Multi Agent Dynamic Systems", 2006 (cit. on p. 18).

[47] Z. Lin, M. Broucke, and B. Francis. *Local control strategies for groups of mobile autonomous agents*. Vol. 49, n.4. IEEE Trans. Automat. Contr., 2004, pp. 622–629 (cit. on p. 18).

[48] W. Ren and R.W. Beard. *Consensus seeking in multiagent systems under dynamically changing interaction topologies*. Vol. 50, n.5. IEEE Trans. Automat. Contr., 2005, pp. 655–661 (cit. on p. 18).

[49] R. Olfati-Saber and R.M. Murray. *Consensus Protocols for Networks of Dynamic Agents*. American Control Conference, 2003 (cit. on p. 19).

[50] H. Li and H. Su. *Second-order consensus in multi-agent systems with directed topologies and communication constraints*. Neurocomputing, 2016 (cit. on pp. 20, 51).

[51] W. Ren and E. Atkins. *Second-order consensus protocols in multiple vehicle systems with local interactions*. San Francisco, CA: Proceedings of AIAA Guidance, Navigation, and Control, 2005 (cit. on p. 20).

[52] W. Ren and E. Atkins. *Distributed multi-vehicle coordinated control via local information exchange*. International Journal of Robust and Nonlinear Control, 2007, 17(10-11):1002–1033 (cit. on p. 20).

[53] W. Yu, G. Chen, and M. Cao. *Some necessary and sufficient conditions for second-order consensus in multi-agent dynamical systems*. Automatica, 2010, 46(6):1089–1095 (cit. on p. 20).

[54] R.A. Brualdi and H.J. Ryser. *Combinatorial Matrix Theory*. Cambridge University Press, 1991 (cit. on p. 41).

[55] W. Yu, G. Chen, M. Cao, and et al. *Second-order consensus for Multi-Agent Systems with directed topologies and non-linear dynamics*. IEEETrans. Syst. Man Cybern. PartB:cCybern, 2010, pp. 881–891 (cit. on p. 41).

[56] E.W. Weisstein. *Algebraic Connectivity*. MathWorld–A Wolfram Web Resource (cit. on p. 42).

[57] F. R. K. Chung. *Spectral Graph Theory*. MathWorld–A Wolfram Web ResourceProvidence, RI: Amer. Math. Soc., 1997 (cit. on p. 42).

[58] W. Hönig and N. Ayanian. «Flying Multiple UAVs Using ROS». In: *Koubaa A. (eds) Robot Operating System (ROS). Studies in Computational Intelligence* vol.707, Springer (2017) (cit. on p. 60).

[59] J. A. Preiss, W. Hönig, G. S. Sukhatme, and N. Ayanian. «Crazyswarm: A large nano-quadcopter swarm». In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 3299–3304. DOI: 10.1109/ICRA.2017.7989376 (cit. on p. 60).

[60] J.M. O'Kane. *A Gentle Introduction to ROS*. CreateSpace Independent Publishing Platform, 2013, pp. 17–40 (cit. on p. 104).

[61] Y. Pyo, H. Cho, R. Jung, and T. Lim. *ROS Robot Programming*. ROBOTIS CO., 2017, ch.4 (cit. on p. 104).