Politecnico di Torino

Master of Science in Mechatronics Engineering

Master Degree Thesis

Decentralized control for multi-robot systems with event-triggered communications

Supervisor:
Prof. Alessandro Rizzo

Candidate:
Pierluigi Pugliese

Co-supervisors:
Prof. Fabrizio Caccavale
Prof. Francesco Pierri

Academic Year 2020/21
Torino

Acknowledgements

Throughout the writing of this thesis I have received all the support I needed to finalize the work. I would first like to thank my co-supervisors Professor Fabrizio Caccavale and Professor Francesco Pierri for giving me the chance to work in this project and for having always helped me out in drafting my thesis. I would also like to thank my supervisor Professor Alessandro Rizzo for showing interest in my project and in its applications. A big thanks to my parents and my brother to be always by my side and to give me always wise advice. You are always there for me.

In addition, I would like to thank my friends for providing stimulating discussions as well as happy distractions to rest during spare time.

Abstract

Multi-robot systems (MRSs) have been widely studied due to their features as flexibility and intrinsic resilience. Classical approaches to multi-robot control are based on centralized architectures, which are very effective in terms of performance but require a central unit and/or complete communication among all robots. This feature might be difficult to meet in practical scenarios or might be undesirable when mobile robots are involved. Therefore, recent studies have been mainly focused on decentralized or distributed systems, as they are better scalable and safer than centralized ones. Decentralized algorithms applied to networked robots should find out control and communication problems such as controlling the MRS centroid and the formation.

This thesis deals with the simulation of a decentralized centroid and formation control, based on a distributed controller-observer scheme for a team of mobile agents (robots). In this context, each robot is characterized by a single-integrator dynamics $\dot{\boldsymbol{x}} = \boldsymbol{u}$ where $\boldsymbol{x}$ is the state and bfu is the input vector. A local observer is used by each agent to estimate the collective state of the system, while the distributed controller is in charge of desired centroid and formation tracking. This design leads to relevant advantages with respect to other layouts:

- each agent evaluates the state of the whole system, which can be exploited to achieve other goals (e.g., fault detection);
- estimation of the whole state results in a strong (i.e., exponential) convergence of the estimation and tracking errors.

To reduce the exchange of information, each robot can communicate only with its neighbours. In the thesis, a multi-robot system characterized by a fixed and strongly connected directed communication graph is considered. Furthermore, in order to decrease the exchange of information between the robotic agents, an event-triggered

control strategy is devised, where each agent communicates with its neighbours only when a triggering condition on the estimation error is verified. Two different triggering conditions are implemented: one based on a Lyapunov's function for the estimation error and the other based on the dynamics of the estimation error. The triggering conditions are verified in simulation and their performances are compared. Matlab and Simulink have been used to perform the simulations.

Contents

List of Figures

Chapter 1

Introduction

Robots have been developed to help people in carrying out different activities, from ordinary tasks to crucial industrial applications. Automation has reached a wide diffusion in several industries as the robots are becoming faster, more effective and more reliable than human beings. Robotics has been widely applied from logistic and transportation sector, to factories and health-care. In these contexts, robots have shown outstanding capabilities to interact between themselves by cooperating in such a way that each of them is effectively able to complete its own task. Development of completely autonomous robots has been widely studied. Autonomous robots are the pinnacle of automation; they can receive commands to execute tasks and perform them without the human interventions. Instructions given to agents provide descriptions on what the robot is expected to do rather than indicate how to do it. Automation has made giant strides. While before robotic research has led to base-fixed robotic manipulator (Figure 1.1),lately another type of robots has become widely used, namely robots with a mobile base (Figure 1.2). These robots are able to navigate through manufacturing plants, in external environment or on battlefields. Some of them perform tasks being remotely controlled by an operator; however, most of these robots are not autonomous, using sensors to allow their operator remote access to perilous, distant or inaccessible places. Some of them can be semi-autonomous, performing sub-tasks automatically, (i.e. the autopilot of a drone stabilizes the flight while the human chooses the flight track). Fully autonomous mobile robots perform tasks on their own, such as transporting material while navigating in different types of terrain (i.e. warehouses, buildings, rough terrain) and in a dynamic environments (i.e. human-robot cohabitated spaces, cars

on the streets).

Figure 1.1. Example of base-fixed robot

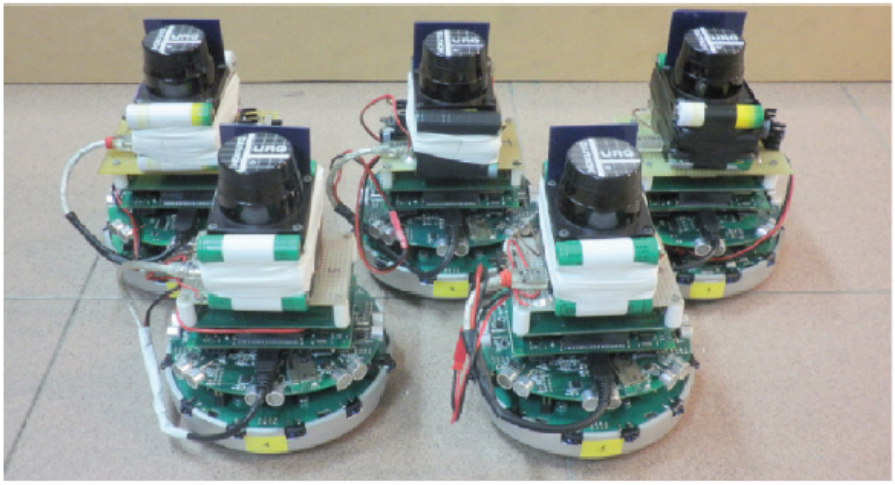

Figure 1.2. Example of mobile robots

1.1 Categories of mobile robots

Motion is the main issue concerning mobile robots overall. Robots operate in planned and controlled scenario, however robots are often used in adverse and extreme environments. Mobile robots can also operate in disparate surroundings, not

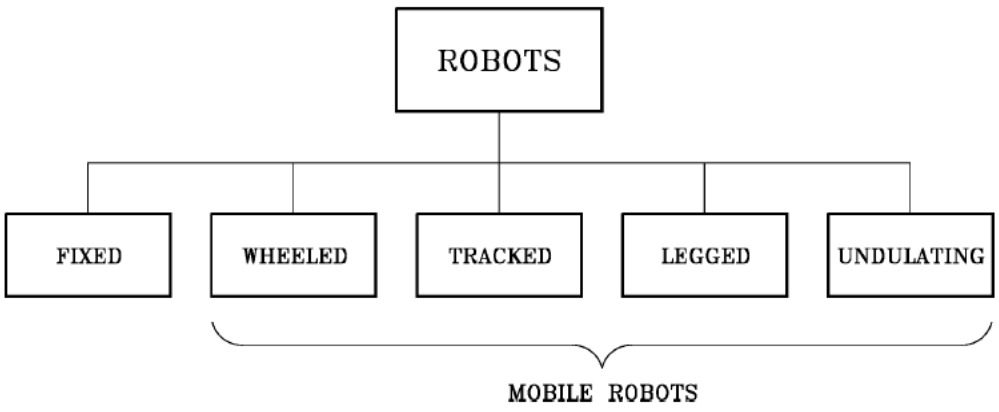

Figure 1.3. Classes of robots

only on the ground as underwater robots are particularly effective in ocean observation or flying drones being applied to patrol perimeters. Robots' motion system represents a crucial task in mobile robots' design by affecting robots operations as well as conditions such as, *inter alia*, maneuverability, controllability, terrain conditions, stability, efficiency. Based on movement system, mobile robots can be classified, as follows:

- Base-fixed (manipulators);
- Land-based:
  - Wheeled mobile robot;
  - Walking (legged) mobile robot;
  - Tracked slip/skid movement;
  - Hybrid;
- Air-based; and
- Water-based.

Wheeled mobile robots are commonly used in several applications such as transportation of various materials inside a warehouse or in the exploration of unknown

environments, being able to consume less energy and move faster compared to other motion mechanisms (i.e. legged robots) explain their widespread application. From a control standpoint, less effort is required, due to their simple mechanisms and reduced stability problems. Although one of the biggest disadvantages of the wheeled robots is their struggle to overcome rough terrain or uneven ground, they are suitable for a wide class of tasks and several applications. Wheeled robots can have disparate outlooks, depending on the number and the configuration of the wheels. On a flat ground three wheels are enough for a robot to balance, more wheels can be added, but more difficult operations must be applied to maintain all the wheels on the ground if the surface is not flat. Based on the wheel's setting, most common wheeled robots can have two, three, four wheels and in some particularly cases even more (Figure 1.4). A bicycle is an example of a two-wheeled mobile robot, another type of a two-wheeled mobile robot is equipped with two independent driving motors, one for each wheel. Many of car-like robots can have dissimilar driving design, like rear-wheel drive, front-wheel drive or four-wheel drive. Legs are another widely used form of locomotion. They are usually more expensive than wheels, legs have several advantages over wheels. The main advantage is their efficiency and the possibility to move on soft and uneven terrain. Walking robot can easily coping with obstacles or holes in the environment.

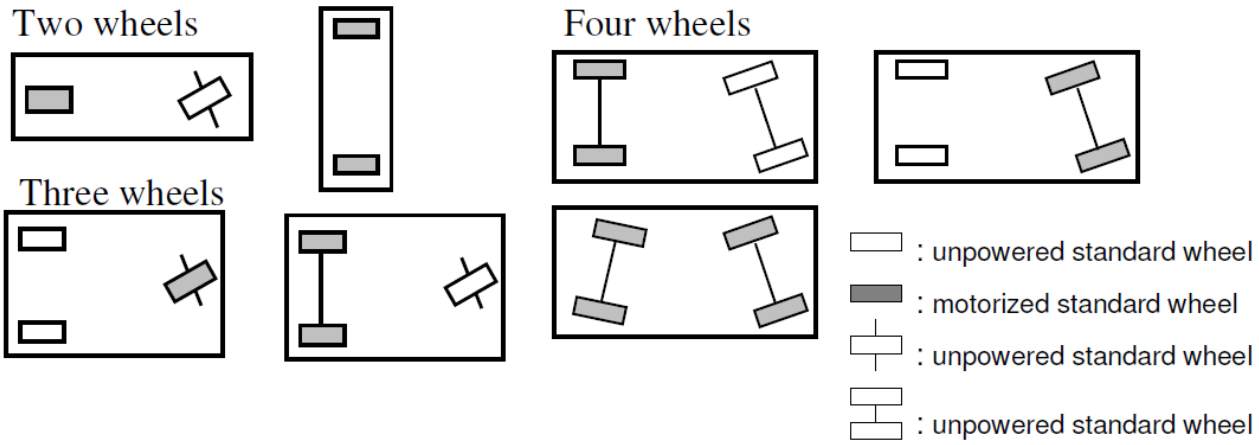

Figure 1.4. Wheel configuration for mobile robots

1.2 Multi-robot systems

A Multi-Robot System (MRS) is a group of robots operating in the same environment and coordinated with each other to achieve some goals. The robots equipment may be a simple sensor, acquiring and processing data. Nowadays MRS are complex mobile platforms, equipped with sensors and actuators, able to execute various tasks; MRS can improve the efficacy of a robotic system either in the performances, or in the robustness of the system. Even when a single robot can achieve a given task, a team of robots can improve the performance of the system. There are two main approaches to multiple robot systems: **collective swarm systems** and **intentionally cooperative systems**. **Collective swarm systems** are those where the robots can execute the given tasks with minimal knowledge of the robots in the team. In **intentionally cooperative systems** the robots are aware of the presence of other robots and act together based on the information exchanged among them. Multi-robot systems have some advantages over single-robot systems. The reasons to prefer a multi-robot system are the following:

- the task is inherently distributed;
- multiple robots can solve problems faster, due to the parallelism;
- multiple robots increase robustness through redundancy; and
- a single robot could not complete some classes of tasks, due to their complexity.

1.3 Motivations

1.3.1 Decentralized control

The control architecture for a Multi-robot system has a huge impact on the robustness and scalability of the system. A relevant difference between a single-robot system and a multi-robot system is that a MRS must manage the internal actions of the robots and coordinate the behavior of the group. There are several architectures for MRS, but they can divided into two broad categories: **centralized** and **decentralized**. A **centralized** architecture is characterized by the presence of a central control unit, which can be either one of the robots or a central processing

device, that always communicate with all the robots and knows the state of all the robots in the system. The central control unit is in charge of computing the path of all the other agents using the received information. In a real world application, a continuous communication between the agents and the central unit at a frequency capable to guarantee an efficient real-time control, is difficult to achieve. Furthermore, the system is extremely vulnerable to failures of the central unit. This type of architecture is very efficient for a small group of robots, but becomes very risky with a large number of robots. In a **Decentralized** or distributed architectures nowadays are the most widespread approaches for a MRS. In this type of architecture the agents aren't coordinated by a central unit, but each is autonomous on computing its control input. Typically in this type of architecture robots have access only to local sensed information and limited inter-agents information. This type of control architecture has numerous advantages:

- reliability: if one of the agents has a malfunction, the system adapts to the situation and reorganizes the tasks based on the circumstances;
- scalability: this type of architecture can be easily implemented for a large number of robots;
- efficiency: the same task can be accomplished in a faster way with more robots;
- flexibility: there is no need of a single powerful hardware, but more robots that have limited functions can be used.

The decentralized architecture also have some disadvantages, e.g. if the system isn't optimized, the performance can be lower than the centralized counterpart.

1.3.2 Controller-observer scheme

In a control system for many reasons, like robustness and reliability, it's useful to have a full state feedback. In real applications, it is often impossible to measure all the states of a system. To overcome this problem, it's possible to use observers, which substitutes a sensor where the measurements cannot be taken physically. The observer computes an estimate of the state of the system that can be used as state feedback of the control system. Using an observer, each agent can build a reliable image of the whole state of the team, thus a lot of function can be added to the

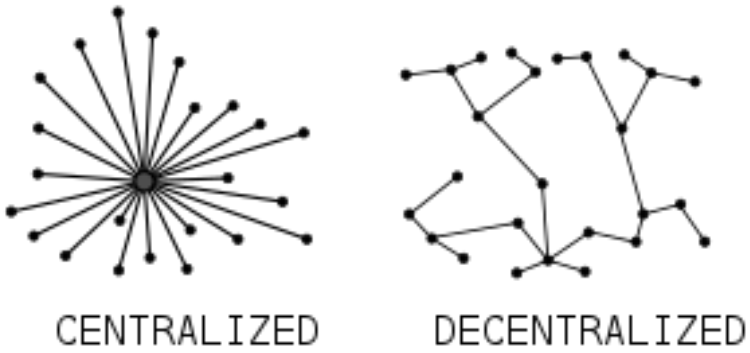

Figure 1.5. Centralized structure vs decentralized

system (e.g. fault diagnosis). Furthermore, the estimation of the whole state allows to obtain a strong convergence result, ensuring good robustness of the closed-loop system.

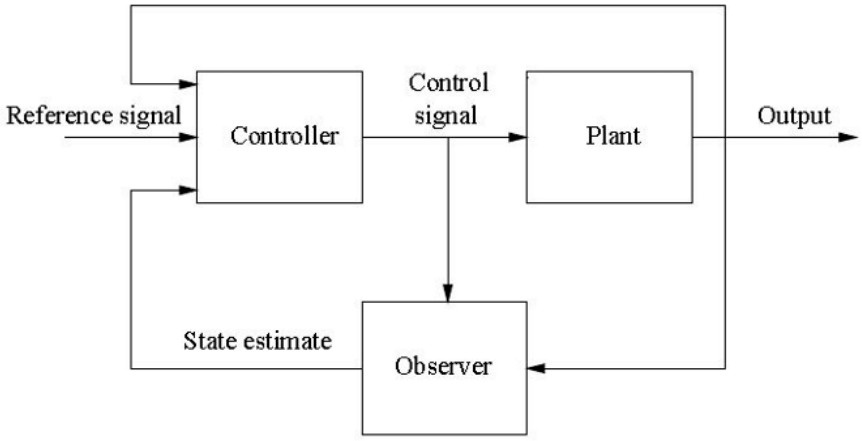

Figure 1.6. A Controller-Observer scheme

1.3.3 Event-triggered control

The common approach of a control system for a MRS is based on a periodic exchange of information among the agents and on a periodic control. In a time-triggered system, all communications and processing are initiated at a scheduled

points in time; the main advantage of this approach is the predictability. However, real systems often are not able to acquire the samples at an exact frequency. Therefore, the use of aperiodic sampling has been subject to many researches in the control community. In **event-triggered** control, the idea of a periodic sampling is replaced by aperiodic one; as it is useless to monitor an already stable system's state, it would be better check the state rarely, to see if it behaves in the right way. The goal of the research is to determine in a precise way when the control signals need to be updated to improve the efficiency and, at the same time, to guarantee control performance. The application of an **event-triggered** function allows to lower extensively the quantity of information exchanged between the agents, facilitating the communication especially in some real scenarios, where the robots can share only one wireless channel. This type of control, could have also some disadvantages, mostly linked to Zeno phenomenon (where a system makes an infinite number of jumps in a finite amount of time), in the case the control is not suitably designed.

1.4 Background

This thesis deals with the simulation of a centroid and formation control, based on a distributed controller-observer scheme for a team of mobile robots. The control scheme is inherited from a research paper [1] and is extended to the case of event-triggered communication. The thesis analyzes a decentralized scheme, finalized to the control of the formation of a Multi-robots (MRS), following the introduction of two different event-triggered function, with the goal of decreasing the exchange of information between the robots.

1.5 Overview of the formation control and goals

The considered control scheme is composed by an observer and a controller, for each agent. The observer provides an estimate of the state of whole system using, respectively, the information collected on its own and the one received from the neighbors agents. The controller generates a control input trough its feedback control law, which is used for steering the robot to comply with the desired task. In light of that the robots are able to follow a desired path in a given formation. On the

basis of this control scheme, two different triggering condition are implemented to reduce the information exchange. One triggering condition is based on the dynamics of the estimation error and the second one based on a Lyapunov function. The goals of this thesis is to validate these triggering conditions in terms of convergence to zero of the estimation error and of the errors related to the given task. The two triggering conditions are compared. It further analyzes the behavior of the system in the case of one task at a time is active.

Chapter 2

Control scheme model

2.1 Preliminaries

2.1.1 Graph theory

Denoted by E the set of the edges of a graph and V the set of the nodes of a graph; a graph is named undirected if $(i, j) \in E$ implies $(i, j) \in E$ for any $i, j \in V$. An undirected graph is called **connected**, if there exists a path between each pair of distinct nodes. A directed graph is called **strongly connected**, if there is a directed path from each node to each other node. A node of a directed graph is called balanced if its in-degree (i.e. number of incoming edges) and its out-degree (i.e. the number of outgoing edges) are equal. A graph is called **balanced** if each node of the graph is balanced.

2.1.2 Mathematical knowledge

If A is an $m \times n$ matrix and B is a $p \times q$ matrix, then the Kronecker product $A \otimes B$ is the $mp \times nq$ block matrix:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \tag{2.1}$$

The main proprieties of Kronecker products are:

- $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$.

- $(A \otimes B)^T = A^T \otimes B^T$.
- $rank(A \otimes B) = rank(A)rank(B)$.
- $rank(A + B) = rank(A) + rank(B)$ if $B^T A = 0$

2.1.3 Consensus

The **consensus** is a fundamental notion of this control scheme. When, in multiple vehicles accord on a variable of interest, they have reached **consensus**. The consensus gives, to the vehicles belonging to the same network, well matched information about some crucial parameters for the coordination task. Usually to achieve consensus, the agents have to share a variable of interest, called **information state**, as well as a proper algorithmic methods for settling to reach consensus on the value of that variable, called the **consensus algorithm**. The idea at the basis of a consensus algorithm is to design each vehicle with similar dynamics on the information states. If the communication among vehicles is continuous, the information state update of each vehicle is modeled using a differential equation.On the other hand, if the communication is discrete, the information about the consensus is modeled using a difference equation.

2.1.4 Hurwitz stability

The Routh-Hurwitz stability criterion gives an algorithm to decide whether or not the zeros of a polynomial are all in the left half of the complex plane (such a polynomial is called at times "Hurwitz"). A Hurwitz polynomial is a fundamental requirement for a linear continuous-time invariant system to be stable (all bounded inputs produce bounded outputs). Given a real polynomial

$$p(z) = a_0 z^n + a_1 z^{n-1} + \cdots + a_{n-1} z + a_n \tag{2.2}$$

the $n \times n$ matrix of the coefficients is

$$A = \begin{bmatrix} a_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ a_3 & a_2 & a_1 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ a_5 & a_4 & a_3 & a_2 & a_1 & 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & a_n \end{bmatrix} \tag{2.3}$$

A square matrix A is called a **Hurwitz** (*stable*) **matrix** if each of its eigenvalue of A has a stricly negative real part, as

$$Re[\lambda_i] < 0 \tag{2.4}$$

for each eigenvalue λ_i. A is called a **stability matrix**, because the differential equation

$$\dot{x} = Ax \tag{2.5}$$

is asymptotically stable, $x(t) \longrightarrow 0$ as $t \longrightarrow \infty$.

2.1.5 Lyapunov criterion for the analysis of stability

Given a non-linear system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \tag{2.6}$$

having $\mathbf{x} \in \mathbb{R}^n$ as state vector and $\bar{x}$ as an equilibrium point . Let $V(x)$ be a scalar function of the state vector, continuous together with its derivative $\dot{V}(\mathbf{x}, \dot{\mathbf{x}})$. $V(\mathbf{x})$ is a **Lyapunov function** if

- Is positive definite in $\bar{\mathbf{x}}$ ($\Longrightarrow V(\mathbf{x}) > \mathbf{0}, \forall \mathbf{x} \neq \bar{\mathbf{x}}$ and $V(\mathbf{x}) = \mathbf{0}, \mathbf{x} = \bar{\mathbf{x}}$)
- Its derivative $\dot{V}(\mathbf{x}, \dot{\mathbf{x}})$ is negative definite in $\bar{\mathbf{x}}$

If the Lyapunov function satisfies these conditions, then the equilibrium point $\bar{\mathbf{x}}$ is asymptotically stable. If $V(\mathbf{x})$ in addition to being a Lyapunov function, is also radially unbounded ($\Longrightarrow V(\mathbf{x}) \longrightarrow \infty, \|\mathbf{x}\| \longrightarrow \infty$), then the equilibrium point ($\bar{\mathbf{x}}$) (that is unique in this case) is **globally** asymptotically stable. The existence of a Lyapunov function is a *sufficient* condition for the asymptotic stability of the equilibrium. If the derivative of a definite positive function $V(\mathbf{x})$ is a not negative definite, it is not possible to to draw any conclusion on the stability of the equilibrium , but it is necessary to seek for another function $V(\mathbf{x})$. The Lyapunov method can be interpreted as a description of an autonomous dynamical systems based on a suitable **energy definition** (positive, by definition). If the derivative of such a function is negative for any value of the state variables, except for the equilibrium point, then the energy decreases along any trajectory of the system, until it attains the minimum, corresponding to the equilibrium point, which results in this way asymptotically stable. If $V(\mathbf{x})$ is positive definite, but its derivative

is only negative semi-definite($\Longrightarrow \dot{V}(\mathbf{x}, \dot{\mathbf{x}}) \leq \mathbf{0}$), the global asymptotical stability of the equilibrium is guaranteed if $\dot{V}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0}$ only in the equilibrium point (**LaSalle – Krasowski theorem**). In the case of linear, time invariant system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} \tag{2.7}$$

necessary and sufficient condition for the asymptotic stability of the system (not only of a single equilibrium) is that for each matrix $\mathbf{Q}$ with dimensions $n \times n$, symmetric and positive definite, there exists a matrix $\mathbf{P}$ with dimension $n \times n$, symmetric and positive definite, solution to the **Lyapunov equation**

$$\mathbf{A}^\mathbf{T}\mathbf{P} + \mathbf{P}\mathbf{A} = -\mathbf{Q} \tag{2.8}$$

If $\mathbf{P}$ exists, the function $V(\mathbf{x}) = \mathbf{x}^\mathbf{T}\mathbf{P}\mathbf{x}$ is a Lyapunov function and its derivative is $\dot{V}(\mathbf{x}) = -\mathbf{x}^\mathbf{T}\mathbf{Q}\mathbf{x}$

2.1.6 Zeno phenomenon

A Zeno phenomenon is a very important aspect to consider in the design of a multi-robot control system. The Zeno phenomenon happens when a system makes an infinite number of transitions in a finite amount of time. This phenomenon can be defined as an hybrid phenomenon because it requires interchange of continuous dynamics (the flow of time) and discrete dynamics (the discrete transitions of a system). The Zeno phenomenon is an important factor when modeling, analyzing, controlling, and simulating hybrid systems. Typically it arises due to model simplification by the designer of a control system. The presence of Zeno execution may compromise the validity of a lot of techniques used for the analysis of a control system. These techniques (as Lyapunov, model checking and deductive methods) are based on the analysis of the system's behavior along the execution. Despite the mathematical correctness of these techniques, they don't guarantee about the evolution of the system beyond the limit of the transition times. If the limit is finite the evolution of the system is an important part of the process being modeled. Some theoretical methods exist for detecting and eliminating the Zeno phenomenon. For the simulations it's possible to detect Zeno phenomenon instantly, and avoid it by defining the behavior of the system beyond the limit time of the discrete transitions.

2.2 System dynamics

Consider a group of N mobile agents, where all the agents are modeled with a single integrator dynamics. The motion of the group is assumed in two dimensions, so n is equal to two. For agent $i(i = 1...N)$, state and control vectors are $\boldsymbol{x}_i = [x_{i1}, x_{i2}]^T$ and $\boldsymbol{u}_i = [u_{i1}, u_{i2}]^T$. Single integrator or first order dynamics for agent i

$$\dot{\boldsymbol{x}}_i = \boldsymbol{u}_i\,, \quad i = 1, \ldots, N\,. \tag{2.9}$$

The collective state and control vectors can be represented as $\boldsymbol{x} = [\boldsymbol{x}_1^T, ..., \boldsymbol{x}_N^T]^T$ and $\boldsymbol{u} = [\boldsymbol{u}_1^T, ..., \boldsymbol{u}_N^T]^T$, with $\boldsymbol{x}, \boldsymbol{u} \in \mathbb{R}^{nN}$. The collective dynamics of the group is

$$\dot{\boldsymbol{x}} = \boldsymbol{u}\,. \tag{2.10}$$

2.2.1 Graph Representations and Laplacian Matrix

The communication between the N agents can be represented by a fixed graph $G(E, V)$. V is the set of indexes labeling the N vertices (nodes) and $E = V \times V$ is the set of arcs connecting the nodes. Defined the communication graph, an adjacency matrix $(A = N \times N)$ is obtained

$$\boldsymbol{A} = a_{ij} : a_{ii} = 0, a_{ij} = \begin{cases} 1 & if\ (j,i) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

In this matrix the element a_{ij} is equal to 1 if the node j can send information to the node i but not necessarily conversely, otherwise a_{ij} is equal to 0. The element a_{ii} is always equal to 0. From the adjacency matrix the Laplacian matrix is derived. Denoted by L: a real symmetric N x N matrix, that may also be considered as a kind of augmented vertex-adjacency matrix. It is defined as the following difference:

$$\boldsymbol{L} = \mathbf{D} - \mathbf{A} \tag{2.12}$$

Where **D** is the degree matrix and **A** is the adjacency matrix of the system. The elements of L are given by

$$\boldsymbol{L} = l_{ij} :\ l_{ii} = \sum_{j=1, j\neq i}^{N} a_{ij},\ l_{ij} = -a_{ij},\ i \neq j \tag{2.13}$$

Moreover, it is assumed that the ith robot receives information only from a reduced set of nodes (called its neighbors) $\mathcal{N}_i = j \in V : (j,i) \in E$, and it does not know the overall communication graph.

2.2.2 Control objective

The objective of the control is to make the centroid of the agents' team follow a desired trajectory and the relative formation follows the desired reference. To this aim, two tasks need to be defined:

- The **centroid** (barycenter) of the system: The easiest way to implement this type of task is to obtain the mean position of the group of robots:

$$\boldsymbol{\sigma}_1(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i \ = \boldsymbol{J}_1 \boldsymbol{x}, \tag{2.14}$$

  where $\boldsymbol{J}_1 \in \mathbb{R}^{n \times Nn}$ is the Jacobian of the task, such that $\dot{\boldsymbol{\sigma}}_1 = \boldsymbol{J}_1 \dot{\boldsymbol{x}}$. $\boldsymbol{J}_1$ is obtained as

$$\boldsymbol{J}_1 = \frac{1}{N}\left(1_N^T \otimes \boldsymbol{I}_n\right), \tag{2.15}$$

  where $\boldsymbol{I}_n$ is the $(n \times n)$ identity matrix and 1_N is $(N \times 1)$ vector of ones.

- The **formation** of the system, can be defined as a given series of relative displacement between the agents:

$$\boldsymbol{\sigma}_2(\boldsymbol{x}) = \left[(\boldsymbol{x}_2 - \boldsymbol{x}_1)^{\mathrm{T}} (\boldsymbol{x}_3 - \boldsymbol{x}_2)^{\mathrm{T}} \dots \ (\boldsymbol{x}_N - \boldsymbol{x}_{N-1})^{\mathrm{T}}\right]^{\mathrm{T}} = \boldsymbol{J}_2 \boldsymbol{x}, \tag{2.16}$$

  where $\boldsymbol{J}_2 \in \mathbb{R}^{(N-1)n \times Nn}$ is the Jacobian of the task, and $\dot{\boldsymbol{\sigma}}_2 = \boldsymbol{J}_2 \dot{\boldsymbol{x}}$. $\boldsymbol{J}_2$ is defined as:

$$\boldsymbol{J}_2 = \begin{bmatrix} -\boldsymbol{I}_n & \boldsymbol{I}_n & \boldsymbol{O}_n & \cdots & \boldsymbol{O}_n & \boldsymbol{O}_n \\ \boldsymbol{O}_n & -\boldsymbol{I}_n & \boldsymbol{I}_n & \cdots & \boldsymbol{O}_n & \boldsymbol{O}_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{O}_n & \boldsymbol{O}_n & \boldsymbol{O}_n & \cdots & -\boldsymbol{I}_n & \boldsymbol{I}_n \end{bmatrix}. \tag{2.17}$$

  where $\boldsymbol{I}_n$ is an identity matrix of size $(n \times n)$, while $\boldsymbol{O}_n$ is a square matrix in which all of the entries are 0 and of size $(n \times n)$. The pseudoinverse of the

Jacobian of the first task can be easy obtained

$$\boldsymbol{J}_1^\dagger = \mathbf{1}_N \otimes \boldsymbol{I}_n \tag{2.18}$$

while the pseduoinverse of the Jacobian $\mathbf{J}_2$ requires deeper examination

$$\boldsymbol{J}_2^\dagger = \begin{bmatrix} \boldsymbol{J}_{2,1}^\dagger \\ \vdots \\ \boldsymbol{J}_{2,i}^\dagger \\ \vdots \\ \boldsymbol{J}_{2,N}^\dagger \end{bmatrix}, \tag{2.19}$$

where

$$\boldsymbol{J}_{2,1}^\dagger = \begin{bmatrix} -\dfrac{(N-1)}{N}\boldsymbol{I}_n & -\dfrac{(N-2)}{N}\boldsymbol{I}_n & \dots & -\dfrac{1}{N}\boldsymbol{I}_n \end{bmatrix}, \tag{2.20}$$

$$\boldsymbol{J}_{2,N}^\dagger = \begin{bmatrix} \dfrac{1}{N}\boldsymbol{I}_n & \dfrac{2}{N}\boldsymbol{I}_n & \dots & \dfrac{(N-1)}{N}\boldsymbol{I}_n \end{bmatrix}, \tag{2.21}$$

for $i = 2, \dots, N-1$

$$\boldsymbol{J}_{2,i}^\dagger = \begin{bmatrix} \dfrac{1}{N}\boldsymbol{I}_n \dots & \dfrac{i-1}{N}\boldsymbol{I}_n & -\dfrac{N-i}{N}\boldsymbol{I}_n & \dots - \dfrac{1}{N}\boldsymbol{I}_n \end{bmatrix}. \tag{2.22}$$

The following properties hold:

$$\sum_{i=1}^{N} \boldsymbol{J}_i^\dagger = \boldsymbol{O}_{n\times(N-1)n}, \tag{2.23}$$

$$\boldsymbol{J}_2 \sum_{i=1}^{N} \boldsymbol{\Gamma}_i^{\mathrm{T}} = \boldsymbol{O}_{(N-1)n\times n}, \tag{2.24}$$

where $\boldsymbol{\Gamma}_i^{\mathrm{T}} = \begin{bmatrix} \boldsymbol{O}_n & \cdots \boldsymbol{I}_n \cdots & \boldsymbol{O}_n \end{bmatrix}$ is a $(n \times Nn)$ matrix, where $\boldsymbol{I}_n$ is placed at the location of the *ith* node. On the basis of the proprieties listed above, it's possible to verify that the two tasks satisfy the following orthogonality conditions:

$$\boldsymbol{J}_1 \boldsymbol{J}_2^{\mathrm{T}} = \boldsymbol{O}_{n\times(N-1)n}, \quad \boldsymbol{J}_1 \boldsymbol{J}_2^{\mathrm{T}} = \boldsymbol{O}_{(N-1)n\times n}, \tag{2.25}$$

$$\boldsymbol{J}_1\boldsymbol{J}_2^{\dagger} = \boldsymbol{O}_{n\times(N-1)n}, \quad \boldsymbol{J}_2\boldsymbol{J}_1^{\dagger} = \boldsymbol{O}_{(N-1)n\times n}\,. \tag{2.26}$$

2.2.3 Trajectory planning

After the definition of the two tasks, the next step is to plan the trajectory that the multi-robot system has to follow during the execution of the algorithm. The objective of the trajectory planning is to create the reference input to the motion system of the robots. The user describes the desired trajectory through a number of parameters. Planning consists of generating a temporal sequence of values assumed by a given function, usually polynomial, chosen as a primitive to interpolate the desired trajectory. In the point-to-point motion, the agents should move from an initial to a final position, in an assigned time t_f. The algorithm has to generate a trajectory and, given some general characteristics, should optimize the displacement between two positions. To define the path, a polynomial function of third degree can be used:

- the position is obtained as $p(t) = a_3t^3 + a_2t^2 + a_1t + a_0$
- the velocity is obtained from the position as $\dot{p}(t) = 3a_3t^2 + 2a_2t + a_1$
- finally the acceleration is the derivative of the velocity $\ddot{p}(t) = 6a_3t + 2a_2$

$$\begin{cases} p(t_i) = a_3t_i^3 + a_2t_i^2 + a_1t_i + a_0 \\ p(t_f) = a_3t_f^3 + a_2t_f^2 + a_1t_f + a_0 \\ \dot{p}(t_i) = 3a_3t_i^2 + 2a_2t_i + a_1 \\ \dot{p}(t_f) = 3a_3t_f^2 + 2a_2t_f + a_1 \\ t_i = 0, \\ p(t_i) = p_i, \\ p(t_f) = p_f, \\ \dot{p}(t_i) = \dot{p}_i, \\ \dot{p}(t_f) = \dot{p}_f \end{cases} \longrightarrow \begin{cases} p_i = a_0 \\ p_f = a_3t_f^3 + a_2t_f^2 + a_1t_f + a_0 \\ \dot{p}_i = a_1 \\ \dot{p}_f = 3a_3t_f^2 + 2a_2t_f + a_1 \end{cases} \tag{2.27}$$

Solving this system is possible to obtain the coefficients of the polynomial function and its derivative, which give the input value for the robots.

The observer and the controller can use only local information, its own state and input,and information received from its neighboring robots, $\mathcal{N}_i$. Furthermore each robot knows the desired value of the tasks function and their first time derivative.

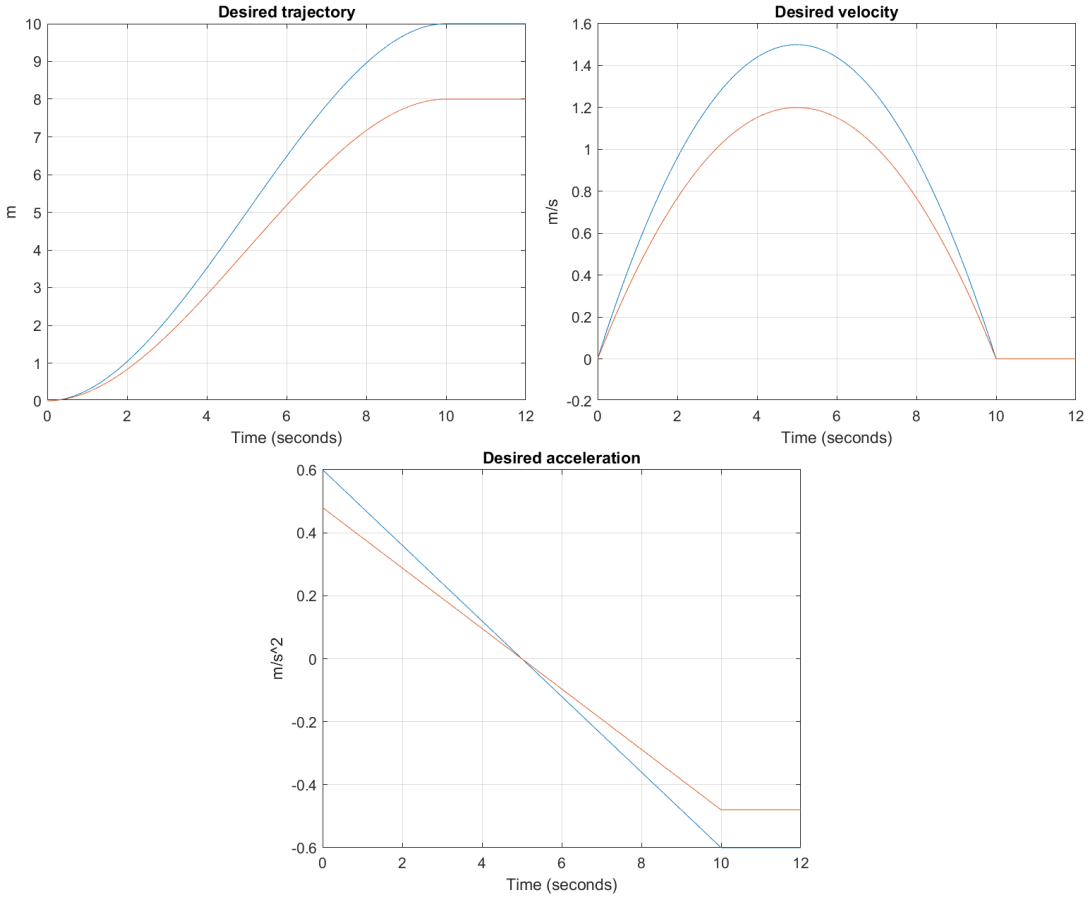

Figure 2.1. The desired trajectory, the desired velocity and the acceleration

2.2.4 State observer design

One of the key concept of the considered control scheme is the design, for each robot, of a state observer providing an estimate of the whole state of the system ${}^{i}\hat{\boldsymbol{x}}$ (i.e. the position of each robot), asymptotically convergent to the collective state $\boldsymbol{x}$, as $t \longrightarrow \infty$.

Let $\boldsymbol{\Pi}_i$ be the matrix $(Nn \times Nn)$ $\boldsymbol{\Pi}_i = \boldsymbol{\Gamma}_i \boldsymbol{\Gamma}_i^{\mathrm{T}}$. The following equality holds: $\sum_{i=1}^{N} \boldsymbol{\Pi}_i = \boldsymbol{I}_{Nn}$. The estimate of the collective state is computed by the ith robot

$(i = 1, ..., N)$ by the observer

$$ {}^{i}\dot{\hat{\boldsymbol{x}}}(t) = k_o \left(\boldsymbol{c}_i(t) + \boldsymbol{\Pi}_i {}^{i}\tilde{\boldsymbol{x}}(t) \right) + {}^{i}\hat{\boldsymbol{u}}(t, {}^{i}\hat{\boldsymbol{x}}), \tag{2.28} $$

where $k_o > 0$ is the gain of the observer, to be properly selected. The estimation error is given by the difference between the real state of the system and its estimation made by the ith agent,

$$ {}^{i}\tilde{\boldsymbol{x}}(t) = \boldsymbol{x}(t) - {}^{i}\hat{\boldsymbol{x}}(t)\,, \tag{2.29} $$

and the consensus term is computed as the sum of the differences of the estimate of the state computed by the neighbours of the ith robot and the estimate of the state made by itself

$$ \boldsymbol{c}_i(t) = \sum_{j \in \mathcal{N}_i} {}^{j}\hat{\boldsymbol{x}}(t) - {}^{i}\hat{\boldsymbol{x}}(t)\,. \tag{2.30} $$

The estimation of the collective control input available to the ith robot is given by

$$ {}^{i}\hat{\boldsymbol{u}}(t, {}^{i}\hat{\boldsymbol{x}}) = \begin{bmatrix} \boldsymbol{u}_1(t, {}^{i}\hat{\boldsymbol{x}}) \\ \boldsymbol{u}_2(t, {}^{i}\hat{\boldsymbol{x}}) \\ \vdots \\ \boldsymbol{u}_N(t, {}^{i}\hat{\boldsymbol{x}}) \end{bmatrix} \in \mathbb{R}^{Nn}\,. \tag{2.31} $$

The exact expression of $\boldsymbol{u}_1(t, {}^{i}\hat{\boldsymbol{x}})$ will be detailed later, since it depends on the control law. To implement this type of observer, the robot uses only local information, since $\boldsymbol{\Pi}_i$ select only the ith component of the collective state $\boldsymbol{x}$, i.e. robot's own state. The state estimates can be stacked into the vector $\hat{\boldsymbol{x}}^* = [{}^{1}\hat{\boldsymbol{x}}^T, ..., {}^{N}\hat{\boldsymbol{x}}^T]^T \in \mathbb{R}^{N^2 n}$. A stacked vector of the estimation error can be defined as follow:

$$ \tilde{\boldsymbol{x}}^\star = \begin{bmatrix} {}^{1}\tilde{\boldsymbol{x}} \\ {}^{2}\tilde{\boldsymbol{x}} \\ \vdots \\ {}^{N}\tilde{\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} - {}^{1}\hat{\boldsymbol{x}} \\ \boldsymbol{x} - {}^{2}\hat{\boldsymbol{x}} \\ \vdots \\ \boldsymbol{x} - {}^{N}\hat{\boldsymbol{x}} \end{bmatrix} = \mathbf{1}_N \otimes \boldsymbol{x} - \hat{\boldsymbol{x}}^\star, \tag{2.32} $$

The collective estimation of the state is given by

$$ \dot{\hat{\boldsymbol{x}}}^\star = -k_o \boldsymbol{L}^\star \hat{\boldsymbol{x}}^\star + k_o {\boldsymbol{\Pi}^\star}^{\mathrm{T}} \tilde{\boldsymbol{x}}^\star + \hat{\boldsymbol{u}}^\star, \tag{2.33} $$

where

$$\boldsymbol{L}^{\star} = \boldsymbol{L} \otimes \boldsymbol{I}_{Nn}, \quad \boldsymbol{\Pi}^{\star} = \text{diag}\left\{\boldsymbol{\Pi}_1, \ldots, \boldsymbol{\Pi}_N\right\} \tag{2.34}$$

and

$$\hat{\boldsymbol{u}}^{\star}(t, \hat{\boldsymbol{x}}^{\star}) = \begin{bmatrix} {}^1\hat{\boldsymbol{u}}(t, {}^1\hat{\boldsymbol{x}}) \\ {}^2\hat{\boldsymbol{u}}(t, {}^2\hat{\boldsymbol{x}}) \\ \vdots \\ {}^N\hat{\boldsymbol{u}}(t, {}^N\hat{\boldsymbol{x}}) \end{bmatrix} \in \mathbb{R}^{N^2 n}. \tag{2.35}$$

2.2.5 Control law

Centralized architecture

In the case of a centralized architecture, a suitable solution for the system could be found via the control law

$$\boldsymbol{u}_{cent}(t, \boldsymbol{x}) = s_1 \boldsymbol{u}_{1,cent}(t, \boldsymbol{x}) + s_2 \boldsymbol{u}_{2,cent}(t, \boldsymbol{x}), \tag{2.36}$$

In this thesis is considered the case when only one of the two tasks is active, this is possible thanks to two binary variables s_1 and s_2. Each of the two variables is equal to 1 if the corresponding task is active, otherwise is equal to 0. The control law for each task is

$$\boldsymbol{u}_{l,cent}(t, \boldsymbol{x}) = \mathbf{J}_{\mathbf{l}}^{\dagger}(\dot{\boldsymbol{\sigma}}_{\mathbf{l,d}}(\mathbf{t}) + \mathbf{k}_{\mathbf{l,c}}\left(\boldsymbol{\sigma}_{\mathbf{l,d}}(\mathbf{t}) - \boldsymbol{\sigma}_{\mathbf{l}}(\boldsymbol{x})\right), \tag{2.37}$$

where $l = 1{,}2$ and $k_{l,c} > 0$ are scalar gains. Thanks to the orthogonality of the two tasks, the tracking error dynamics for both the tasks is given by ($l = 1{,}2$)

$$\tilde{\sigma_{\mathbf{l}}} = -\mathbf{k}_{\mathbf{l,c}}\tilde{\sigma_{\mathbf{l}}} \tag{2.38}$$

which ensure exponential convergence to zero of the tracking errors

$$\tilde{\sigma_{\mathbf{l}}} = \sigma_{\mathbf{l,d}} - \sigma_{\mathbf{l}} \tag{2.39}$$

In this case all the robots communicate with the central unit, so the control law is computed by measuring the state of an agent.

2.2.6 Decentralized architecture

In the case of decentralized architecture, the agents doesn't know each other state, but the control input of the *ith* robot can be computed using the estimate of the system state according to the following law:

$$\boldsymbol{u}_i(t, {}^i\hat{\boldsymbol{x}}) = s_1\boldsymbol{u}_{i,1}(t, {}^i\hat{\boldsymbol{x}}) + s_2\boldsymbol{u}_{i,2}(t, {}^i\hat{\boldsymbol{x}}), \tag{2.40}$$

with

$$\boldsymbol{u}_{i,1}(t, {}^i\hat{\boldsymbol{x}}) = \dot{\boldsymbol{\sigma}}_{1,d}(t) + k_{1,c}\left(\boldsymbol{\sigma}_{1,d}(t) - \boldsymbol{\sigma}_1({}^i\hat{\boldsymbol{x}})\right), \tag{2.41}$$

and

$$\boldsymbol{u}_{i,2}(t, {}^i\hat{\boldsymbol{x}}) = \boldsymbol{J}_{2,i}^{\dagger}\left(\dot{\boldsymbol{\sigma}}_{2,d}(t) + k_{2,c}\big(\boldsymbol{\sigma}_{2,d}(t) - \boldsymbol{\sigma}_2({}^i\hat{\boldsymbol{x}})\big)\right), \tag{2.42}$$

The input estimate in equation (2.31), used by the observer (2.28), becomes ($j = 1, ..., N$)

$$\boldsymbol{u}_j(t, {}^i\hat{\boldsymbol{x}}) = \dot{\boldsymbol{\sigma}}_{1,d}(t) + k_{1,c}\left(\boldsymbol{\sigma}_{1,d}(t) - \boldsymbol{\sigma}_1({}^i\hat{\boldsymbol{x}})\right) + \boldsymbol{J}_{2,j}^{\dagger}\left(\dot{\boldsymbol{\sigma}}_{2,d}(t) + k_{2,c}\big(\boldsymbol{\sigma}_{2,d}(t) - \boldsymbol{\sigma}_2({}^i\hat{\boldsymbol{x}})\big)\right), \tag{2.43}$$

where $\boldsymbol{J}_{2,j}^{\dagger}$ comes from a block partition of $\boldsymbol{J}_2^{\dagger}$

$$\boldsymbol{J}_2^{\dagger} = \begin{bmatrix} \boldsymbol{J}_{2,1}^{\dagger} \\ \vdots \\ \boldsymbol{J}_{2,i}^{\dagger} \\ \vdots \\ \boldsymbol{J}_{2,N}^{\dagger} \end{bmatrix}, \tag{2.44}$$

2.2.7 Dynamics of the estimation error

The collective estimation of the state is given by (2.33), the resulting dynamics of the estimation error is:

$$\boldsymbol{u}_i(t, {}^i\hat{\boldsymbol{x}}) = s_1\boldsymbol{u}_{i,1}(t, {}^i\hat{\boldsymbol{x}}) + s_2\boldsymbol{u}_{i,2}(t, {}^i\hat{\boldsymbol{x}}), \tag{2.45}$$

$$\dot{\tilde{\boldsymbol{x}}}^{\star} = k_o\tilde{\boldsymbol{L}}\tilde{\boldsymbol{x}}^{\star} + \mathbf{1}_{\mathbf{N}} \otimes \boldsymbol{u} - \hat{\boldsymbol{u}}^{\star}, \tag{2.46}$$

with $\tilde{\boldsymbol{L}} = -(\boldsymbol{L}^\star + \boldsymbol{\Pi}^\star)$. The i-th block of $\mathbf{1}_N \otimes \boldsymbol{u} - \hat{\boldsymbol{u}}^\star$ is

$$\boldsymbol{u} - {}^i\hat{\boldsymbol{u}}(t, {}^i\hat{\boldsymbol{x}}) = \begin{bmatrix} \boldsymbol{u}_1(t, {}^1\hat{\boldsymbol{x}}) - \hat{\boldsymbol{u}}_1(t, {}^i\hat{\boldsymbol{x}}) \\ \boldsymbol{u}_2(t, {}^2\hat{\boldsymbol{x}}) - \hat{\boldsymbol{u}}_2(t, {}^i\hat{\boldsymbol{x}}) \\ \vdots \\ \boldsymbol{u}_N(t, {}^N\hat{\boldsymbol{x}}) - \hat{\boldsymbol{u}}_N(t, {}^i\hat{\boldsymbol{x}}) \end{bmatrix}, \tag{2.47}$$

which in turn is composed by N blocks, composed by the difference between the control input of the j-th robot and the estimated control input of the j-th robot based on the estimation of the system made by the i-th robot, having the following expression ($j = 1, \ldots, N$)

$$\begin{aligned} \boldsymbol{u}_j(t, {}^j\hat{\boldsymbol{x}}) - \hat{\boldsymbol{u}}_j(t, {}^i\hat{\boldsymbol{x}}) &= \left(s_1 k_{1,c} \boldsymbol{J}_1 + s_2 k_{2,c} \boldsymbol{J}_{2,j}^\dagger \boldsymbol{J}_2\right) \left({}^i\hat{\boldsymbol{x}} - {}^j\hat{\boldsymbol{x}}\right) \\ &= \boldsymbol{K}_{c,j}^{\mathrm{T}} \left({}^i\hat{\boldsymbol{x}} - {}^j\hat{\boldsymbol{x}}\right) \\ &= -\boldsymbol{K}_{c,j}^{\mathrm{T}} \left({}^i\tilde{\boldsymbol{x}} - {}^j\tilde{\boldsymbol{x}}\right) \\ &= -\boldsymbol{K}_{c,j}^{\mathrm{T}} \left(\boldsymbol{\Gamma}_i^{\mathrm{T}} - \boldsymbol{\Gamma}_j^{\mathrm{T}}\right) \otimes \boldsymbol{I}_N \tilde{\boldsymbol{x}}^\star \,. \end{aligned} \tag{2.48}$$

where the control input is expressed as a function of the difference between the estimate of the state computed by the i-th agent and the one computed by the j-th agent. The term $(s_1 k_{1,c} \boldsymbol{J}_1 + s_2 k_{2,c} \boldsymbol{J}_{2,j}^\dagger \boldsymbol{J}_2)$ can be grouped in the the term $\boldsymbol{K}_{c,j}$. Therefore, the collective difference between the control input $\mathbf{u}$ and the one estimated by the i-th robot can be defined as

$$\boldsymbol{u} - {}^i\hat{\boldsymbol{u}}(t, {}^i\hat{\boldsymbol{x}}) = -\begin{bmatrix} \boldsymbol{K}_{c,1}^{\mathrm{T}} \left(\boldsymbol{\Gamma}_i - \boldsymbol{\Gamma}_1\right)^{\mathrm{T}} \otimes \boldsymbol{I}_N \\ \boldsymbol{K}_{c,2}^{\mathrm{T}} \left(\boldsymbol{\Gamma}_i - \boldsymbol{\Gamma}_2\right)^{\mathrm{T}} \otimes \boldsymbol{I}_N \\ \vdots \\ \boldsymbol{K}_{c,N}^{\mathrm{T}} \left(\boldsymbol{\Gamma}_i - \boldsymbol{\Gamma}_N\right)^{\mathrm{T}} \otimes \boldsymbol{I}_N \end{bmatrix} \tilde{\boldsymbol{x}}^\star = -\boldsymbol{K}_{c,i}^{\star^{\mathrm{T}}} \tilde{\boldsymbol{x}}^\star \,, \tag{2.49}$$

and

$$\mathbf{1}_N \otimes \boldsymbol{u} - \hat{\boldsymbol{u}}^\star = -\begin{bmatrix} \boldsymbol{K}_{c,1}^{\star^{\mathrm{T}}} \\ \boldsymbol{K}_{c,2}^{\star^{\mathrm{T}}} \\ \vdots \\ \boldsymbol{K}_{c,N}^{\star^{\mathrm{T}}} \end{bmatrix} \tilde{\boldsymbol{x}}^\star = -\boldsymbol{K}_c^\star \tilde{\boldsymbol{x}}^\star \,. \tag{2.50}$$

Established

$$\begin{aligned}\tilde{\boldsymbol{L}}_c &= -\left(k_o \boldsymbol{L}^\star + k_o \boldsymbol{\Pi}^\star + \boldsymbol{K}_c^\star\right) \\ &= -\left(k_o \tilde{\boldsymbol{L}} + \boldsymbol{K}_c^\star\right) \\ &= -\left(k_o \boldsymbol{L}^\star + \tilde{\boldsymbol{K}}_c\right) ,\end{aligned} \tag{2.51}$$

with $\tilde{\boldsymbol{K}}_c = (k_o \boldsymbol{\Pi}^\star + \boldsymbol{K}_c^\star)$ the dynamics of the estimation error (2.46) becomes

$$\dot{\tilde{\boldsymbol{x}}}^\star = \tilde{\boldsymbol{L}}_c \tilde{\boldsymbol{x}}^\star , \tag{2.52}$$

More details about the matrices $-\tilde{\boldsymbol{K}}_c$ and $\tilde{\boldsymbol{L}}_c$ are reported in the appendix. If $s_1 = s_2 = 1$, the matrix $-\boldsymbol{K}_c^*$ admits:

- Nn eigenvalues in 0
- $(N-1)n$ eigenvalues in $-k_{1,c}$
- $(N-2)Nn + n = (N-1)^2 n$ eigenvalues in $-k_{2,c}$

Therefore the matrix is Hurwitz. The eigenvalues of the matrix $-\boldsymbol{K}_c^*$ show that, if both tasks are active, the estimation error is globally and exponentially convergent to zero, even in case of lack of the consensus term ($\mathbf{L}^* = \mathbf{0}_{N^2 n}$). However, if one of the two tasks is not active ($s_1 = 0$ or $s_2 = 0$), because the matrix $-\boldsymbol{K}_c^*$ admits eigenvalues equal to zero, the estimation error converges only in presence of the consensus term ($\mathbf{L}^* \neq \mathbf{0}_{N^2 n}$). This phenomenon will be shown in the simulation results. The matrix $\tilde{\boldsymbol{L}}_c$ can be shown to be Hurwitz as well.

2.2.8 Event-triggered control based on the estimation error

The i-th agent sends its state to the neighbours at discrete time instants

$$t \in \left\{ {}^i t_0, {}^i t_1, \dots {}^i t_k, \dots \right\} , \tag{2.53}$$

similarly it receives the estimation of the state of the system from the j-th ($j \in \mathcal{N}_i$) at discrete instants

$$t \in \left\{ {}^j t_0, {}^j t_1, \dots {}^j t_k, \dots \right\} \tag{2.54}$$

With this notation, the consensus term becomes:

$$\bar{\boldsymbol{c}}_i(t) = \sum_{j \in \mathcal{N}_i} {}^{j}\hat{\boldsymbol{x}}({}^{j}t_k) - {}^{i}\hat{\boldsymbol{x}}(t) = \boldsymbol{c}_i(t) + \sum_{j \in \mathcal{N}_i} {}^{j}\hat{\boldsymbol{x}}({}^{j}t_k) - {}^{j}\hat{\boldsymbol{x}}(t)\,. \tag{2.55}$$

In this case, to the consensus explained previously, is added the sum of the difference between the estimate computed by the j-th agent at the time ${}^{j}t_k$ and the estimate computed at the present time t. Computed:

$$\boldsymbol{\delta}_i(t) = \sum_{j \in \mathcal{N}_i} {}^{j}\hat{\boldsymbol{x}}({}^{j}t_k) - {}^{j}\hat{\boldsymbol{x}}(t) = \sum_{j \in \mathcal{N}_i} {}^{j}\boldsymbol{\delta}(t)\,, \tag{2.56}$$

The estimation dynamics made by the i-th robot becomes

$${}^{i}\dot{\hat{\boldsymbol{x}}}(t) = k_o \left(\boldsymbol{c}_i(t) + \boldsymbol{\Pi}_i {}^{i}\tilde{\boldsymbol{x}}(t) + \boldsymbol{\delta}_i(t)\right) + {}^{i}\hat{\boldsymbol{u}}(t, {}^{i}\hat{\boldsymbol{x}})\,. \tag{2.57}$$

Given this definition, the collective estimation dynamics is rearranged as follow:

$$\dot{\hat{\boldsymbol{x}}}^{\star} = -k_o \boldsymbol{L}^{\star}\hat{\boldsymbol{x}}^{\star} + k_o \boldsymbol{\Pi}^{\star}\tilde{\boldsymbol{x}}^{\star} + k_o \boldsymbol{\delta}^{\star} + \hat{\boldsymbol{u}}^{\star}\,, \tag{2.58}$$

With

$$\boldsymbol{\delta}^{\star} = \begin{bmatrix} \boldsymbol{\delta}_1 \\ \boldsymbol{\delta}_2 \\ \vdots \\ \boldsymbol{\delta}_N \end{bmatrix}\,. \tag{2.59}$$

With this formulation the dynamics of the estimation error can be written as

$$\dot{\tilde{\boldsymbol{x}}}^{\star} = \tilde{\boldsymbol{L}}_c \tilde{\boldsymbol{x}}^{\star} - k_o \boldsymbol{\delta}^{\star}\,, \tag{2.60}$$

Consider the transition matrix corresponding to $\tilde{\mathbf{L}}_c$

$$\boldsymbol{\Phi}(t) = e^{\tilde{\boldsymbol{L}}_c t} \tag{2.61}$$

Given a real number $\epsilon > 0$, there exists a constant $k_M(\epsilon)$ such that

$$\|\boldsymbol{\Phi}(t)\| \leq k_M(\epsilon)\, e^{(\lambda_M + \epsilon)t}\,, \quad \forall t \geq 0\,, \tag{2.62}$$

where $\lambda_M = \max_{i=1,N^2 n} \left\{ \mathrm{Re}\left[\lambda_i(\tilde{\boldsymbol{L}}_c)\right] \right\}$ and $\lambda_i(\tilde{\boldsymbol{L}}_c)$ is the i-th eigenvalue of $\tilde{\boldsymbol{L}}_c$.

Since $\tilde{\boldsymbol{L}}_c$ is a Hurwitz matrix, it is $\lambda_M < 0$; thus, it is always possible to choose ϵ small enough to satisfy $\lambda_M + \epsilon < 0$. The time evolution of the estimation error is given by:

$$\tilde{\boldsymbol{x}}^\star(t) = \boldsymbol{\Phi}(t)\tilde{\boldsymbol{x}}^\star(0) - k_o \int_0^t \boldsymbol{\Phi}(t,s)\boldsymbol{\delta}^\star(s)ds \tag{2.63}$$

The first term is the **zero-input response** and represents how the system's state would evolve in the absence of any input. The second term is the **zero-state response** and defines the contribution of the input on the system's state evolution. From equation (2.63) follows:

$$\|\tilde{\boldsymbol{x}}^\star(t)\| \leq k_M\, e^{(\lambda_M+\epsilon)t}\, \|\tilde{\boldsymbol{x}}^\star(0)\| + k_o k_M\, e^{(\lambda_M+\epsilon)t} \int_0^t e^{-(\lambda_M+\epsilon)s}\, \|\boldsymbol{\delta}^\star(s)\|\, ds\,. \tag{2.64}$$

If it is verified the inequality

$$\|\boldsymbol{\delta}^\star(t)\| \leq \sigma\bar{\delta}\, e^{(\lambda_M+\epsilon)t}\,, \quad \forall t \geq 0\,, \tag{2.65}$$

with $\bar{\delta} > 0$ and $0 < \sigma < 1$, the collective dynamics of the estimation error becomes

$$\|\tilde{\boldsymbol{x}}^\star(t)\| \leq k_M\, e^{(\lambda_M+\epsilon)t} \left(\|\tilde{\boldsymbol{x}}^\star(t_0)\| + k_o\bar{\delta}\, t\right)\,, \tag{2.66}$$

which ensures $\|\tilde{\boldsymbol{x}}^\star(t)\| \to 0$ asymptotically. The norm of $\delta^\star$ can be written as

$$\|\boldsymbol{\delta}^\star\| \leq \sum_{i=1}^{N} \|\boldsymbol{\delta}_i\| \leq \sum_{i=1}^{N}\sum_{j\in\mathcal{N}_i} \left\|{}^j\boldsymbol{\delta}\right\| \leq \sum_{i=1}^{N}\sum_{j=1}^{N} \left\|{}^j\boldsymbol{\delta}\right\|\,, \tag{2.67}$$

Thus, condition (2.65) is implied by

$$\left\|{}^i\boldsymbol{\delta}(t)\right\| \leq \sigma\frac{\bar{\delta}}{N^2}\, e^{(\lambda_M+\epsilon)t}\,, \quad \forall t \geq t_0\,, \tag{2.68}$$

which can be easily verified by the i-th agent because it does not need information from its neighbours. $\bar{\delta} > 0$ is a very important term because it modulates the contribution of t in the (2.65): lowering it , the triggering condition becomes more restrictive. The triggering condition for the i-th agent is

$${}^it_{k+1} = \inf\left\{t > {}^it_k :\ \left\|{}^i\boldsymbol{\delta}(t)\right\| = \sigma\frac{\bar{\delta}}{N^2}\, e^{(\lambda_M+\epsilon)t}\right\}\,. \tag{2.69}$$

This condition can't be affected by the Zeno phenomenon.

2.2.9 Triggering condition based on the Lyapunov function

Another triggering condition is introduced, based on a Lyapunov function for the estimation error dynamics. It is possible to redefine $\tilde{\boldsymbol{L}}_c$ as follows

$$\tilde{\boldsymbol{L}}_c = -(\boldsymbol{L}^\star + \boldsymbol{\Pi}^\star + \frac{1}{k_o}\boldsymbol{K}_c^\star) \tag{2.70}$$

Considering the Lyapunov function

$$V_c = \tilde{\boldsymbol{x}}^{\star\mathrm{T}}\boldsymbol{P}_c\tilde{\boldsymbol{x}}^\star \tag{2.71}$$

with $\boldsymbol{P}_c$ a positive-definite symmetric matrix, the derivative of V_c becomes

$$\dot{V}_c = -k_o\tilde{\boldsymbol{x}}^{\star\mathrm{T}}\boldsymbol{Q}_c\tilde{\boldsymbol{x}}^\star - 2k_o\tilde{\boldsymbol{x}}^{\star\mathrm{T}}\boldsymbol{P}_c\boldsymbol{\delta}^\star\,, \tag{2.72}$$

where $\boldsymbol{Q}_c$ is a symmetric positive-definite matrix and $\boldsymbol{P}_c$ is the positive definite solution of the Lyapunov equation

$$\tilde{\boldsymbol{L}}_c^{\mathrm{T}}\boldsymbol{P}_c + \boldsymbol{P}_c\tilde{\boldsymbol{L}}_c = -\boldsymbol{Q}_c. \tag{2.73}$$

It is possible to find an upper bound of the derivative of V_c as

$$\dot{V}_c \leq -k_o\lambda_{Q_m}\left\|\tilde{\boldsymbol{x}}^\star\right\|^2 + 2k_o\lambda_{P_M}\left\|\tilde{\boldsymbol{x}}^\star\right\|\left\|\boldsymbol{\delta}^\star\right\|\,, \tag{2.74}$$

where λ_{Q_m} is the minimum eigenvalue of $\boldsymbol{Q}_c$ while λ_{P_M} is the largest eigenvalue of $\boldsymbol{P}_c$.

Since

$$\left\|\tilde{\boldsymbol{x}}^\star\right\|\left\|\boldsymbol{\delta}^\star\right\| \leq \frac{d}{2}\left\|\tilde{\boldsymbol{x}}^\star\right\|^2 + \frac{1}{2d}\left\|\boldsymbol{\delta}^\star\right\|^2 \tag{2.75}$$

inequality (2.74) becomes

$$\dot{V}_c \leq -k_o\left(\lambda_{Q_m} - \lambda_{P_M}d\right)\left\|\tilde{\boldsymbol{x}}^\star\right\|^2 + \frac{k_o\lambda_{P_M}}{d}\left\|\boldsymbol{\delta}^\star\right\|^2\,. \tag{2.76}$$

The pararmeter d is chosen so that the following condition is met

$$d < \frac{\lambda_{Q_m}}{\lambda_{P_M}}\,, \tag{2.77}$$

Assume that

$$\|\boldsymbol{\delta}^\star(t)\|^2 \leq \sigma \frac{(\lambda_{Q_m} - \lambda_{P_M} d)}{\lambda_{P_M}/d} \|\tilde{\boldsymbol{x}}^\star(t)\|^2 \,, \forall t. \tag{2.78}$$

If inequalities (2.77) and (2.78) are satisfied, with $0 < \sigma < 1$, the estimation error converges asymptotically to zero, since $\dot{V}_c < 0$, $\forall t$. Condition (2.78) can be checked in a centralized architecture, but not in a decentralized architecture. A condition, exploitable in a decentralized architecture is

$$\left\|{}^i\boldsymbol{\delta}\right\|^2 \leq \frac{\sigma}{N^2} \frac{(\lambda_{Q_m} - \lambda_{P_M} d)}{\lambda_{P_M}/d} \left\|{}^i\tilde{\boldsymbol{x}}_i\right\|^2 \,, \quad \forall i \in \{1, \ldots, N\}\,, \tag{2.79}$$

which can be easily verified by the i-th agent because it does not need information from its neighbours. Being

$$\|\boldsymbol{\delta}^\star\| \leq \sum_{i=1}^{N} \|\boldsymbol{\delta}_i\| \leq \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_i} \left\|{}^j\boldsymbol{\delta}\right\| \leq \sum_{i=1}^{N} \sum_{j=1}^{N} \left\|{}^j\boldsymbol{\delta}\right\| \,, \tag{2.80}$$

If (2.79) is satisfied

$$\|\boldsymbol{\delta}^\star\| \leq \sqrt{\frac{\sigma}{N^2} \frac{(\lambda_{Q_m} - \lambda_{P_M} d)}{\lambda_{P_M}/d}} \sum_{i=1}^{N} \sum_{j=1}^{N} \left\|{}^j\tilde{\boldsymbol{x}}_j\right\| \,. \tag{2.81}$$

by exploiting the inequality

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \left\|{}^j\tilde{\boldsymbol{x}}_j\right\| = N \sum_{j=1}^{N} \left\|{}^j\tilde{\boldsymbol{x}}_j\right\| \leq N \|\tilde{\boldsymbol{x}}^\star\| \,, \tag{2.82}$$

(2.81) leads to

$$\|\boldsymbol{\delta}^\star\| \leq \sqrt{\sigma \frac{(\lambda_{Q_m} - \lambda_{P_M} d)}{\lambda_{P_M}/d}} \|\tilde{\boldsymbol{x}}^\star\| \,, \tag{2.83}$$

which implies (2.78). Thus, (2.79) implies (2.78). Based on (2.79) the triggering

condition for the i-th agent can be written as

$$ {}^{i}t_{k+1} = \inf\left\{ t > {}^{i}t_k : \left\| {}^{i}\boldsymbol{\delta}(t) \right\|^2 = \mu \left\| {}^{i}\tilde{\boldsymbol{x}}_i(t) \right\|^2 \right\} , \tag{2.84} $$

with

$$ \mu = \frac{\sigma}{N^2} \frac{(\lambda_{Q_m} - \lambda_{P_M} d)}{\lambda_{P_M}/d} . \tag{2.85} $$

To avoid the Zeno phenomenon, it is necessary to add a constraint

$$ {}^{i}t_{k+1} = \inf\left\{ t > {}^{i}t_k : \left\| {}^{i}\boldsymbol{\delta}(t) \right\|^2 = \max\left\{ \mu \left\| {}^{i}\tilde{\boldsymbol{x}}_i(t) \right\|^2 , \omega_i(t) \right\} \right\} , \tag{2.86} $$

where $\omega_i(t)$ is the solution of the differential problem

$$ \dot{\omega}_i(t) = -\lambda \omega_i(t) , \quad \omega_i(t_0) > 0 , \quad \lambda > 0 , \tag{2.87} $$

which ensures $\omega_i(t) > 0$, $\forall t$ and $\omega_i(t) \to 0$.

Chapter 3

Simulations

A set of simulations have been carried out in the Matlab/Simulink environment concerning the proposed control scheme. The goal of simulating the presented control scheme is to verify its performance in different conditions and compare the triggering conditions. After simulating the system with both tasks active, only one task a time is activated to analyze the difference in the results.

All the simulations are carried out adopting a fixed step of 1 ms. The number of robots for each simulation is set equal to four. The desired trajectory of the centroid is a straight line, starting from the point [0;0]m to the point [10;8]m. The inital states of the robots are:

- Robot No. 1: [0;0]
- Robot No. 2: [0.5;1.2]
- Robot No. 3: [1;0.5]
- Robot No. 4: [0.6;0.7]

The simulations have a length of 12 seconds. The robots, during the trajectory, must maintain a square formation where each robot is a vertex of the square. Each of the vertexes is 1.5m away from the two on its side and 2.1 m away from the one diagonally opposite. The observer gain is set equal to 3 while the controller gains are set equal to 1.

3.0.1 Code

The Figure 3.1 shows the controller and the observer scheme running on each robot. The controller receives as inputs the desired trajectory, the relative formation and their derivative, while it gives as output the estimate control input of the i-th robot. The observer takes as inputs the state of the system, the estimate control input of the i-th robot, the estimate of the state of the system computed by the i-th robot and the estimate of the state made by the j-th robot. The observer gives as output the dynamics of the estimate state. The last block is an integrator which models the agent's dynamics. The Figure 3.2 shows the block responsible for updating the information exchanged among the agents. If the triggering condition is satisfied, the information exchanged is the one at the present instant, otherwise information of the previous instant is sent.

```
function [u3,y] = fcn(dsigma_1_d,sigma_1_d,dsigma_2_d,sigma_2_d,
    x_hat,J1, J2,k1, k2,s1, s2)

u3 = zeros(2,1);

sigma_1 = J1*x_hat;       %definition of the first task
sigma_2 = J2*x_hat;       %definition of the second task

y = s1*(pinv(J1)*(dsigma_1_d+k1*(sigma_1_d-sigma_1)))+ s2*(pinv(J2)
    *(dsigma_2_d+k2*(sigma_2_d-sigma_2)));     %definition of the
    decentralized control law
u3 = y(5:6);       %each agent has access only to the corresponding
    information

end
```

Listing 3.1. Code for the controller

```
function dxhat = fcn(uhat,x,xhat,A, ko,x_nei)

xh1 = x_nei(1:8,1);
xh2 = x_nei(9:16,1);
xh3 = x_nei(17:24,1);
xh4 = x_nei(25:end,1);

consensus = A(3,1)*(xh1-xhat)+A(3,2)*(xh2-xhat)+A(3,3)*(xh3-xhat)+A
    (3,4)*(xh4-xhat);              %definition of the consensus

```

```
Gam = [zeros(2) zeros(2) eye(2)  zeros(2)];

Sel = Gam'*Gam;

dxhat = ko*consensus+ko*Sel*(x-xhat)+uhat;   %estimation of the
    whole ssytem made by the i-th agent
end
```

Listing 3.2. Code for the observer

```
function [xn1,xn2,xn3,xn4,t,delta] = fcn(xhat,x_r,A,time,delta_sign
    , max_lambda_L_tilde_c, epsilon, N, sigma, mu)

persistent xnei tempo       %definition of the variables where to
    save the update estimations and the time of the simulation

if isempty(xnei)
    xnei = zeros(32,4);
    tempo = 0;
end

xn = zeros(32,4);
delta = zeros(8,4);
xtilde = zeros(8,4);

for i=1:4
    delta(:,i) = xnei(8*(i-1)+1:8*i,i)-xhat(8*(i-1)+1:8*i,1); %the
    difference between the estimation made at the current instant
    and the one made in the previous instant
    xtilde(:,i) = (x_r - xhat(8*(i-1)+1:8*i));  %estimation error
end

for i=1:4
    if norm(delta(:,i))>=((sigma/N^2)*(delta_sign)*(exp((
    max_lambda_L_tilde_c +epsilon)*time)))       %if the triggering
     condition is met,the estimation is update, otherwise the
    previous vale is assigned
        xn(:,i) = xhat;     % update of the estimation to send to
    the i-th robot
        xnei(:,i) = xhat;   % keep the estimation in the variable
    xnei
        t = time;           % update of the time variable with the
    current time
```

```
24          tempo = time;        % update of the persistent variable
    tempo
25      else
26          xn(:,i) = xnei(:,i); % the robot i-th receive the previous
    estimation
27          t = tempo;           % set the time as the previous value
28      end
29  end
```

Listing 3.3. Code for triggering condition based on the estimation error

```
1
2   for i=1:4
3       if norm(delta(:,i))^2>=mu*norm(xtilde(:,i))^2  %if the
    triggering condition is met,the estimation is update, otherwise
     the previous value is assigned
4           xn(:,i) = xhat;      % update of the estimation to send to
    the i-th robot
5           xnei(:,i) = xhat;    % keep the estimation in the variable
    xnei
6           t = time;            % update of the time variable with the
     current time
7           tempo = time;        % update of the persistent variable
    tempo
8       else
9           xn(:,i) = xnei(:,i); % the robot i-th receive the previous
     estimation
10          t = tempo;           % set the time as the previous value
11      end
12  end
```

Listing 3.4. Code for triggering condition based on the Lyapunov function

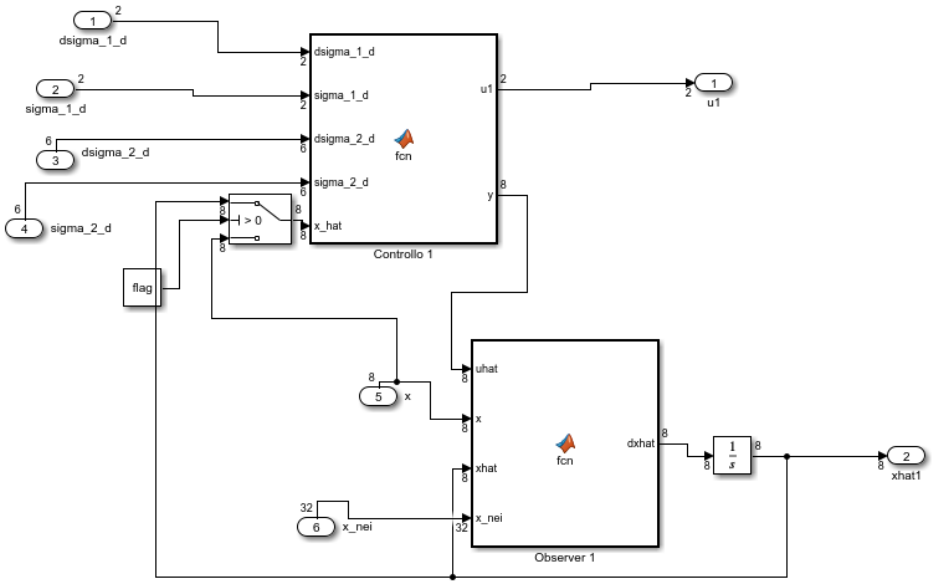

Figure 3.1. Controller and observer scheme running on each robot

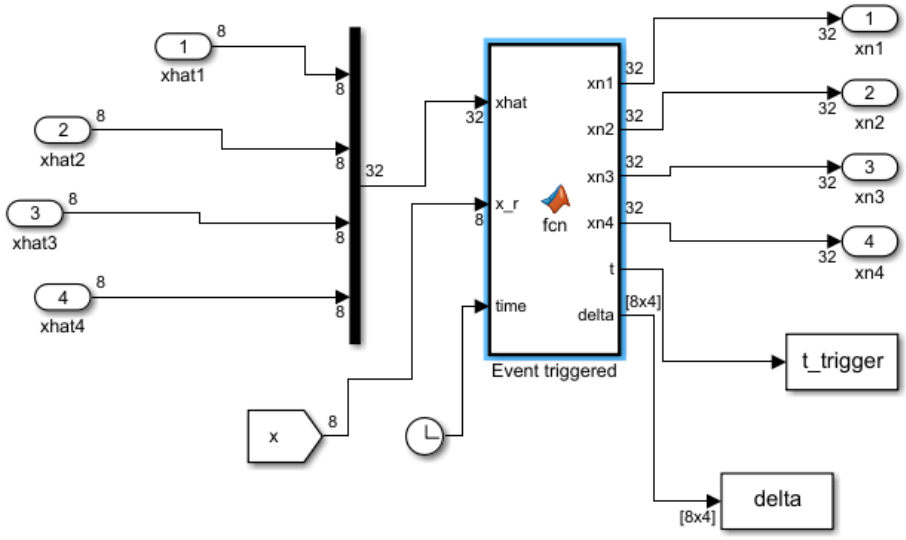

Figure 3.2. The block responsible for updating the information exchanged among the agents

3.0.2 Simulations without the triggering condition

The simulation about the basic control scheme (without the triggering conditions), shows exponential convergence to zero of both the estimation errors and the tracking error relative to the two tasks. Without triggering condition, the agents update the information to exchange with each other every instant. The Figure 3.5 shows in detail the update of the information each step of the simulation, on the abscissa there is the time of the simulation, while on the ordinate there is the information's time (the instant in which the information exchanged is taken). This chart is very important because it shows when the information is updated, if the same ordinate lasts for more than one simulation step (1 ms), it means that the triggering condition is not satisfied and the agent sends the information taken in the previous instant to its neighbors. In the Figure 3.6 can be seen the path travelled by the robots, the agents have a transient behavior at the start, due to difference between the path they must follow and their initial position.

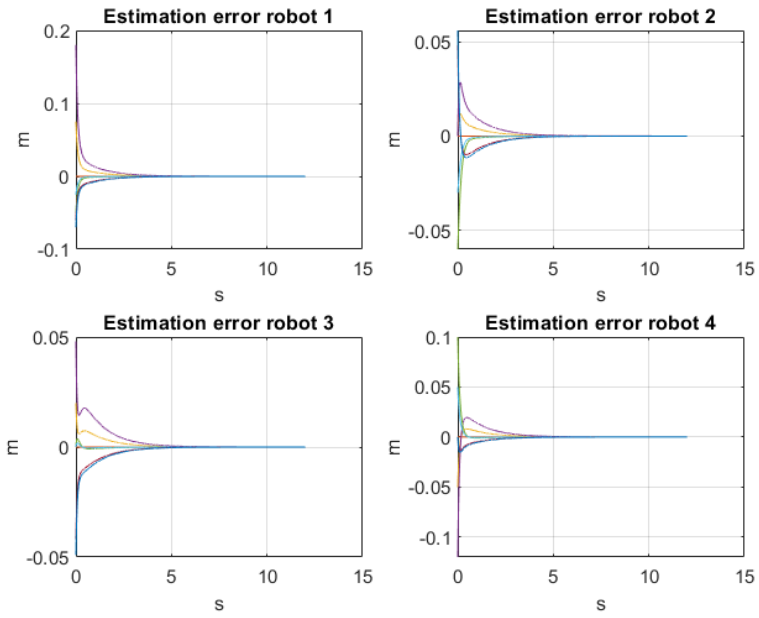

Figure 3.3. Estimation errors relative to the four robots without triggering condition

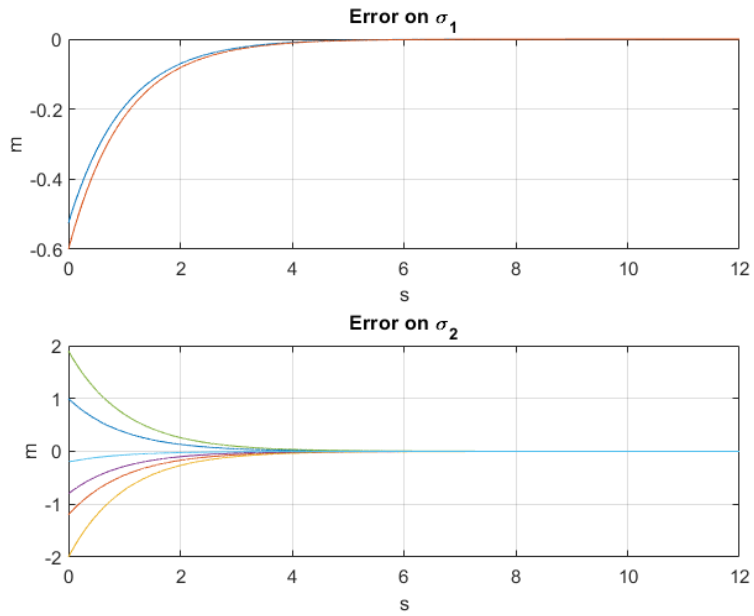

Figure 3.4. Centroid and formation tracking errors without triggering condition

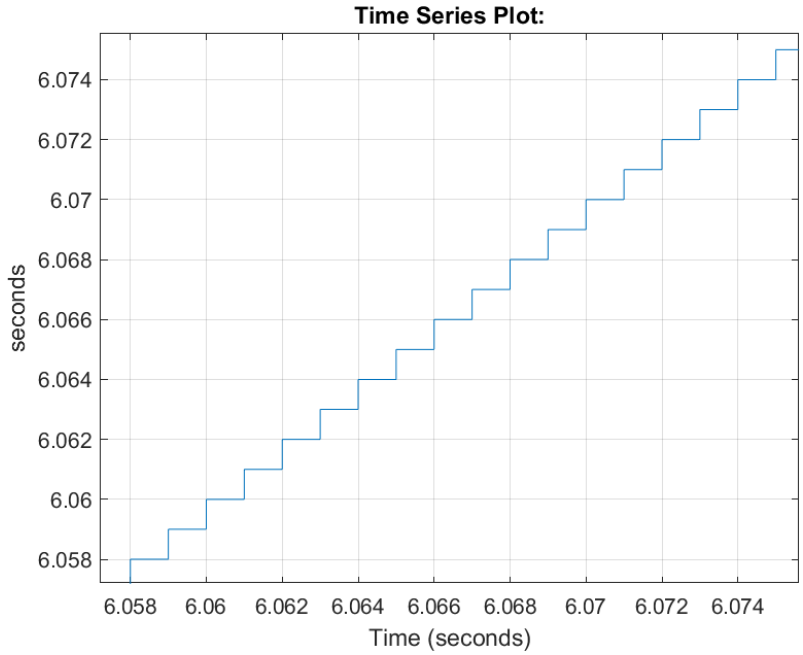

Figure 3.5. Time update without triggering condition

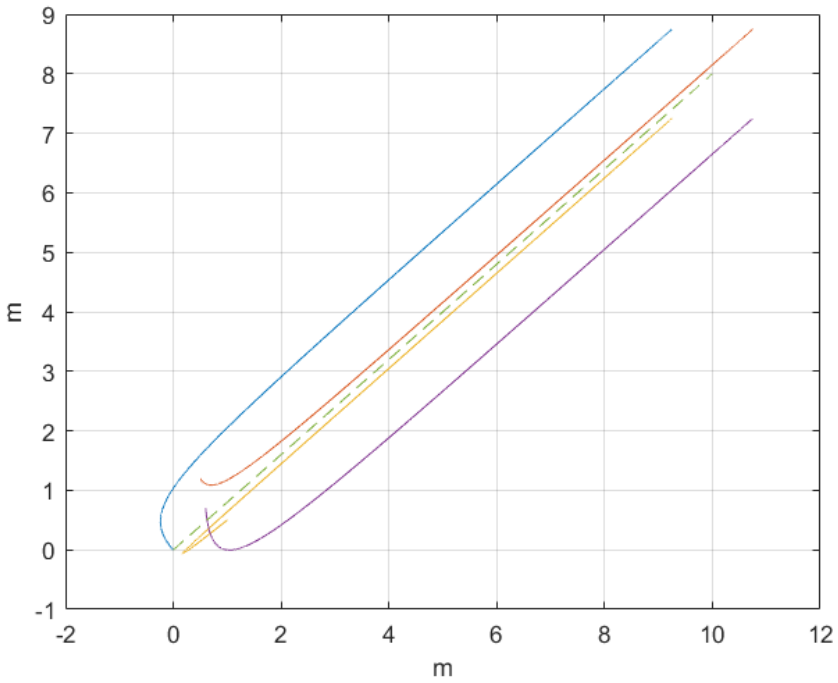

Figure 3.6. Path travelled by the robots

Simulations without the triggering condition with only σ_1 active

In this case only one task is active: the centroid of the system. Without triggering condition, despite the lack of one task, the estimation error converge exponentially and globally to zero Figure 3.7 since the communication is always active, and the estimation exchanged among the robots is updated every instant. The centroid tracking error converge as well Figure 3.8.

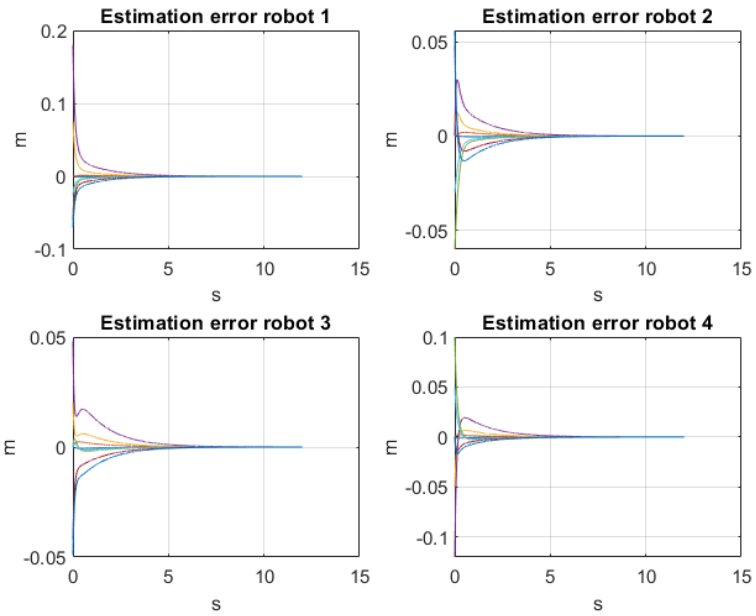

Figure 3.7. Estimation errors relative to the four robots without the triggering condition with only σ_1 active

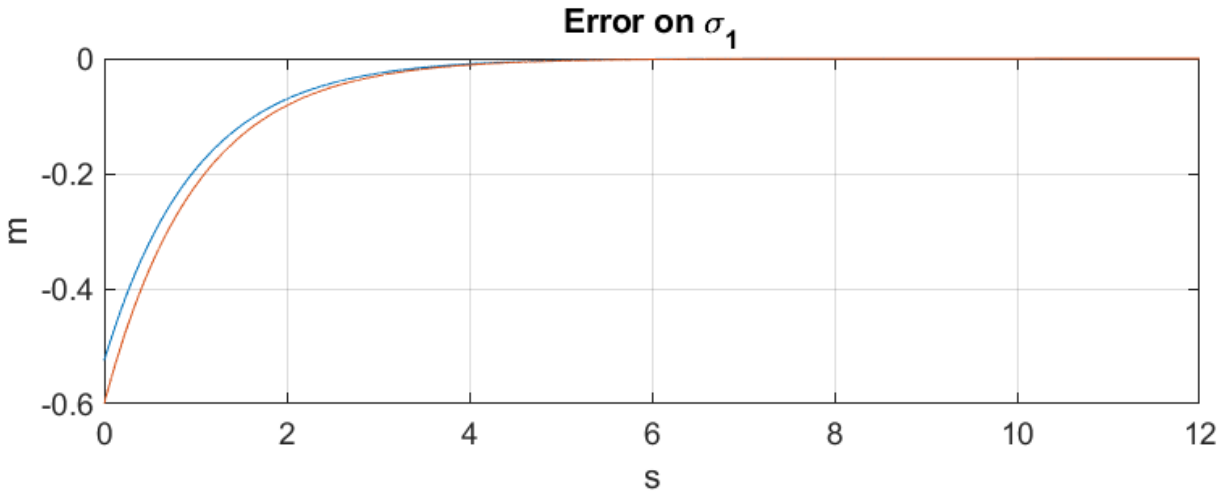

Figure 3.8. Tracking error on one task without the triggering condition with only σ_1 active

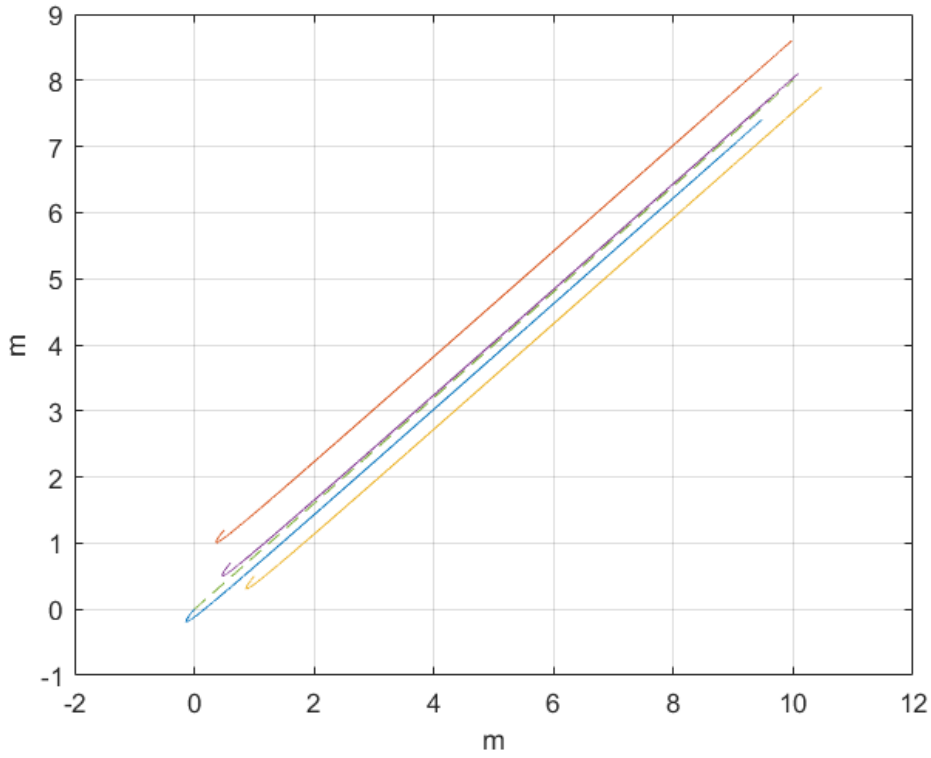

Figure 3.9. Path travelled by the robots

Simulations without the triggering condition with only σ_2 active

The estimation error converges to zero Figure 3.10, and the observer is able to estimate the state of the whole system, since the communication is not interrupted and the information are updated every instant. As shown in the Figure 3.11 the formation tracking error converges to zero.

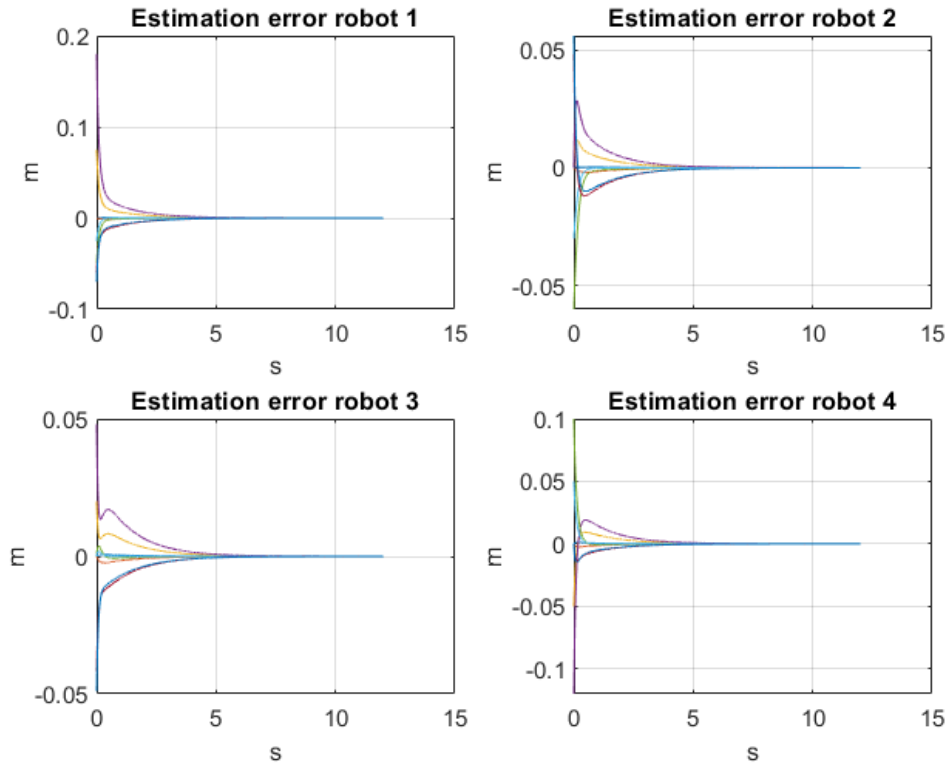

Figure 3.10. Estimation errors relative to the four robots without triggering condition with only σ_2 active

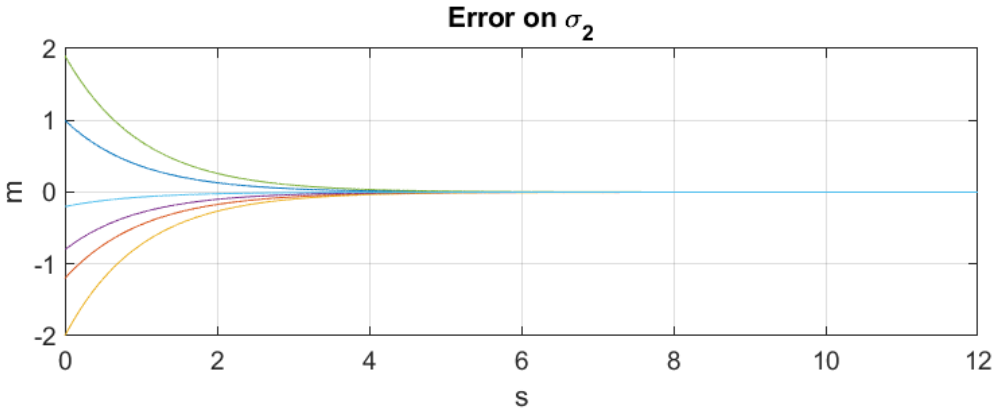

Figure 3.11. Tracking error on one task without triggering condition with only σ_2 active

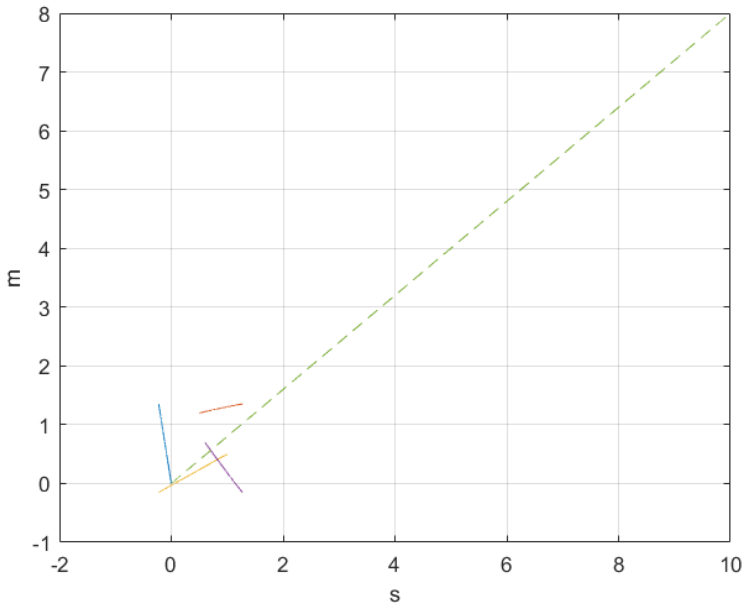

Figure 3.12. Path travelled by the robots

3.0.3 Simulations with the triggering condition based on the estimation error

With the introduction of the triggering condition based on the estimation error (2.69), the behavior of the system highlights some noticeable changes. As can be seen in the Figure 3.15, the exchange of information is not updated every time instant, but only when condition (2.69) is met. This triggering condition is a good compromise between the need of reducing the information exchanged and the performance. Indeed, Figure 3.13 shows that the estimation error, despite an initial transient, converges. As can be seen in Figure 3.16, the robots follow the desired centroid path and the given formation. The condition based on the transition matrix depends on the value of λ_M which changes when $\tilde{\boldsymbol{L}}_c$ changes. To have comparable simulations between the case with both the task active and the one with only one task active, a constant very little value of ϵ is chosen with the changed λ_M.

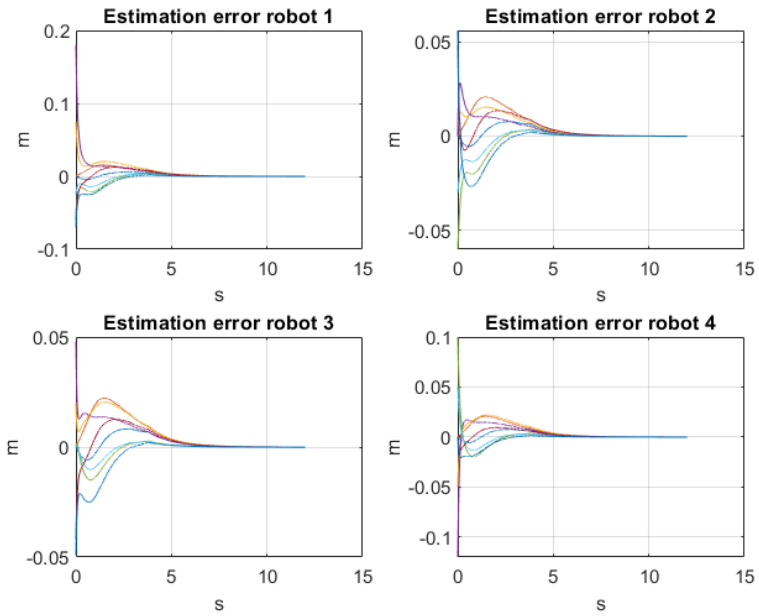

Figure 3.13. Estimation errors relative to the four robots with triggering condition based on the estimation error

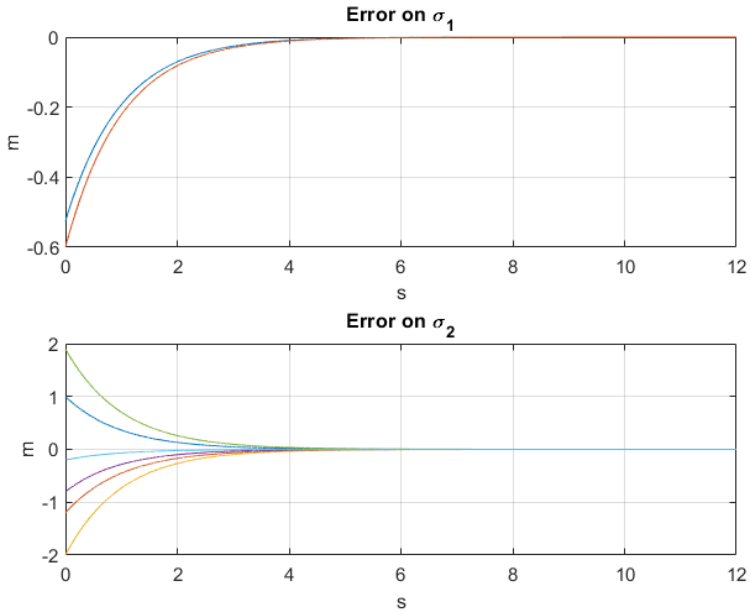

Figure 3.14. Tracking error of the two tasks with triggering condition

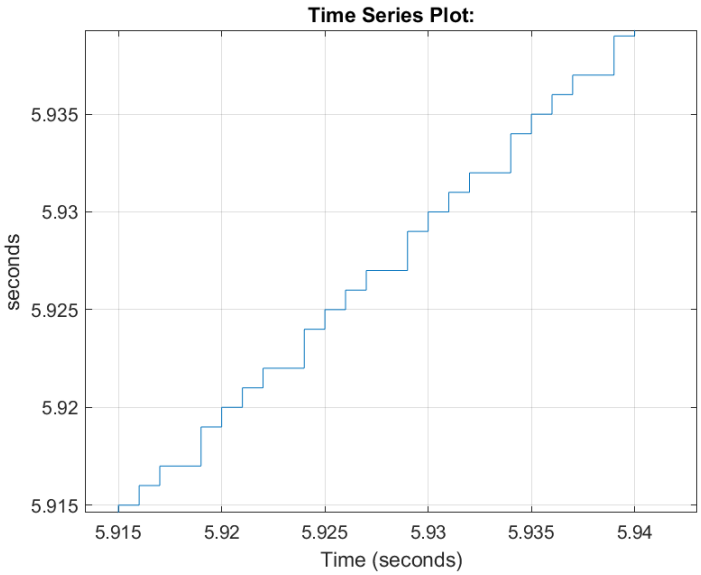

Figure 3.15. Detail of the time update

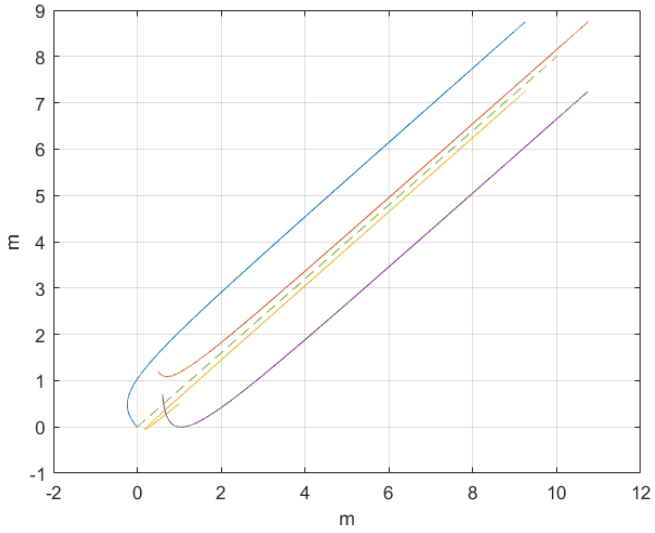

Figure 3.16. Path travelled by the robots

Simulations with the triggering condition and only σ_1 active

In this case, the exchange of information is drastically reduced Figure 3.19, but for this reason, the estimation error struggles to converge. The convergence to zero

occurs, but with some delay compared to the results obtained with both the tasks active.

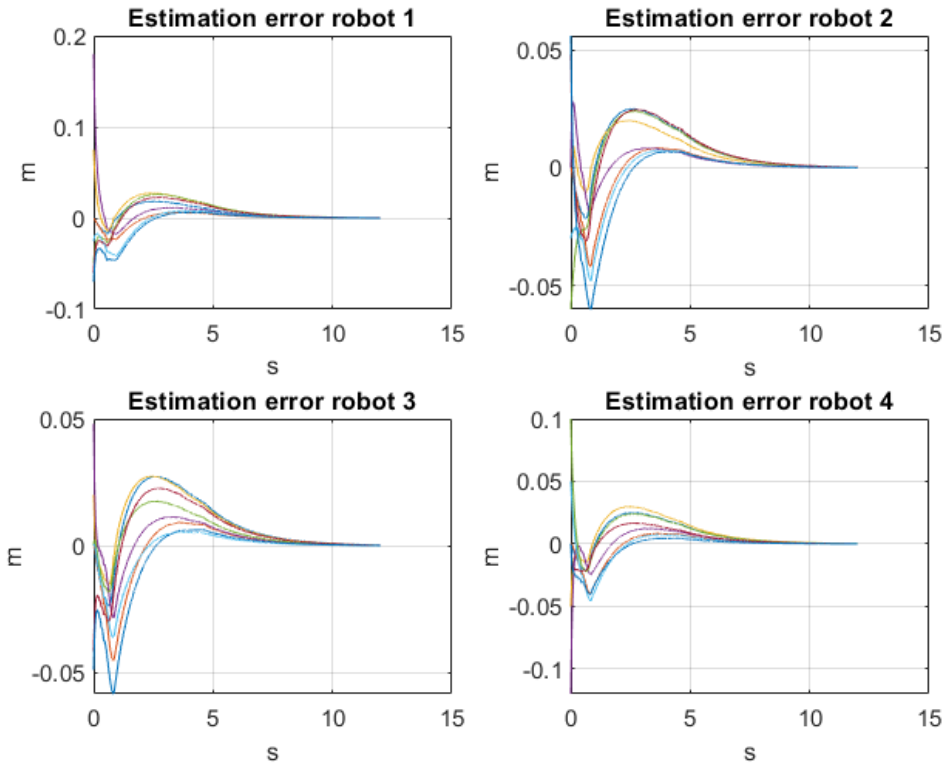

Figure 3.17. Estimation errors relative to the four robots with triggering condition and only σ_1 active

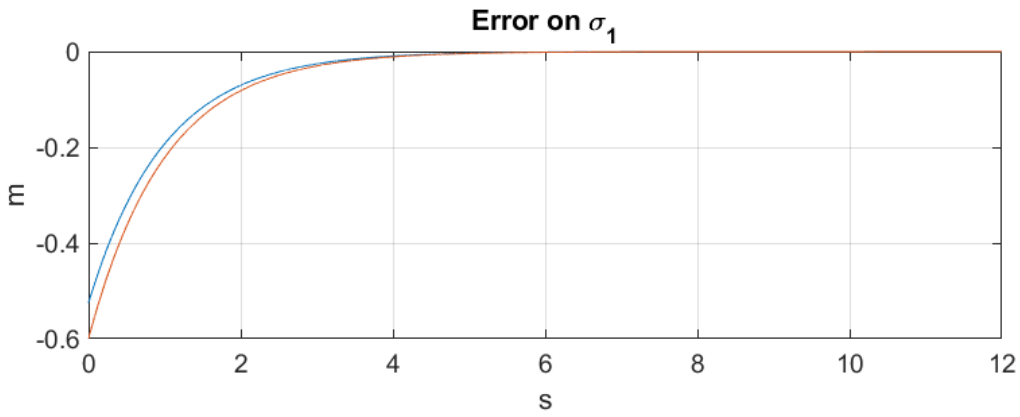

Figure 3.18. Tracking error on the task with triggering condition and σ_1 active

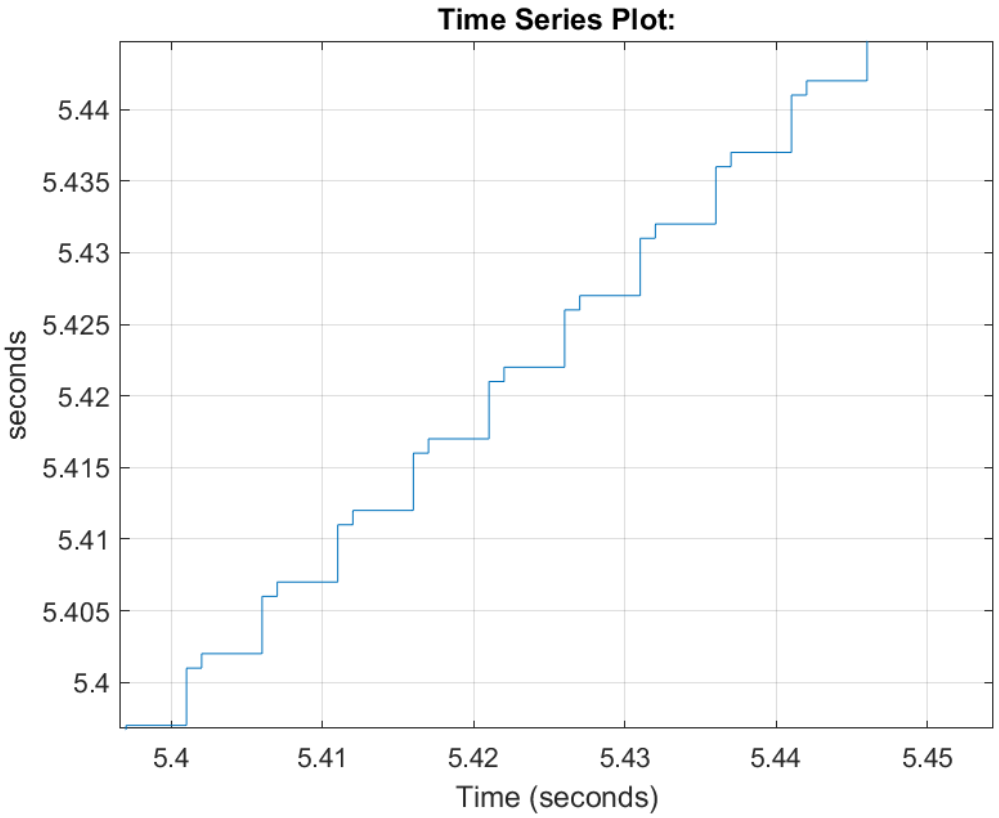

Figure 3.19. Detail of the time update

Simulations with the triggering condition with only σ_2 active

Simulations with the triggering condition and only σ_2 active have shown interesting results. The estimation errors are large during the transient Figure 3.20. The convergence is extremely slow because the information is updated at a slower rate compared to the previous cases. Figure 3.22 shows that with this configuration the robots spend more than five simulation steps without update the estimation of the system received from the neighbors.

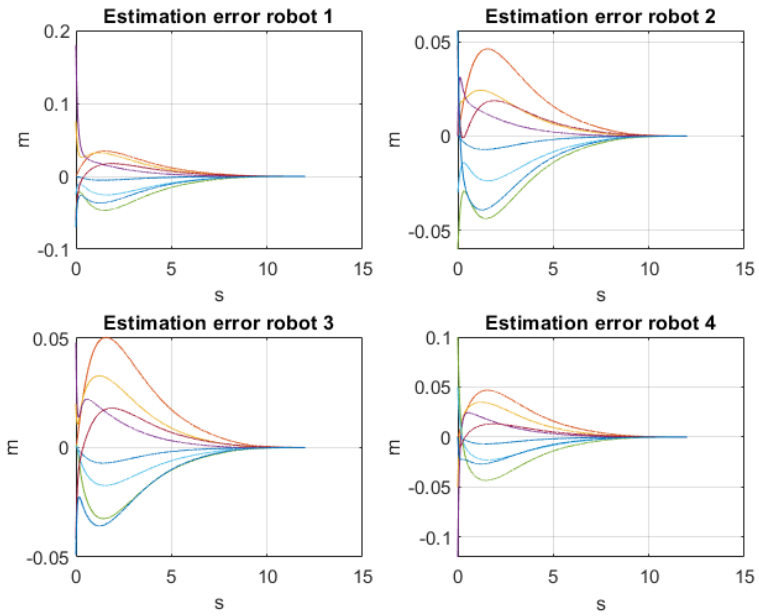

Figure 3.20. Estimation errors relative to the four robots with triggering condition and only σ_2 active

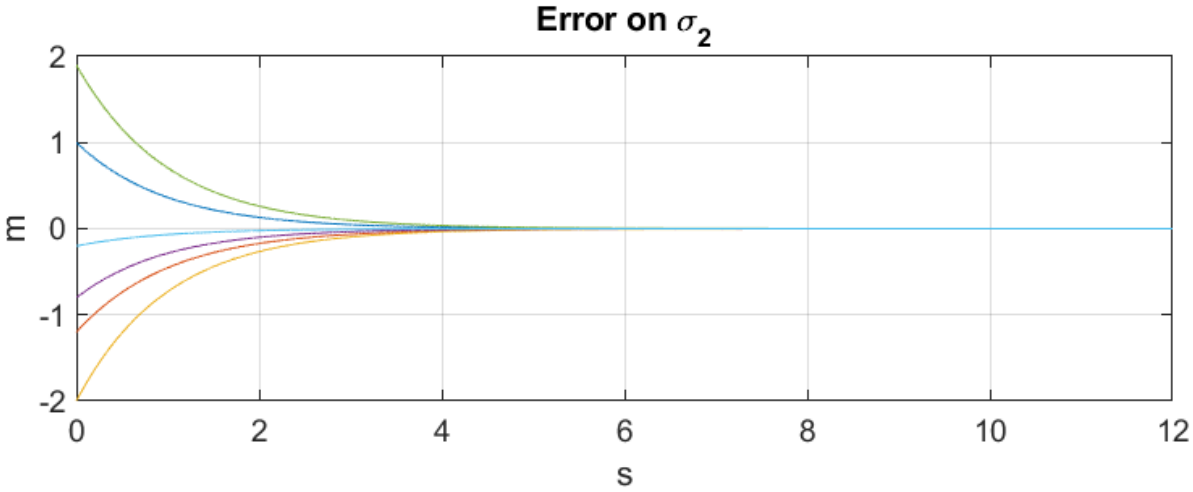

Figure 3.21. Tacking error on the task with triggering condition and only σ_2 active

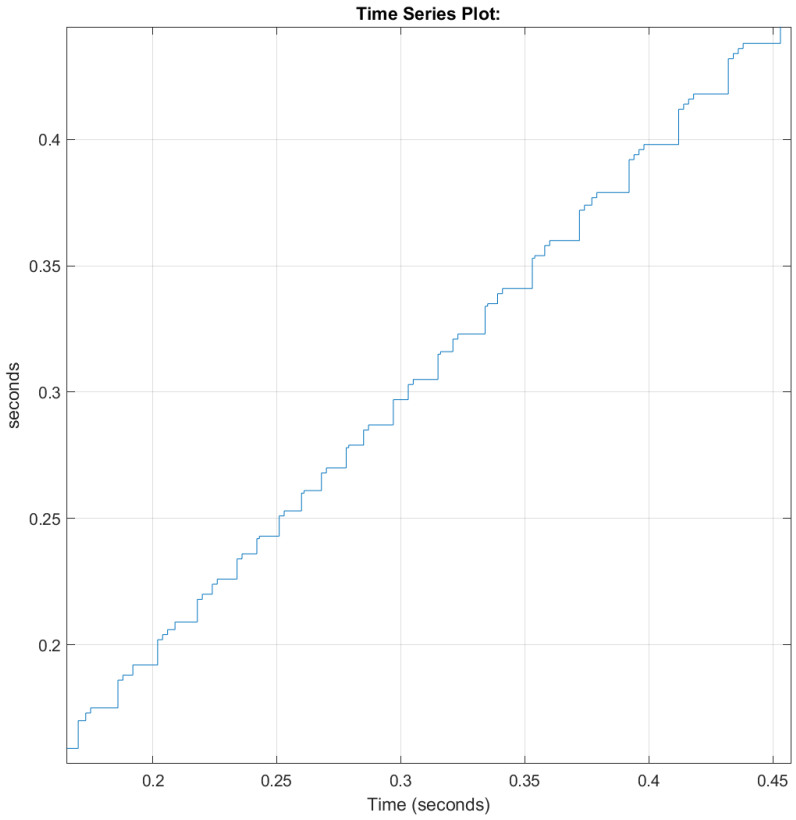

Figure 3.22. Detail of the time update

3.0.4 Simulations with the triggering condition based on the Lyapunov function

Using the triggering condition computed in (2.84), the communication is always triggered, since the threshold is too small. To obtain a suitable threshold is necessary a manual tuning, to have savings in the exchanged information. With the introduction of the triggering condition based on the Lyapunov function (2.84), the differences between the case in which both the tasks are active and the one with only one task active, are more noticeable with respect to the case with the triggering condition based on the estimation error. In fact, as shown in the Figures 3.25 and 3.23, the estimation error converges very quickly despite the lack of updated information available to the robots. As shown in Figure 3.26 the robots follow the desired behavior.

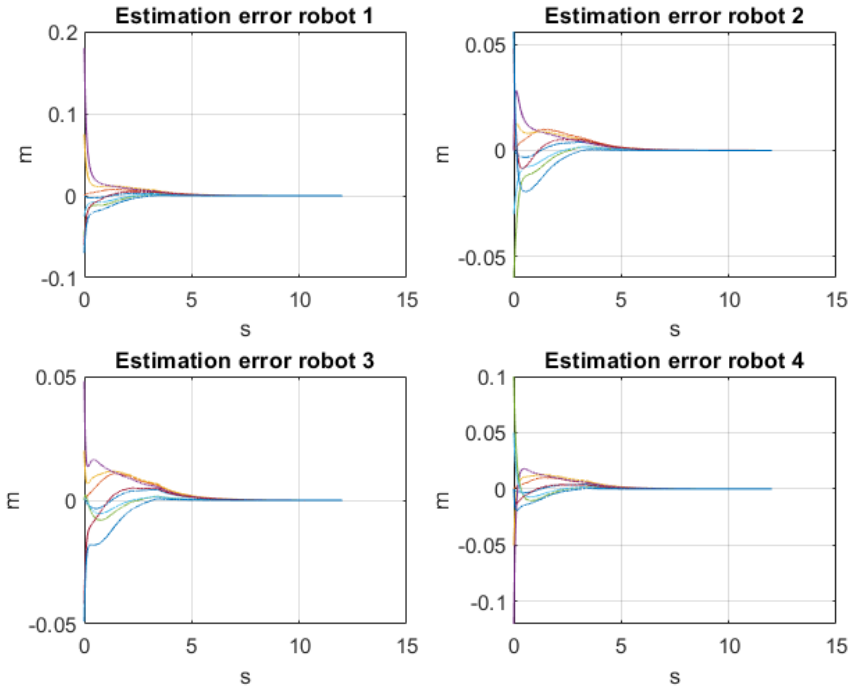

Figure 3.23. Estimation errors relative to the four robots with triggering condition based on Lyapunov function

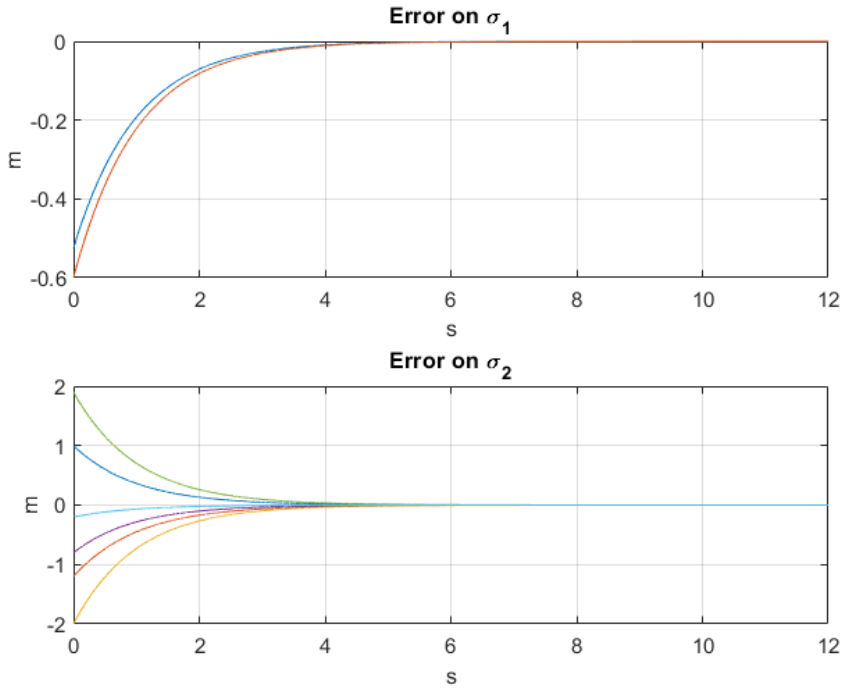

Figure 3.24. Estimation errors on the two tasks with triggering condition based on Lyapunov function

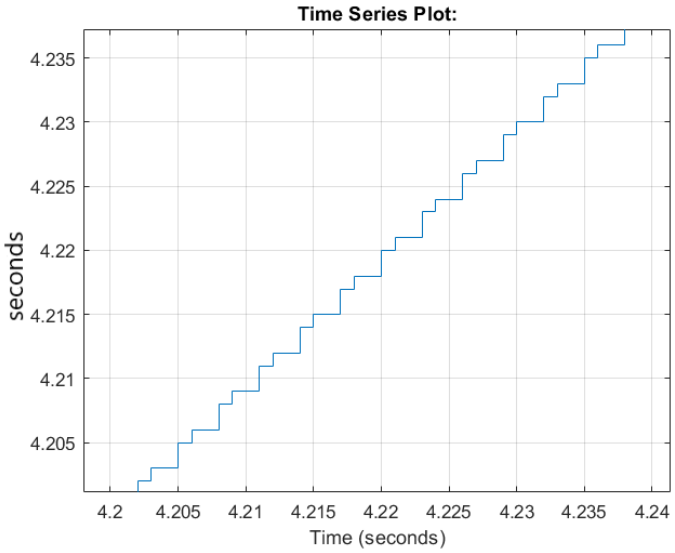

Figure 3.25. Detail of the time update

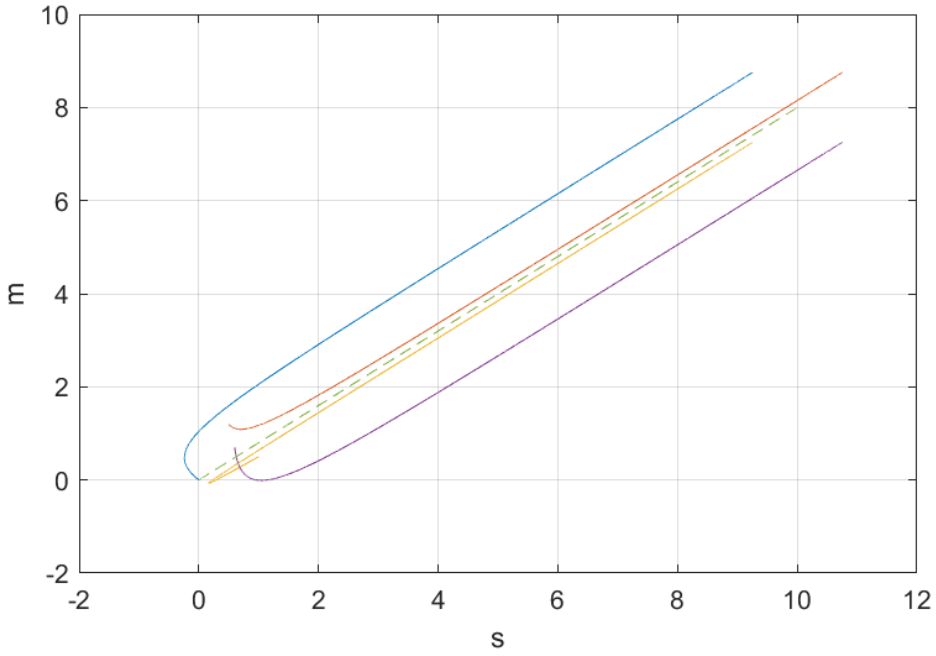

Figure 3.26. Path travelled by the robots

Simulations with the triggering condition based on the Lyapunov function with only σ_1 active

The biggest difference between the triggering condition based on the estimation error and the one based on the Lyapunov function, can be seen when only one task is active. With the Lyapunov function, the estimation error (Figure 3.27) is large at the start of the simulation, then it tends to decrease, but it does not converge at the time of the simulation. Figure 3.29 shows how the communication is restricted: the estimations are exchanged every several steps of the simulation.

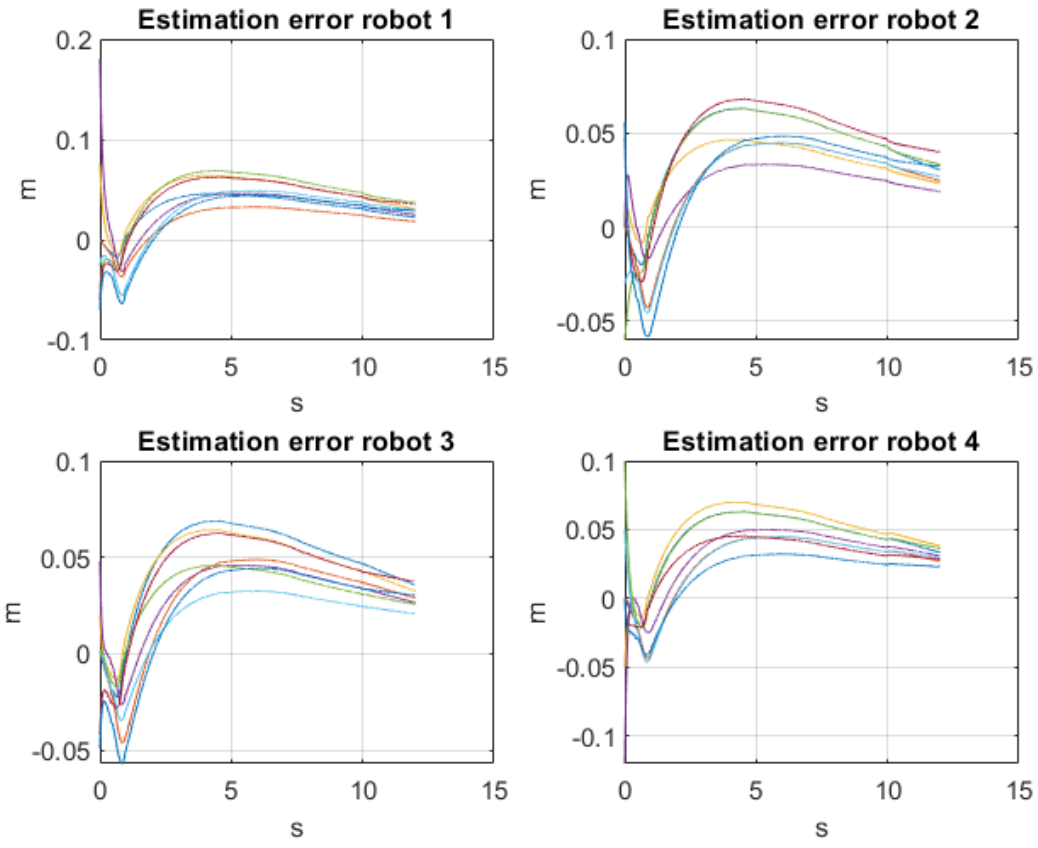

Figure 3.27. Estimation errors relative to the four robots with triggering condition based on Lyapunov function

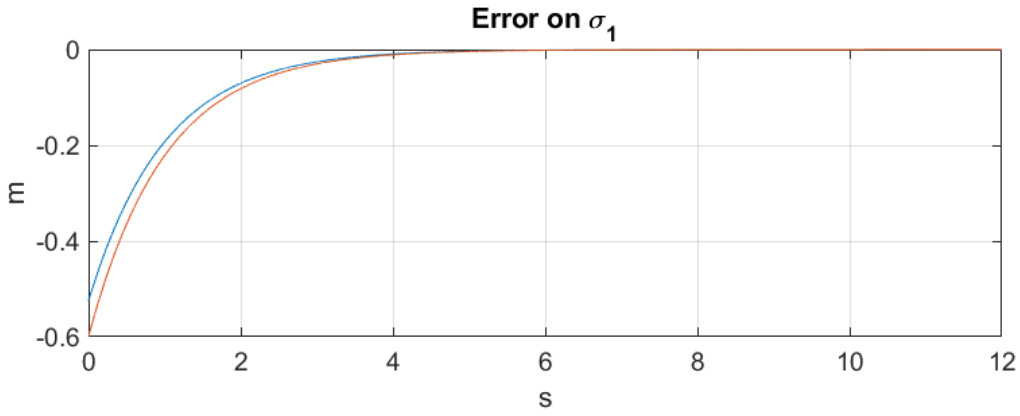

Figure 3.28. Estimation error on the task with triggering condition based on Lyapunov function

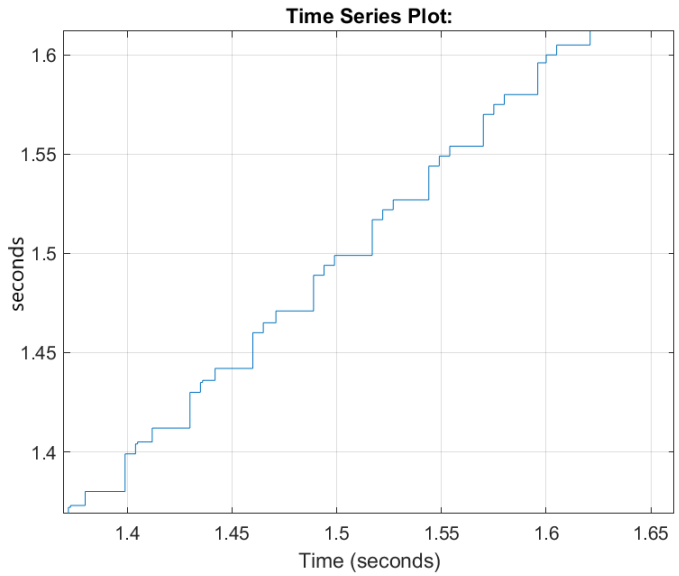

Figure 3.29. Detail of the time update

Simulations with the triggering condition based on the Lyapunov function with only σ_2 active

As can be seen in Figure 3.32, this configuration is the one where the estimated state is updated more rarely, in some intervals of the simulation the exchanged information is not updated for nearly 10 simulation stepsm, indeed. This lack of updates leads to an estimation error unable to converge in the time of the simulation as can be seen in the Figure 3.30.

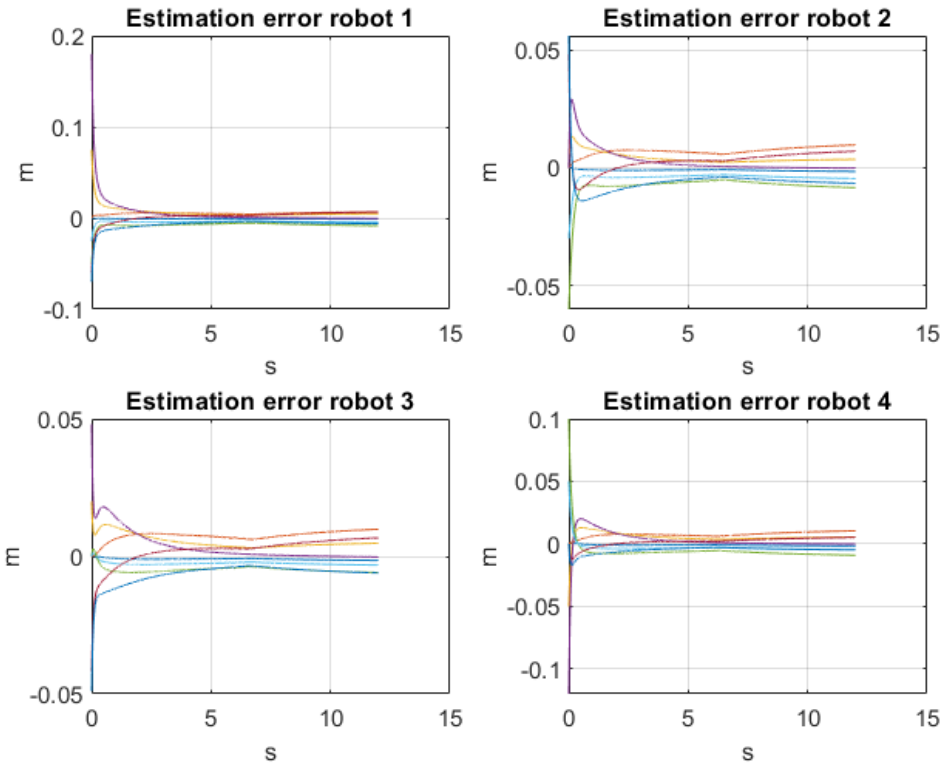

Figure 3.30. Estimation errors relative to the four robots with triggering condition based on Lyapunov function

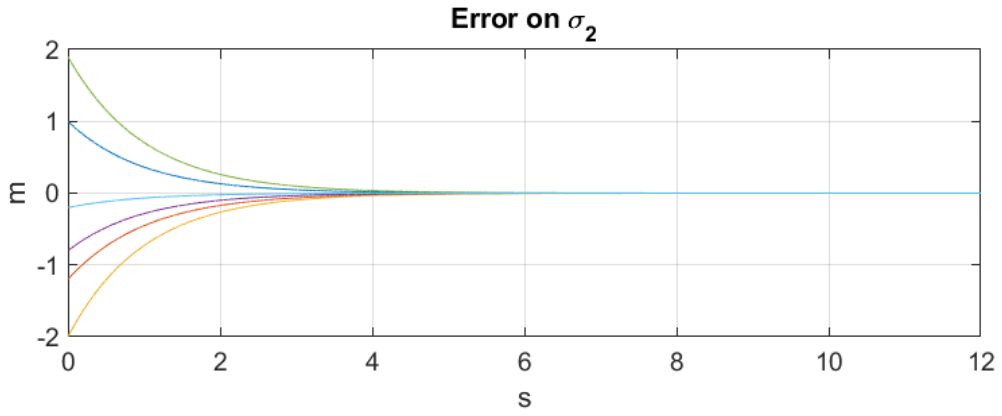

Figure 3.31. Estimation error on the task with triggering condition based on Lyapunov function and only σ_2 active

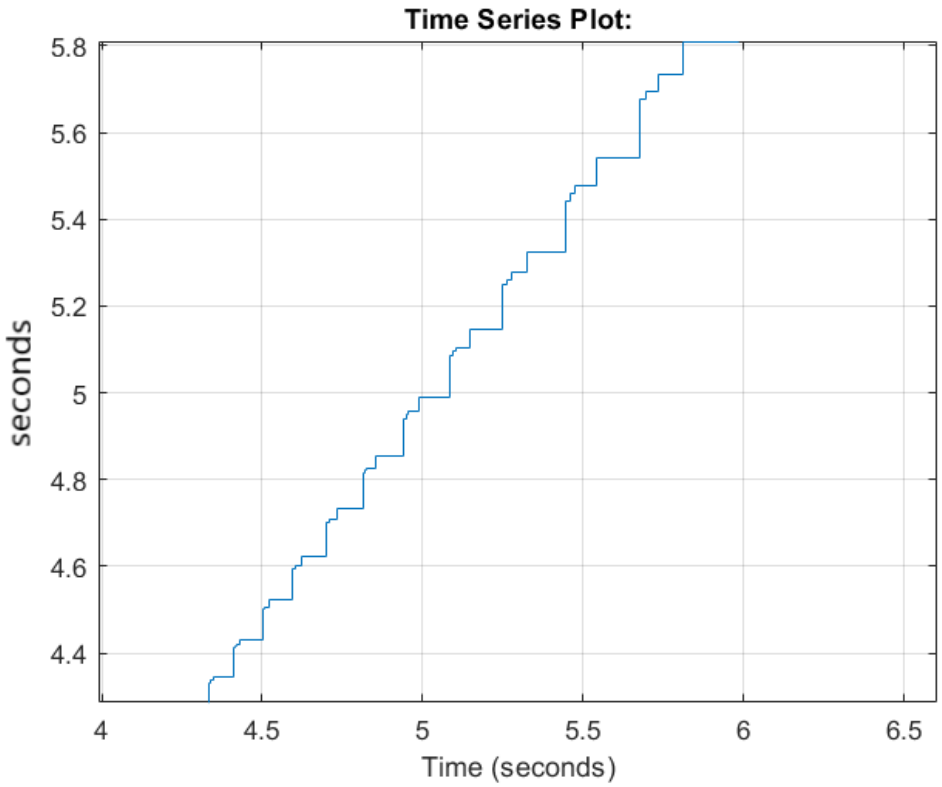

Figure 3.32. Detail of the time update

3.0.5 Real world applications

Robot motion planning and control plays a key role in designing autonomous systems capable to execute different tasks for navigating and manipulating objects in harsh environments. Designing reliable control of a robot consists of numerous tasks: theoretical development of the model, its solution, simulations and implementation of the control algorithm through a suitable hardware. The theoretical development of the model and its simulation represent a crucial step, as trough a strong algorithm and some useful simulations it is possible to predict the behavior of the system when it will be implemented on the hardware. This thesis focuses on this two steps of the development of a control algorithm. Indeed the validation of the presented control scheme permits its future implementation on a real system. The adopted decentralized control scheme, as illustrated before, has some advantages with respect to a classical centralized approach. In a real world application, it is very useful to have less powerful hardware on board because of its cost and its ease to be replaced in case of a malfunction. In a centralized architecture, if a malfunction is found, the whole system ceases to work and the replacement of the central control unit could require a large efforts and maintenance time. A centralized architecture requires large bandwidth available to the central unit, as it has to manage all the information received by robots' team, while a decentralized architecture can be implemented with a smaller bandwidth due to its smaller amount of data exchanged between the agents. In a harsh environment is difficult to obtain a complete communication due to the weather, the vastness of the area to be covered or the ground morphology, while, with a decentralized architecture each robot can be set to exchange information only with the nearest one. The control scheme presented on this thesis could be implemented on various types like drones, wheeled robots or underwater robots; and, with the appropriate sensors, different tasks can be implemented, e.g. visual exploration, transportation of materials and air or ground patrol.

Chapter 4

Conclusion

During the past few decades, substantial progresses have been made into multi-robot localization, formation control, cooperative object transportation and environmental exploring. Several important challenges still remain ahead, as several control strategies are inefficient or reliable only in some particular cases. This thesis has mainly focused on the following two points:

- simulation of a controller-observer scheme for tracking control of the centroid and the relative formation of a multi-robot system. Each robot of the team is equipped with an observer to estimate the whole system's state and a motion control strategy for tracking of the desired centroid and formation.

- introduction, in the control system previously described, of two different triggering conditions, one based on the estimation error and one based on the Lyapunov function, with the aim of reducing the exchanged information among the agents. The main goal of the simulations is to validate these two triggering conditions and verify when they are suitable and when they are not.

The basic control scheme, as already established in the paper [1], is a very robust control scheme. It presents exponential convergence to zero of both the estimation error of the system state and the tracking error relative to the tasks. In the case only one task is active, the control scheme is so robust to make the estimation error converge also if the control law does not concern one task. This happens due to the continue exchange of information (the estimated state of the system) among neighboring robots. After the introduction of the triggering conditions, the results are slightly different. The triggering condition based on the estimation error shows

very good results when both the tasks are active, both the estimation error and tracking error converge to zero. When only one task is active, the convergence to zero of the errors starts to be delayed, due to the presence of eigenvalue equal to zero in the matrix $-\tilde{\boldsymbol{L}}_c$. The second triggering condition, based on the Lyapunov function, shows similar results in the case both the tasks are active. In this respect, the estimation error and the task error present a strong convergence to zero. When only one task is active the control scheme starts to have worse results: the estimation error cannot converge in the time of the simulation. The two triggering condition show similar results in the case of both the tasks are active, but in the case of only one task active the one based on the Lyapunov function is less reliable than the one based on the estimation error. A very important step in the design and the simulation of the triggering conditions is the tuning of the parameters decided by the user $(\bar{\delta}, \sigma, \epsilon)$ since they need to vary in order to guarantee the convergence of the errors. Tuning of these parameters is crucial to obtain a compromise between the reduction of the exchanged information among the robots and tracking performance.

Chapter 5

Appendix

5.0.1 Mathematical expression of $-\tilde{\boldsymbol{K}}_c$ and $\tilde{\boldsymbol{L}}_c$

Given the equations (2.48)–(2.50), the matrix $-\tilde{\boldsymbol{K}}_c$ is composed by

$$-\tilde{\boldsymbol{K}}_c = -\begin{bmatrix} \tilde{\boldsymbol{K}}_{11} & \tilde{\boldsymbol{K}}_2 & \dots & \tilde{\boldsymbol{K}}_i & \dots & \tilde{\boldsymbol{K}}_N \\ \tilde{\boldsymbol{K}}_1 & \tilde{\boldsymbol{K}}_{22} & \dots & \tilde{\boldsymbol{K}}_i & \dots & \tilde{\boldsymbol{K}}_N \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ \tilde{\boldsymbol{K}}_1 & \tilde{\boldsymbol{K}}_2 & \dots & \tilde{\boldsymbol{K}}_{ii} & \dots & \tilde{\boldsymbol{K}}_N \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ \tilde{\boldsymbol{K}}_1 & \tilde{\boldsymbol{K}}_2 & \dots & \tilde{\boldsymbol{K}}_i & \dots & \tilde{\boldsymbol{K}}_{NN} \end{bmatrix}, \tag{5.1}$$

with $(i = 1, \dots, N)$, each vector of $-\tilde{\boldsymbol{K}}_c$ is defined as

$$\tilde{\boldsymbol{K}}_{ii} = \begin{bmatrix} \boldsymbol{K}_{c,1}^{\mathrm{T}} \\ \boldsymbol{K}_{c,2}^{\mathrm{T}} \\ \vdots \\ k_o \boldsymbol{\Gamma}_i^{\mathrm{T}} \\ \vdots \\ \boldsymbol{K}_{c,N}^{\mathrm{T}} \end{bmatrix}, \quad \tilde{\boldsymbol{K}}_i = \begin{bmatrix} \boldsymbol{O}_{n \times Nn} \\ \boldsymbol{O}_{n \times Nn} \\ \vdots \\ -\boldsymbol{K}_{c,i}^{\mathrm{T}} \\ \vdots \\ \boldsymbol{O}_{n \times Nn} \end{bmatrix}, \tag{5.2}$$

and knowing that $\boldsymbol{K}_{c,i}$ is obtained as

$$\boldsymbol{K}_{c,i}^{\mathrm{T}} = \left(s_B k_{cB} \boldsymbol{J}_B + s_F k_{cF} \boldsymbol{J}_{F,i}^{\dagger} \boldsymbol{J}_F \right). \tag{5.3}$$

Breaking down the expressions of $\boldsymbol{J}_B$ and $\boldsymbol{J}_F^{\dagger}$, is possible to obtain $\boldsymbol{K}_{c,i}$ as a function of δ

$$\begin{aligned}\boldsymbol{K}_{c,i}^{\mathrm{T}} &= \delta k_c(\mathbf{1_N} \otimes \boldsymbol{I}_{\mathbf{n}}) + \mathbf{k_{cF}}\boldsymbol{\Gamma}_{\mathbf{i}}^{\mathrm{T}} \\ &= \begin{bmatrix}\delta k_c \boldsymbol{I}_n & \delta k_c \boldsymbol{I}_n & \dots & (\delta k_c + k_{cF})\boldsymbol{I}_n & \dots & \delta k_c \boldsymbol{I}_n\end{bmatrix}, \end{aligned} \tag{5.4}$$

with $\delta k_c = (k_{cB} - k_{cF})/N$.

Thanks to the definition of $\boldsymbol{L}^{\star}$ as $\boldsymbol{L}^{\star} = \boldsymbol{L} \otimes \boldsymbol{I}_{Nn}$, it is possible to obtain

$$\tilde{\boldsymbol{L}}_c = -\begin{bmatrix} k_o l_{11}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_{11} & k_o l_{12}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_2 & \dots & k_o l_{1i}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_i & \dots \\ k_o l_{21}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_1 & k_o l_{22}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_{22} & \dots & k_o l_{2i}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_i & \dots \\ \vdots & \vdots & \dots & \vdots & \dots \\ k_o l_{i1}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_1 & k_o l_{i2}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_2 & \dots & k_o l_{ii}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_{ii} & \dots \\ \vdots & \vdots & \dots & \vdots & \dots \\ k_o l_{N1}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_1 & k_o l_{N2}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_2 & \dots & k_o l_{Ni}\boldsymbol{I}_{Nn} + \tilde{\boldsymbol{K}}_i & \dots \end{bmatrix} \tag{5.5}$$

where the Laplacian $\boldsymbol{L}$ is defined as

$$\boldsymbol{L} = \{l_{ij}\} : \quad l_{ii} = \sum_{j=1, j\neq i}^{N} a_{ij}, \quad l_{ij} = -a_{ij},\ i \neq j\,, \tag{5.6}$$

and $a_{ij} = 1$ if the agent j th can send its estimation of the state of the system to the i th agent, otherwise $a_{ij} = 0$ ($a_{ii} = 0$).

Chapter 6

References

[1] Antonelli G, Arrichiello F,Caccavale F, Marino A (2014) Decentralized time-varying formation control for multi-robot systems. The International Journal of Robotics Research, 33(7), 1029-1043. doi:10.1177/0278364913519149

- Parker L. (2008) Multiple mobile robot systems. In: Siciliano B and Khatib O (eds). Springer Handbook of Robotics. Heidelberg, Germany: Springer-Verlag, pp. 921-941.

- Siciliano B, Sciavicco L, Villani L, Oriolo G (2010) Robotics: Modelling, Planning and Control. Springer. ISBN 978-1-84628-641-4

- Mordechai B, Mondada F (2018) Elements of Robotics. Springer. ISBN 978-3-319-62532-4

- Bemporad A and Rocchi C (2011) Decentralized linear time-varying model predictive control of a formation of unmanned aerial vehicles. In: 50th IEEE conference on decision and control and European control conference (CDC-ECC), Orlando, FL, 12-15 December 2011, pp. 7488-7493

- Ren W and Beard R (2008) Distributed Consensus in Multi-vehicle Cooperative Control (Communications and Control Engineering). Berlin: Springer

- Godsil C and Royle G (2001) Algebraic Graph Theory (Graduate Texts in Mathematics). New York: Springer

- Kågström, "Bounds and perturbation bounds for the matrix exponential,"BIT Numerical Mathematics, vol. 17, no. 1, pp. 39—57,1977. [Online]. Available: https://doi.org/10.1007/BF01932398

- C. V. Loan, "The sensitivity of the matrix exponential,"SIAM Journalon Numerical Analysis, vol. 14, no. 6, pp. 971–981, 1977. [Online].Available: http://www.jstor.org/stable/215667