PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

ÉCOLE NATIONALE POLYTECHNIQUE

Control engineering department

Final Project Thesis

Thesis submitted for the degree of state engineer in control engineering

---

Cooperative observer design for a multi-agent system with defaulting sensors - application on UAVs

---

HADJ-LAZIB Maya

Under the supervision of Pr. **M.TADJINE** and Pr. **A.TAYEBI** Presented

and defended publicly 11/07/2022

Jury member:

President	Mr. Omar STIHI	École Nationale Polytechnique- Algiers
Examiner	Dr. Messaoud CHAKIR	École Nationale Polytechnique- Algiers

2022

Control engineering department / ENP, 10, Oudek brothers' avenue, 182, 16200 El-Harrach, Algiers

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

ÉCOLE NATIONALE POLYTECHNIQUE

Control engineering department

Final Project Thesis

Thesis submitted for the degree of state engineer in control engineering

---

Cooperative observer design for a multi-agent system with defaulting sensors - application on UAVs

---

HADJ-LAZIB Maya

Under the supervision of Pr. **M.TADJINE** and Pr. **A.TAYEBI** Presented

and defended publicly 11/07/2022

Jury member:

President	Mr. Omar STIHI	École Nationale Polytechnique- Algiers
Examiner	Dr. Messaoud CHAKIR	École Nationale Polytechnique- Algiers

2022

Control engineering department / ENP, 10, Oudek brothers' avenue, 182, 16200 El-Harrach, Algiers

ملخص

في هذا المشروع، درسنا مشكل تقدير حالة النظام متعدد العوامل كما قد قمنا بمقارنة و محاكاة مختلف ملاحظات الحالة المقدمة.

كما درسنا في هذا المشروع نمطين من الانظمة متعددة العوامل و مراجعة أدبياتها في مجال المراقبة. و أخيرا تم اقتراح مقاربة في تصحيح أخطاء جهاز الاستشعار في هذه الانظمة و تجربتها على نظام متعدد عوامل الطائرات دون طيار

الكلمات الدالة : النظام متعدد العوامل، ملاحظات الحالة المقدمة، ملاحظات الحالة المقدمة الموزعة، ، تصحيح أخطاء جهاز الاستشعار

Résumé

Dans cette thèse, le problème de l'estimation d'état des systèmes multi-agents est étudié. Différents observateurs d'état distribués ont été comparés et simulés.

Cette thèse étudie deux types de systèmes multi-agents (SMA) et passe en revue leur littérature dans le domaine de l'observation. Enfin, une approche pour la correction d'erreur de capteurs dans les systèmes multi-agents a été proposée et testée sur un SMA de quadrotors.

Mots clés : Systèmes multi-agents, Observation distribuée, Estimation d'état, Correction d'erreur capteur.

Abstract

In this thesis, the problem of state estimation of multi-agent systems is studied. Different distributed state observers have been compared and simulated. This thesis studies two types of multi-agent systems (MAS) and reviews their literature in the field of observation. Finally, an approach for sensor error correction in MAS has been proposed and tested on a multi-agent quadrotor system.

Keywords: Multi-agent systems, Distributed observation, State estimation, Sensor fault correction.

Acknowledgment

I want to thank Pr. Mohamed TADJINE for his help, patience and guidance through the thesis.

I want to thank Pr. Abdelhamid TAYEBI for being an excellent mentor and supervisor and sharing with me his considerable knowledge while inspiring me to always do better.

I would also like to thank the members of the jury of taking their time in reading my work.

I would like to thank the open-source communities who magically had the answers to all my programming questions.

Finally, I wish to thank Aghiles DJEBARA, Samir BOUAZIZ, and Rezkia OULEBSIR who spend several hours proof reading this very document with me.

I dedicate this work

To my mother, my father, and my sister who have always supported me when I needed it, who made thousands of sacrifices for me and for my academic success.

To my beloved ones without whom these last months would have been tiring but above all unbearable: Aicha, Chiraz, Lyna, Lynda, Malika, Sabrina, and Ghiles.

To my fellow control engineering classmates of the year 2021/2022.

And to all the extraordinary people I met at the National Polytechnic School of Algiers.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

$\mathbf{I}_n$	Identity matrix of size n.
$\mathbf{O}_{n\times m}$	Null matrix of size $n \times m$.
$\mathbf{u}_i$	The i^{th} unit vector.
$\mathbb{R}$	The set of real numbers.
$.^T$	Transpose.
$\|\|.\|\|_2$	Euclidean norm
$\|\|.\|\|_\infty$	Infinity norm
$\mathbf{Diag}(v_1, v_2 \ldots, v_n)$	Bloc diagonal matrix with $(v_1, v_2 \ldots, v_n)$ on its diagonal
$Card(.)$	Cardinal.
$Ker(.)$	Kernel.
MAS	multi-agent system.
LTI	Linear time-invariant.
DLO	Distruted Luenberger observer.
DOLTI	Distributed observer for linear time-invariant systems.
DFTO	Distributed finite-time observer.

.

CHAPTER 1

INTRODUCTION

In recent years, multi-agent systems (MAS) have attracted the interest of scientists in many fields. This is because they have appealing properties: they are scalable, efficient, and each agent is usually inexpensive. In a MAS, it is relatively simple to work with each agent, and the focus is usually on the characteristics of the system as a group rather than on the individual agents. For instance, in control system theory, as in many other fields, the tendency is to design distributed algorithms that each agent can implement on its central unit. This is in contrast to the centralized techniques that were initially favored.

Distributed observers have emerged due to the increase in multi-agent systems. Therefore, during the last decades, several studies have been carried out to advance research in this field; many algorithms have been proposed in this sense starting with the various contributions of the Luenberger distributed observer [1], [2], [3] to distributed optimal observers [4]. Each of these research papers has contributed to making this area a trend in control system theory. As the focus in multi-agent systems is not on individuals but the group, individual agents are prone to several defects. Our goal throughout this thesis is to use distributed observation algorithms to estimate the states of faulty agents, using information sent to them by their neighbors in addition to some coupled or relative measures. Thus, we have designed a solution to correct sensor defects in multi-agent systems by estimating the states of the failing subsystems.

Since quadrotor swarms have sparked a lot of attention during the last decade as they are frequently used in critical situations such as rescue missions, surveying, mapping, or agriculture. We have decided to test the proposed algorithm and the different distributed estimators on a multi-agent quadrotor system.

In conclusion, we can summarize the work done throughout this thesis in three distinct points: the study of existing distributed estimators in the literature, the implementation of these observers in Python through a package that we have developed, and finally, the proposal of an application of these distributed algorithms for the correction of sensor defects in a multi-agent system.

1.1 Thesis outline

This thesis is organised as follows:

Chapter 2 It presents the necessary preliminaries for the thesis.

Chapter 3 It introduces three distributed observers and presents a comparison between them.

Chapter 4 It presents distributed observers applied to multi-agent systems containing coupled or relative measures.

Chapter 5 It is devoted to the presentation of observation algorithms used to correct sensor faults.

Chapter 6 It is dedicated to the simulation of the algorithms presented in chapter 5 on a multi-agent quadrotor system.

Chapter 7 It presents the Python package that was designed to implement distributed observers.

Chapter 8 It contains the general conclusion of the thesis, as well as future perspectives.

.

CHAPTER 2

MATHEMATICAL BACKGROUND

2.1 Introduction

In this chapter, we go over some of the mathematical foundations employed in the formulation and the study of multi-agent estimation laws. We introduce graph theory, state observation, and multi-agent systems.

2.2 Graph theory

Graph theory is a field of mathematics commonly used in optimization. It provides key concepts to model, analyze and design network systems and distributed algorithms and defines the interactions between a system of agents.

2.2.1 What is a graph

A graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ consists of a set of nodes (vertices) $\boldsymbol{V}$ and edges $\boldsymbol{E} \subseteq \boldsymbol{V} \times \boldsymbol{V}$. The edges define the link between the vertices. If there exists an edge that connects vertex A to vertex B, then B is said to be in the neighborhood of A, it is denoted (A, B) [5].

If all the edges link two vertices symmetrically (i.e. if A being in the neighborhood of B is equivalent to B being in A's), we say that the graph is undirected. Otherwise, it becomes a digraph (directed graph). The set of neighbors of a vertex A is denoted $\mathcal{N}_{\mathcal{A}}$.

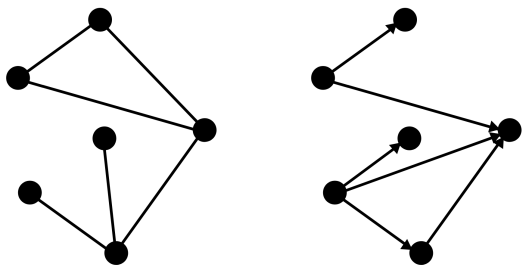

Figure 2.1: Examples of a graph and a digraph

2.2.2 Connected graphs

A graph $\boldsymbol{G}$ is *connected* if there is a path between any two nodes. If the graph is directed, the directions of the edges are to be taken into account and we say that $\boldsymbol{G}$ is *strongly connected*. On the other hand, we say that $\boldsymbol{G}$ is *weakly connected* if the undirected version of the digraph is connected.

2.2.3 The adjacency matrix

The adjacency matrix of a given weighted digraph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, a_{e \in \boldsymbol{E}})$ is a non-negative matrix $A \in \mathbb{R}^{n \times n}$ defined so that each coefficient (i, j) of $\boldsymbol{A}$ is the weight $a_{(i,j)}$ of the edge (i, j), and all other entries of $\boldsymbol{A}$ are equal to zero [6].

$$\begin{cases} a_{(i,j)} > 0 & (i,j) \in \boldsymbol{E} \\ a_{(i,j)} = 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

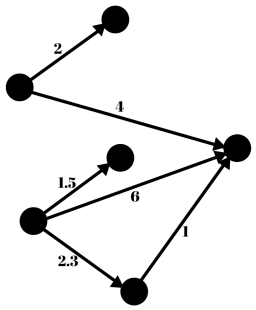

Figure 2.2: An example of a weighted digraph

Consider the case of an unweighted graph, if (i, j) is an edge of $\boldsymbol{G}$ then $a_{(i,j)} = 1$, this matrix is called the binary adjacency matrix.

2.2.4 The degree matrix

Consider a weighted digraph with an adjacency matrix $\boldsymbol{A}$. The degree matrix $\boldsymbol{D}$ is a diagonal matrix, it is computed by summing the weights of each neighbor's vertex.

$$\boldsymbol{D} = \mathbf{diag}(\boldsymbol{A} 1_n) \tag{2.2}$$

2.2.5 The Laplacian matrix

Given a weighted digraph with an adjacency matrix $\boldsymbol{A}$ and a degree matrix $\boldsymbol{D}$, the Laplacian matrix of $\boldsymbol{G}$ is defined as follows.

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} \tag{2.3}$$

It is a positive semi-definite matrix, that is, all its eigenvalues are non-negative.

Consider the following graph.

Figure 2.3: An example graph

The adjacency matrix, the degree matrix and the Laplacian matrix are defined as follows.

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \boldsymbol{D} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \boldsymbol{L} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.4}$$

2.3 Kronecker product

The Kronecker product is an operation on two matrices: $\mathbb{C}^{m\times n} \otimes \mathbb{C}^{p\times q} \to \mathbb{C}^{pm\times qn}$ such that:

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \dots & a_{nn}B \end{bmatrix}$$

With $a_{ij}B$ being the product between the scalar a_{ij} and the matrix B.

The Kronecker product is bilinear and associative but not commutative, that is, for three matrices A, B, and C, and a scalar k:

$$A \otimes (B + C) = A \otimes B + A \otimes C$$
$$(B + C) \otimes A = B \otimes A + C \otimes A$$
$$A \otimes (kB) = (kA) \otimes B = k(A \otimes B)$$
$$A \otimes (B \otimes C) = A \otimes B \otimes A \otimes C$$
$$A \otimes 0 = 0 \otimes A = 0$$

The inverse of a Kronecker product is given by:

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$$

Its transpose is given by:

$$(A \otimes B)^{T} = A^{T} \otimes B^{T}$$

2.4 State observers

2.4.1 State-space representation

A state-space representation of a system in control engineering is a mathematical model written as a set of inputs, outputs and state variables. This model is derived from the differential equations that govern the physical system.

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases} \tag{2.5}$$

With $x \in \mathbb{C}^n$, $u \in \mathbb{C}^m$, and $y \in \mathbb{C}^p$, and with A, B and C matrices with the appropriate dimensions.

2.4.2 Observability

Let us use the definition proposed by *R. Kalman* in his article *"On the general theory of control systems"*.

Definition: *A plant is said to be observable if its exact value at time 0 $x(0)$ can be determined from measurements of the output signal $y(t)$ over the finite interval $0 \leq t \leq t_2$ [...] if every state is observable, we say that the plant is "completely observable" [7].*

Observability deals with the problem of restoring the state vector x(t) from partial and possibly inaccurate measurements called observations, which have been collected during a given time [8]. A linear system is observable if and only if the following observability matrix has a $rank(R) = n$, where n is the number of states in the system.

$$R = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{2.6}$$

The rank of this matrix is called the observability index. It defines the number of observable states in the system.

2.4.3 Detectability

Detectability is weaker than observability; we consider a system detectable if all its unobservable states do not affect its stability and the ability of its output to reach a given state. Let us borrow the definition from [9].

Definition: *A plant is said to be detectable if all its unobservable states are stable.*

2.4.4 Observability sub-spaces

Let's recall the definition of the undetectable subspace of a matrix pair (C, A) as defined in equation 2.5. Let $\alpha_A(s)$ denote the minimal polynomial of A (The polynomial whose roots are the eigenvalues of matrix A). The undetectable subspace of (C, A) is defined in equation 2.7.

$$\mathcal{C} = \cap_{l=1}^{n} Ker(CA^{l-1}) \cap Ker(\alpha_A^+(A)) \tag{2.7}$$

With $\alpha_A^+(s)$ the divisor of $\alpha_A(s)$ such that $\alpha_A^+(s)$'s zeros are in the open left half-plane of the complex plane.

As for the unobservable subspace, it is defined in equation 2.8.

$$\mathcal{O} = \cap_{l=1}^{n} Ker(CA^{l-1}) \tag{2.8}$$

2.4.5 Canonical decomposition

We will borrow the definition from Julio H. Braslavsky lecture on Control systems design [10]:

Definition: *"The Canonical Decomposition of state equations will establish the relationship between [...] Observability , and a transfer matrix and its minimal realisations."*

Throughout this document, we will mainly focus on observability. Thus, let us define the Staircase Observable/Unobservable Decomposition.

Consider a system defined as in equation 2.5, with an observability index of n_O. There exists a transformation matrix T that is orthonormal and decomposes the system into the observable and unobservable states: $\bar{x} = Tx$.

$$\begin{cases} \dot{\bar{x}} = \begin{bmatrix} \dot{\bar{x}}_{\bar{O}} \\ \dot{\bar{x}}_{O} \end{bmatrix} = \begin{bmatrix} \bar{A}_{\bar{O}} & \bar{A}_{12} \\ 0 & \bar{A}_{O} \end{bmatrix} \begin{bmatrix} \bar{x}_{\bar{O}} \\ \bar{x}_{O} \end{bmatrix} + \begin{bmatrix} \bar{B}_{\bar{O}} \\ \bar{B}_{O} \end{bmatrix} u \\ \bar{y} = \begin{bmatrix} 0 & \bar{C}_{O} \end{bmatrix} \bar{x} \end{cases} \tag{2.9}$$

All the matrices and arrays with a O as a subscript are observable and those with a $\bar{O}$ are unobservable.

Computing the staircase decomposition can be done in Matlab or using Harold's package in Python using the *obsvf(.)* function. This function applies the Staircase Algorithm introduced by H.H. Rosenbrock in 1970.

Consider a control system as defined in equation 2.5, with the matrices A, B, and C defined as follows.

$$A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{2.10}$$

This system has an observability index of 1, that is, only one of its states are observable. Applying the staircase algorithm, we find.

$$\bar{A} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \qquad \bar{B} = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix} \qquad \bar{C} = \begin{bmatrix} 0 & 1 \end{bmatrix} \qquad \bar{T} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{2.11}$$

2.4.6 State observer

In control theory, an observer gives an approximation or estimate of the internal state of a system from measurements of its input and output. It is a mathematical tool to correctly estimate the values

of the state vector x at a higher convergence rate than that of the system. The most known observer is the Luenberger observer.

Consider the linear system 2.5 without the control u. An observer of that system can be chosen as:

$$\begin{cases} \dot{\hat{x}} = A\hat{x} + L(y - \hat{y}) \\ \hat{y} = C\hat{x} \end{cases} \tag{2.12}$$

Finding the value of the gain L is equivalent to solving this observation problem, thus estimating the states.

Obviously, in order for the observer in 2.12 to exist, the system 2.5 has to be observable.

2.5 Performance study

To conduct a system performance study, we must first select a few criteria on which to base our investigation. These factors will help us determine the optimal solution later on. This latter will be *"the finest"* based on the criteria we have picked, not in absolute terms.

Before we define the criteria we just mentioned, let's us define *"robustness"*, a critical attribute of a controller or estimator.

Robustness: It refers to the ability to meet requirements and to withstand parameter fluctuations, as well as disturbances and noises induced by the system's surroundings [11].

We can now review the performance indices we will be using throughout this thesis.

The integral square error: It provides a clear picture of current and past inaccuracies while ensuring that values do not cancel out: $J = \int_0^{t_f} e^2 dt$.

Steady-state error: It is the difference between the control system output and the expected response [12].

Settling time: It is the time it takes for the response to approach values close to the steady-state value while remaining within a specified error band [13].

Overshoot: It is the difference between the maximum peak and the steady-state value. A system with a fast response will often have a higher overshoot value [14].

2.6 Multi-agent systems (MAS)

Nature has always been the first inspirational factor for human creations, and this is no different for multi-agent systems whose principles were inspired by bird flocking, fish schooling, ant colonies, bee swarm, and even bacterial growth. In this section, we will introduce the multi-agent control theory. However, before diving into the mathematics of our topic, let us consider two examples.

Consider the flocking behavior that many animal species like birds exhibit. Modeling this behavior in a decentralized manner, we consider a simple "alignment rule" for each animal to steer towards the average heading of its neighbors.

We consider each bird to be a node capable of sensing the heading θ_j of its neighbors $\mathcal{N}_i$ and aligning itself with the detected average heading. Using this definition, we can write a mathematical model (see equation2.13) for the movement of the birds [6].

$$\dot{\theta}_i = \sum_{j \in \mathcal{N}_i} \theta_j - \theta_i \tag{2.13}$$

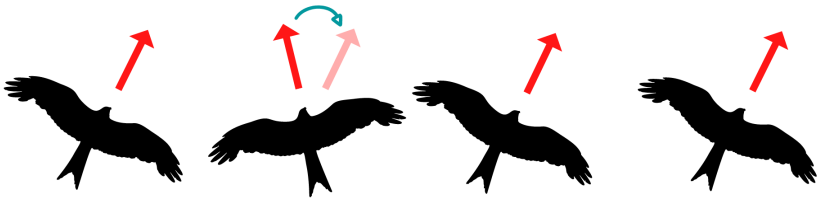

Figure 2.4: Bird flocking

In addition to being ubiquitous in nature, multi-agent systems are widely used in fields such as robotics. One of the examples of multi-agent control that we can mention is "the coordination of robots".

Consider a group of robots that can move in a plane such that $\dot{p}_i = u_i$, with p_i the position of the robot. The robots' task is to find each other at a common point using only onboard sensors. In most cases, this is achieved using a cycling pursuit where each robot picks a neighbor and pursues it. This pursuit is carried out by choosing the following control law: (2.14).

$$u_i = p_j - p_i \tag{2.14}$$

With j chosen from $\mathcal{N}_i$.

2.6.1 Definition

A multi-agent system is a group of agents that meet a set of requirements. These requirements allow them to interact with the environment and to operate autonomously in it. According to [15], an agent needs to have the following properties:

- **Autonomy:** the ability to perform tasks without any human intervention.
- **Perception:** the ability to sens the changes in its environment.
- **Interaction:** the ability to interact with other agents from its environment.
- **Reactivity:** the ability to react to the changes of its environment.
- **Pro-activeness:** the ability to take action to fulfill its mission.

2.6.2 The state-space model of MAS

Consider a group of m agents that can communicate with each other. Each agent has a set of neighbors with which it can share information. This relationship is defined through a directed or undirected graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. This latter has no self-arcs (namely $(i, i) \notin \boldsymbol{E}$). $\boldsymbol{A}$, $\boldsymbol{D}$, and $\boldsymbol{L}$ are the adjacency matrix, the degree matrix and the Laplacian matrix of the graph $\boldsymbol{G}$, respectively.

Each agent is defined using a state-space model. If all the agents of the system have the same model, we say that the MAS is homogeneous. Otherwise, it is heterogeneous. Throughout the document, we will only consider linear systems or linearized models of non-linear systems.

$$\begin{cases} \dot{x}_i = A_i x_i + B_i u \\ y_i = C_i x_i \end{cases} \tag{2.15}$$

With $x \in \mathbb{R}^{n_i}$ and $i \in \{1, 2, \ldots, m\}$.

We can also consider each agent to be a sub-system of a larger plant. The sub-systems have access to some of the outputs of the plant.

$$\begin{cases} \dot{x} = Ax + Bu \\ y_i = C_i x \end{cases} \tag{2.16}$$

With $x \in \mathbb{R}^{nm}$, m is the number of agents and n the number of states of each system, for $i \in \{1, 2, \ldots, m\}$.

In this case, x is the global state of all local states of each agent, and u is the control of the global plant.

2.6.3 The consensus algorithm

The consensus problem is a fundamental problem in formation control theory, its goal is for the different agents to converge to the same value or *agree* on a result, we say that the system has reached consensus [16].

In control theory, the most common continuous-time consensus algorithm for a group of agents connected through the topology of a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ is given by the following equation.

$$\dot{x}_i = -\sum_{j=1}^{m} a_{ij}(x_i(t) - x_j(t)) \tag{2.17}$$

With the adjacency matrix of the graph $\boldsymbol{G}$ given by $\boldsymbol{A} = [a_{ij}]$ for all $i, j \in \{1, 2, \ldots, m\}$ and $x \in \mathbb{R}^n$.

$$\dot{x}_i = -\sum_{j=1}^{m} a_{ij} x_i(t) + \sum_{j=1}^{m} a_{ij} x_j(t)$$

$$\dot{x} = -\begin{bmatrix} \sum_{j=1}^{m} a_{ij} & 0 & \ldots & 0 \\ 0 & \sum_{j=1}^{m} a_{ij} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sum_{j=1}^{m} a_{ij} \end{bmatrix} x(t) + \begin{bmatrix} a_{00} & \ldots & a_{0j} & \ldots & a_{0m} \\ a_{10} & \ldots & a_{ij} & \ldots & a_{00} \\ \vdots & & \vdots & & \vdots \\ a_{m0} & \ldots & a_{mj} & \ldots & a_{mm} \end{bmatrix} x(t)$$

$$\dot{x} = -\boldsymbol{D}x(t) + \boldsymbol{A}x(t) = -\boldsymbol{L}x(t)$$

We considered a group of 10 agents with $x \in \mathbb{R}^3$ and simulated the system using the consensus algorithm with a random adjacency matrix and a random initialization of x with $x(t = 0) \in [-3, 3]$, we obtained the following results.

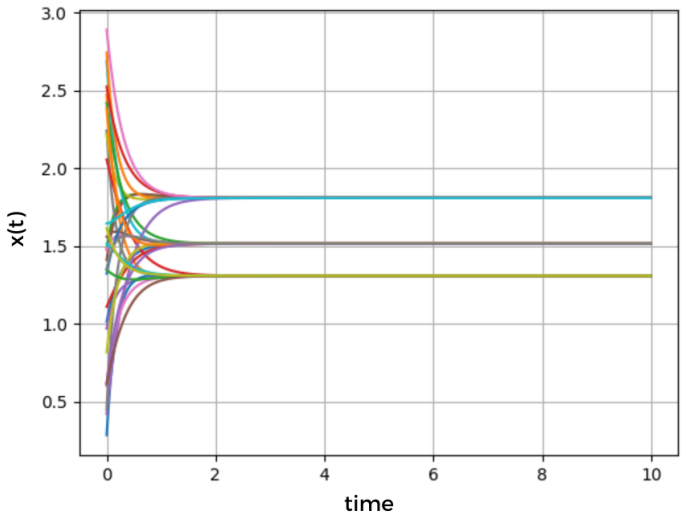

Figure 2.5: Example of a system reaching consensus

All the three state-space variables of each agent reach the same value after less than 2 seconds.

2.7 Software

To implement the different solutions, we have used a variety of software tools that will be introduced in this section.

Python: It is a dynamically semantic, interpreted, object-oriented high-level programming language. Its high-level built-in data structures, as well as its dynamic typing and binding, make it ideal for rapid application development as well as for use as a scripting or glue language to link existing components [17].

Numpy It is the most important open-source Python library for scientific computing. Numpy or Numerical Python can be used as a multidimensional container for generic data in addition to using it for scientific applications. [18].

Scipy It is the abbreviation for Scientific Python. It provides several utility functions for optimization, statistics, and signal processing. Like NumPy, SciPy is open-source [19].

Matplotlib It is a Python package dedicated to creating static, animated, and interactive visualizations [20].

Control The Python Control Systems Library is an open-source Python module that requires NumPy, Scipy, and Matplotlib. It implements basic operations for the analysis and design of feedback control systems [21]. For instance, we can compute the observability matrix of a state-space system using the function *obsv*().

Harold This package is an open-source systems and controls toolbox for Python3 with the purpose of being a control engineer's/student's/daily researcher's workhorse. It makes use of scientific Python tools like NumPy and SciPy. For example, it includes the staircase decomposition function, which is missing from the control library. It was created by Ilhan Polat [22].

.

CHAPTER 3

DISTRIBUTED OBSERVATION OF A SYSTEM OF AGENTS

3.1 Introduction

In recent years, there has been tremendous progress in control engineering in general, and multi-agent systems research in particular. When decisions are made by independent and individual agents rather than a central body, there are many benefits, particularly regarding computation time. However, that also introduces additional difficulties and unsolved problems. Such challenges have been identified as important study topics in state estimation and control [23]. For this reason, there is currently considerable interest in designing a distributed observer for locally estimating the states of a plant. The most attractive feature of this type of observers is its structure, which depends only on local estimates and information exchanged between agents. These observers offer great flexibility and are often needed when the observability or detectability of a system can only be achieved by bringing together a group of agents.

3.2 Problem statement

Throughout this chapter, we are interested in a **fixed** network of m agents which can communicate with their neighbors. The connections between the agents are defined using a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. We won't consider each agent to be its own neighbor; that is, $\boldsymbol{G}$ has no self-arcs. The plant is modeled using a **linear time-invariant** state-space representation:

$$\dot{x} = Ax + Bu \tag{3.1}$$

With $x \in \mathbb{R}^n$, and $u \in \mathbb{R}^r$ the input of the system.

Here, all the agents are part of the same plant; x is the global state of all local states of each agent, and u is the control applied to the plant in its entirety.

Each agent can measure a linear combination of the plant states:

$$y_i = C_i x \tag{3.2}$$

With $i \in \{1, 2, \ldots, m\}$, and $y_i \in \mathbb{R}^{q_i}$, with q_i the number of outputs of agent i.

The goal is to define a suitable group of observers that can jointly estimate the states of the plant. There are generally three objectives that we try to accomplish [24]:

- Obtain an observation error that is exponentially stable with a preassigned convergence rate.
- Reduce the information exchanged between the neighboring agents.
- Reduce the dimensions of the observers.

3.3 Distributed observers

A distributed observer provides an estimate of the internal states of a plant that forms a multi-agent system using measurements of inputs and outputs of all or parts of the plant [8, Chapter 5]. These observers are not centralized. Therefore the measurements are made separately by each agent. Indeed, each agent computes an approximation of the states of the system as a whole. Thus, getting a correct estimation of the plant states would mean reaching a consensus between all subsystems.

In this section, we will go through three different types of distributed observers that have been found in the literature. The most extensively used estimator, the distributed variant of the Luenberger observer (DLO), will be mentioned first. Then, we will present a tweaked version of the DLO. Finally, we will introduce a distributed observer based on decentralized control principles.

3.3.1 The distributed Luenberger observer (DLO)

The distributed Luenberger observer (DLO) is inspired from the Luenberger observer and the consensus problem, it estimates the states of the plant while ensuring a consensus in the estimated values. It was first proposed in [25].

Consider an agent i that can communicate with its neighborhood and **receive output values** y_j **such that** $j \in \mathcal{N}_i$, the estimate x_i of the plant states x is made by each agent i using the following law.

$$\dot{\hat{x}}_i = A\hat{x}_i + L_i(y_i - C_i\hat{x}_i) + \gamma M_i^{-1}(k_i) \sum_{j \in \mathcal{N}_i} \alpha_{ij}(\hat{x}_j - \hat{x}_i) \tag{3.3}$$

For all $i \in \{1, 2, 3, \ldots, m\}$, with α_{ij} the elements of the adjacency matrix $\boldsymbol{A}$ of the graph $\boldsymbol{G}$. L_i and M_i are gain matrices, and γ and k_i are parameters such that $k_i > 0$, $\gamma > 0$. **Each agent, therefore, has an observer of the size of the plant.**

We considered that the system is not under the influence of any control law: u = 0. This assumption is reasonable since we are interested in the state observation only, in this case, the system would evolve depending only on its initial conditions. We may later add an observer-based control to our system.

The term $(y_i - C_i\hat{x}_i)$ ensures that each estimate converges to the real values of the states while the term $\sum_{j \in \mathcal{N}_i}(\hat{x}_j - \hat{x}_i)$ ensures that all the estimated states converge to the same value.

The parameters L_i and M_i are chosen using the intrinsic properties and matrices of the system. For each agent, we define T_i as the orthonormal transformation matrix that transforms the system into its observability staircase form (that depends on the output matrix C_i) [3], [2].

Consider the observable/unobservable decomposition of the state-space system defined in equation 3.1, such that $\bar{x}_i = T_i x_i$. For all $i \in \{1, \ldots, m\}$.

$$\begin{cases} \dot{\bar{x}}_i = \begin{bmatrix} \dot{\bar{x}}_{i\bar{O}} \\ \dot{\bar{x}}_{iO} \end{bmatrix} = \begin{bmatrix} A_{i\bar{O}} & A_{i12} \\ 0 & A_{iO} \end{bmatrix} \begin{bmatrix} \bar{x}_{i\bar{O}} \\ \bar{x}_{iO} \end{bmatrix} + \begin{bmatrix} B_{i\bar{O}} \\ B_{iO} \end{bmatrix} u \\ \bar{y}_i = \begin{bmatrix} 0 & C_{iO} \end{bmatrix} \bar{x}_i \end{cases} \tag{3.4}$$

Let us rewrite the parameters L_i and M_i as follows to uncouple their impact on the observable and unobservable states.

$$L_i = T_i \begin{bmatrix} L_{id} \\ 0_{n-v_i \times q_i} \end{bmatrix}, \quad M_i(k_i) = T_i \begin{bmatrix} k_i M_{id} & 0 \\ 0 & I_{v_i \times v_i} \end{bmatrix} T_i^T \tag{3.5}$$

With v_i the dimension of the said unobservable subspace.

Using a well-chosen Lyapunov function, it is possible to choose the gain matrices L_i and M_i to ensure the convergence of the observer [2]. It has also been shown that for sufficiently large values of γ and k_i, the estimates $\hat{x}_i(t)$ always approach $x(t)$ **exponentially at preassigned rate** [3]. This rate is defined by assigning the right eigenvalues when determining the parameters L_i and M_i. Indeed, they must be chosen according to the following conditions:

- The following matrix must have eigenvalues with negative real part.

$$A_{i_O} - L_{id} C_{i_O} \tag{3.6}$$

- The parameter M_{id} is the solution to the following equation.

$$(A_{i_O} - L_{id} C_{i_O})^T M_{id} + M_{id}(A_{i_O} - L_{id} C_{i_O}) = -I \tag{3.7}$$

3.3.2 The Distributed Finite-Time Observer (DFTO)

This observer is an enhanced version of the distributed Luenberger observer (DLO). It allows obtaining the estimation in a finite time [26].

Consider a system as defined in equation 2.5, and suppose that the matrix A is given in the form of a block diagonal matrix and the output matrices C_i are the i^{th} unit vector, that is, the plant is in the form of multiple chains of integrators.

$$A = \begin{bmatrix} A_1 & 0 & \dots & 0 \\ 0 & A_2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & A_m \end{bmatrix} \tag{3.8}$$

$$C_i = \begin{bmatrix} 0 & \dots & I_{n_i} & \dots & 0 \end{bmatrix} \tag{3.9}$$

With n_i being the size of matrix A_i, while A is an $N \times N$ matrix with N being the sum of the sizes of the subsystems A_i such that $N = \sum_{i=1}^{m} n_i$, for $i \in \{1, \dots, m\}$.

The finite-time observer presented in [26] is a clever combination between the distributed Luenberger observer (DLO) and the super twisting observer. As in the previous section, we denote agent i's

estimate by x_i, and consider that the system is not controlled: u = 0. This assumption is reasonable since we are interested in the state observation only, in this case, the system would evolve depending only on its initial conditions. We may later add an observer-based controller to our system.

Consider an agent i that can communicate with its neighborhood $\mathcal{N}_i$ and receive output values y_j such that $j \in \mathcal{N}_i$:

$$\dot{\hat{x}}_i = A\hat{x}_i + L_i[y_i - C_i\hat{x}_i]^\gamma + \sum_{j\in\mathcal{N}_i} \alpha_{ij}[\hat{x}_j - \hat{x}_i]^\beta \tag{3.10}$$

With L_i is a gain matrix, γ and β, two parameters that must be well chosen and α_{ij} the elements of the adjacency matrix $\boldsymbol{A}$ of the graph $\boldsymbol{G}$. In this case as well, **each agent has an observer of the size of the plant.**

With $[.]^a$ the sign preserving element-wise exponentiation defined as follows

$$[x]^a = \begin{bmatrix} sign(x_1)|x_1|^a & sign(x_2)|x_2|^a & \dots & sign(x_n)|x_n|^a \end{bmatrix}$$

The parameters γ and β are chosen according to the following law.

$$\gamma = \frac{1+v}{1+(N-1)v} \tag{3.11}$$

$$\beta = 1 + v \tag{3.12}$$

We choose v experimentally to obtain the desired convergence rate.

The gain matrix L_i is chosen in such a way so that it only affects the observable states of the system. Since it is obvious that the states observed by agent i are those multiplied by matrix A_i (see equation 3.8), L_i is designed as follows.

$$L_i = \begin{bmatrix} 0_{n_1\times n_i} \\ \vdots \\ L_{n_i} \\ \vdots \\ 0_{n_m\times n_i} \end{bmatrix} \tag{3.13}$$

With L_{n_i} an $n_i \times n_i$ gain matrix.

Finally, we can use pole placement to compute L_{n_i} by choosing the eigenvalues of $A_i - L_{n_i}C_i$. This technique allows us to control the convergence rate of the observer by simply manipulating the eigenvalues.

3.3.3 The Distributed Observer for Time-Invariant linear systems (DOLTI)

This observer uses an augmented state z_i to estimate the states of the plant defined in equations 3.1 and 3.2. It assumes that the agents can communicate with one another, but the only information that can be transmitted is the outputs y_j and the augmented state z_j, for $j \in \mathcal{N}_i$.

We denote the estimate of agent i as $\hat{x}_i$, such that $\hat{x}_i \in \mathbb{R}^n$.

$$\begin{cases} \dot{z}_i = \sum_{j\in\mathcal{N}_i}(H_{ij}z_j + K_{ij}y_j) \\ \dot{\hat{x}}_i = \sum_{j\in\mathcal{N}_i}(M_{ij}z_j + N_{ij}y_j) \end{cases} \tag{3.14}$$

With H_{ij}, K_{ij}, M_{ij}, and N_{ij} gain matrices with the appropriate dimensions, and with $i \in \{1, 2, \ldots, m\}$.

It is possible to simplify the observer by considering that the only information that can be transmitted from one neighboring agent to another is z_i.

$$\dot{z}_i = \sum_{j\in\mathcal{N}_i}(H_{ij}z_j) + K_i y_i \tag{3.15}$$

$$\dot{\hat{x}}_i = \sum_{j\in\mathcal{N}_i} M_{ij}z_j \tag{3.16}$$

Since the goal is to obtain $\hat{x}_i = x$, equation 3.16 must hold for any possible $x \in \mathbb{R}^n$, that is, there exist $z_i \in \mathbb{R}^{n_i}$ which is the solution of equation 3.16 for any $x \in \mathbb{R}^n$.

We write v_i the solution of equation 3.16 when $x^T = u_i$, where u_i is the i^{th} unit vector. We stack all these solutions and obtain:

$$\begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix} = \sum_{j\in\mathcal{N}_i} M_{ij} \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ v_1 & v_2 & \ldots & v_n \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{3.17}$$

$$I_n = \sum_{j\in\mathcal{N}_i} M_{ij}V_j \tag{3.18}$$

Let us consider the observation error $x - \hat{x}_i$. From the equation above (3.18), we can deduce that $x = \sum_{j\in\mathcal{N}_i} M_{ij}V_j x$, furthermore, we know that $\dot{\hat{x}}_i = \sum_{j\in\mathcal{N}_i} M_{ij}z_j$. Therefore, we find that the observation error is proportional to e_i, such that $e_i = z_j - V_j x$.

$$x - \hat{x}_i = \sum_{j\in\mathcal{N}_i} M_{ij}e_j \tag{3.19}$$

Since designing the DOLTI comes down to finding the conditions that govern the different gain matrices, we will study the dynamics of the error e_i, which is proportional to the observation error.

$$\dot{e}_i = \dot{z}_i - V_i\dot{x} = \sum_{j\in\mathcal{N}_i}(H_{ij}z_j) + K_iC_ix - V_iAx \tag{3.20}$$

We add and substract to the equation above (3.20) the term $\sum_{j\in\mathcal{N}_i}(H_{ij}V_j)x$, and obtain.

$$\dot{e}_i = \sum_{j \in \mathcal{N}_i} H_{ij}(e_j) + [\sum_{j \in \mathcal{N}_i} (H_{ij} V_j) + K_i C_i - V_i A]x \tag{3.21}$$

Finally, to ensure the exponential stability of the observer, we can adopt the following assertions and assumptions.

- The matrix $\sum_{j \in \mathcal{N}_i} H_{ij}$ must be Hurwitz.
- The matrix $V_i = I_n$, therefore, $I_n = \sum_{j \in \mathcal{N}_i} M_{ij}$.
- The term multiplied by x in equation 3.21 is canceled, hence, $A - K_i C_i = \sum_{j \in \mathcal{N}_i} H_{ij}$.

The problem reduces itself to finding the matrices H_{ij}, M_{ij} and K_i which respect the assumptions defined previously. In order to achieve this, we will transform this observation problem into a decentralized control that is well documented [27], [28].

Consider the matrix $H = [H_{ij}]$ that can be written is as follows.

$$H = I_m \otimes A + \sum_{i=1}^{m} \sum_{j \in \mathcal{N}_i} B_i F_{ij} C_{ij} \tag{3.22}$$

With $B_i = u_i \otimes I_n$ (u_i is the i^{th} unit vector), $C_{ij} = b_{ij} \otimes I_n$ (b_{ij} is the column of the transpose matrix of the graph $\boldsymbol{G}$ corresponding to the arc from j to i), $C_{ii} = C_i B_i^T$, and finally $F_{ii} = -K_i$ and $F_{ij} = H_{ij}$, for $j \in \mathcal{N}_i$.

Finally, we can define the following multi-agent system, to which an output feedback control $u_{ij} = F_{ij} y_{ij}$ is applied.

$$\begin{cases} \dot{x} = (I_m \otimes A)x + \sum_{i=1}^{m} \sum_{j \in \mathcal{N}_i} B_i u_{ij} \\ y_{ij} = C_{ij} x \end{cases} \tag{3.23}$$

Choosing F_{ij} that will stabilize the multi-agent system defined above (3.23) is equivalent to finding H_{ij} and K_i for which the observation error is exponentially stable.

The easiest approach to solve this decentralized stabilization problem is to find a gain matrix F_{ij} for which one channel of the system defined in equation 3.23 is controllable and observable [28]. According to [24], the distributed observer is constructed in the following way:

1. Choose the matrices M_{ij} that verifies $I_n = \sum_{j \in \mathcal{N}_i} M_{ij}$.
2. Choose F_{pq} so that, for $p \in \{1, \ldots, m\}$ and $q \in \mathcal{N}_p$, (H, B_p) is controllable with a controllability index equal to the number of agents, and (C_{pq}, H) is observable. For simplicity, we will suppose that $p = m$.
3. Build the following system by increasing the size of the estimator by $m - 1$.

$$\bar{H} = \begin{bmatrix} H & 0 \\ 0 & I_{(m-1)\times(m-1)} \end{bmatrix} \tag{3.24}$$

$$\bar{U} = \begin{bmatrix} U_m & 0 \\ 0 & I_{(m-1)\times(m-1)} \end{bmatrix} \tag{3.25}$$

$$\bar{C} = \begin{bmatrix} C_{mq} & 0 \\ 0 & I_{(m-1)\times(m-1)} \end{bmatrix} \tag{3.26}$$

4. Ensure that $(\bar{H} + \bar{U}K\bar{C})$ is Hurwitz, with:

$$K = \begin{bmatrix} K_0 & K_1 & \dots & K_i & \dots & K_m \end{bmatrix} = \begin{bmatrix} -F_{00} & -F_{11} & \dots & -F_{ii} & \dots & -F_{mm} \end{bmatrix}$$

3.4 Simulations

We decided to put our observers to the test on a 2-chains integrator system composed of two agents that can communicate depending on the undirected graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, a_{e \in \boldsymbol{E}})$ presented in figure 3.1.

Figure 3.1: The communication graph.

The state-space of the 2-chain integrator system is defined in equation 3.27.

$$\dot{x} = \begin{bmatrix} -2 & 0 \\ 0 & -3 \end{bmatrix} x + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u \tag{3.27}$$

And we chose the output matrices defined in equation 3.28 to ensure the jointly observability of the system.

$$\begin{aligned} C_1 &= \begin{bmatrix} 1 & 0 \end{bmatrix} \\ C_2 &= \begin{bmatrix} 0 & 1 \end{bmatrix} \end{aligned} \tag{3.28}$$

This multi-agent system is stable but each agent on its own cannot ensure the observability of the plant; each of them have an observability index of 1.

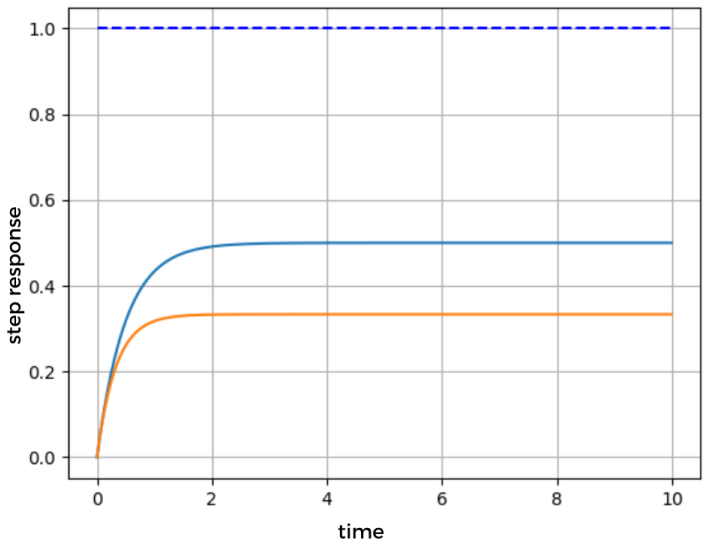

Figure 3.2: The step response of the system defined in 3.27

In order to start the simulations, we set the initial values of the estimates $\hat{x}$ to randomly vary from -3 to 3 for all the agents and implement all three observers earlier defined.

First, we start by simulating the chosen system without the addition of noise on the three observers provided before. We will make a comparison between the different systems based on three criteria:

- The settling time to a step $J_1 = t(\text{error} < 10^{-2})$ (time after which the observation error becomes lower than 10^{-2}).
- The vector containing the integrals of the square of the observation error of each agent

$$J_2 = \left[\int_0^{t_{max}} ||y_1 - \hat{y}_1||_2^2 \quad \dots \quad \int_0^{t_{max}} ||y_i - \hat{y}_i||_2^2 \quad \dots \quad \int_0^{t_{max}} ||y_m - \hat{y}_m||_2^2\right]$$

- The vector containing the observation error of each agent at each time:

$$J_3(t) = \left[||x_1(t) - \hat{x}_1(t)||_2^2 \quad \dots \quad ||x_i(t) - \hat{x}_i(t)||_2^2 \quad \dots \quad ||x_m(t) - \hat{x}_m(t)||_2^2\right]$$

3.4.1 The distributed Luenberger observer (DLO)

The observer was implemented in Python using a package we have implemented (see chapter 7 for more details). The parameters L_i and M_i were chosen using the equations 3.7 and 3.6, and applying the pole placement technique with the eigenvalues of $A_{iO} - L_{id}C_{iO}$ for $i \in \{1, 2\}$ chosen as $\lambda_{L_1} = -1.5$ and $\lambda_{L_2} = -2$ (see equation 3.6). We took $\gamma = 6$ and took the values of both k_i to 1, for $i \in \{1, 2\}$.

The results are illustrated in figure 3.3, with the estimates in colored solid curves and the real values in blue dashed curves.

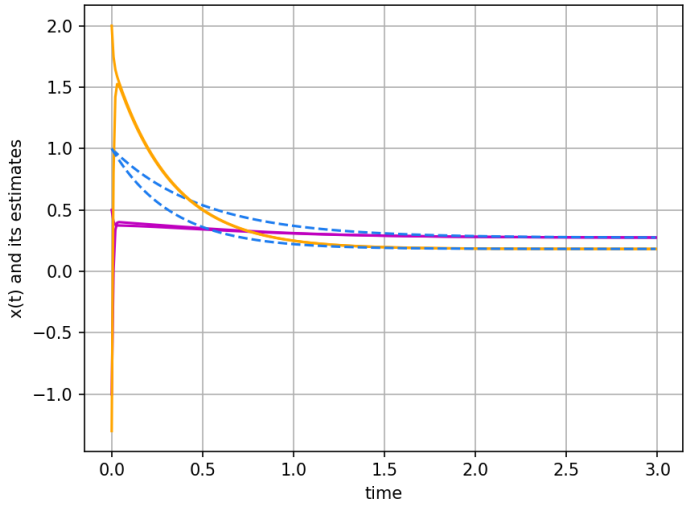

Figure 3.3: Trajectories of local estimates using the distributed Luenberger observer (DLO).

For this observer, we obtained a settling time to a step equal to $J_1 = 2,12s$.

We have illustrated the values of our criteria J_2 and J_3 in the following figures.

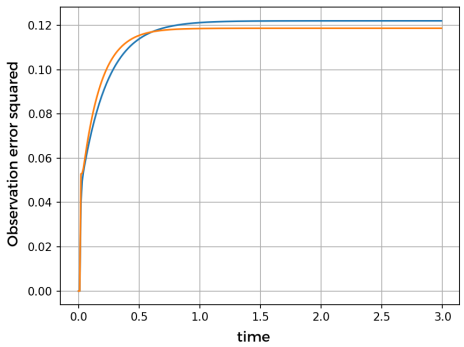

Figure 3.4: The integral square observation error J_2 using the DLO

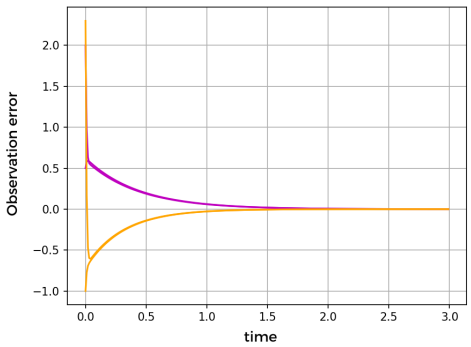

Figure 3.5: The observation error J_3 using the DLO

Table 3.1: DLO comparison criteria

settling time (J_1) (s)	**Integral square error** ($\|\|J_2\|\|_\infty$)	**Steady-state error** ($J_3(t_{max})$)	**Overshoot**
1.89	0.122	0	0

3.4.2 The distributed Finite-Time Observer (DFTO)

The observer was also implemented in Python using the same library as before. We took $v = -0.02$ and obtained $\gamma = 1$ and $\beta = 0,98$. As for L_i, we chose the same eigenvalues as in the previous simulation $\lambda_{L_1} = -1.5$ and $\lambda_{L_1} = -2$.

The results are illustrated in the figure shown below 3.9, with the estimates in colored solid curves and the real values in blue dashed curves.

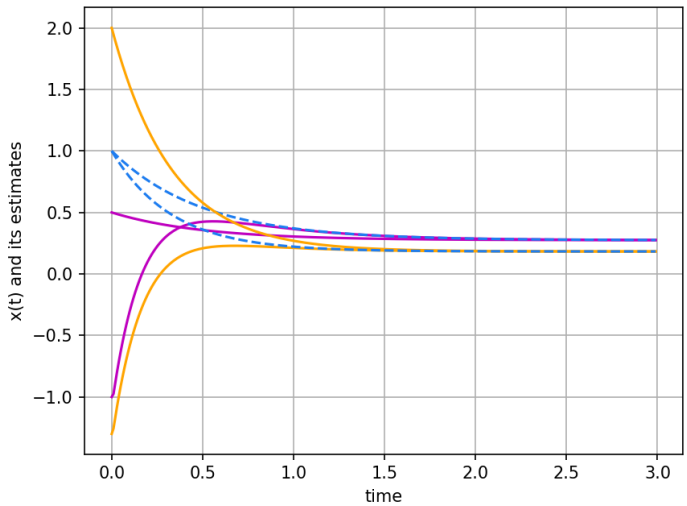

Figure 3.6: Trajectories of local estimates using the distributed Finite-Time Observer (DFTO).

For this observer, we obtained a settling time to a step equal to $J_1 = 1,88s$. We illustrated criateria J_2 and J_3 in the following figures.

Table 3.2: DFTO comparison criteria

settling time (J_1) (s)	**Integral square error** ($\|\|J_2\|\|_\infty$)	**Steady-state error** ($J_3(t_{max})$)	**Overshoot**
2.13	0.533	0	0

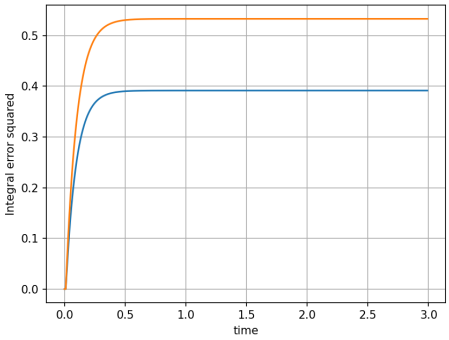

Figure 3.7: The integral square observation error J_2 using the DFTO

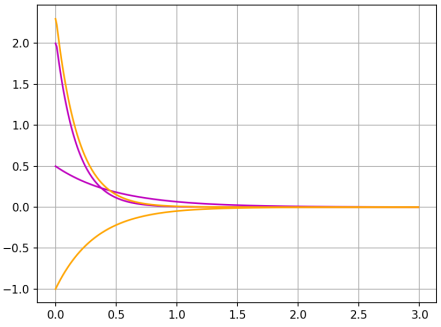

Figure 3.8: The observation error J_3 using the DFTO

3.4.3 The Distributed Observer for Time-Invariant linear systems (DOLTI)

The parameters of this final observer were defined using the equations defined in the last sections and by implementing a decentralized control on the system 3.22.

The results are illustrated in figure 3.9. The estimates are in solid colors, the augmented state in solid green, and $x(t)$ is in blue dashed curves

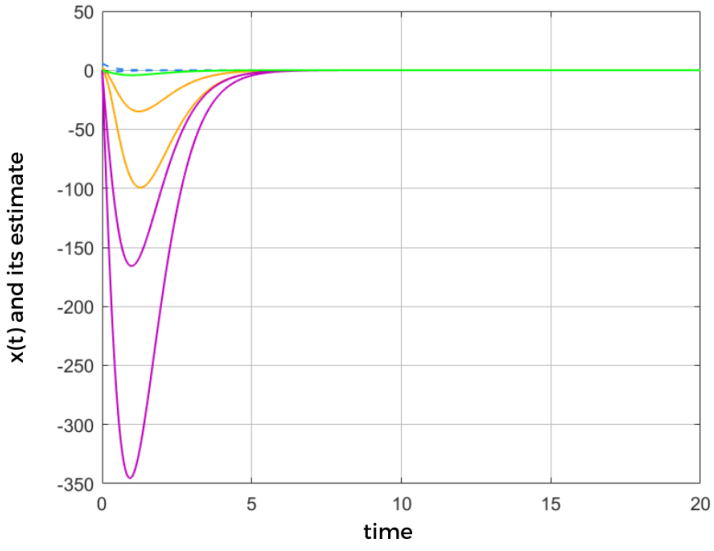

Figure 3.9: Trajectories of local estimates using the Distributed Observer for Time-Invariant linear systems (DOLTI)

We obtained a settling time to a step of $J_1 = 7,14s$ for this observer.

The estimations' highest values range from 30 to 350 units, implying that the distributed LTI estimator's observation error is unquestionably larger than that of the other observers.

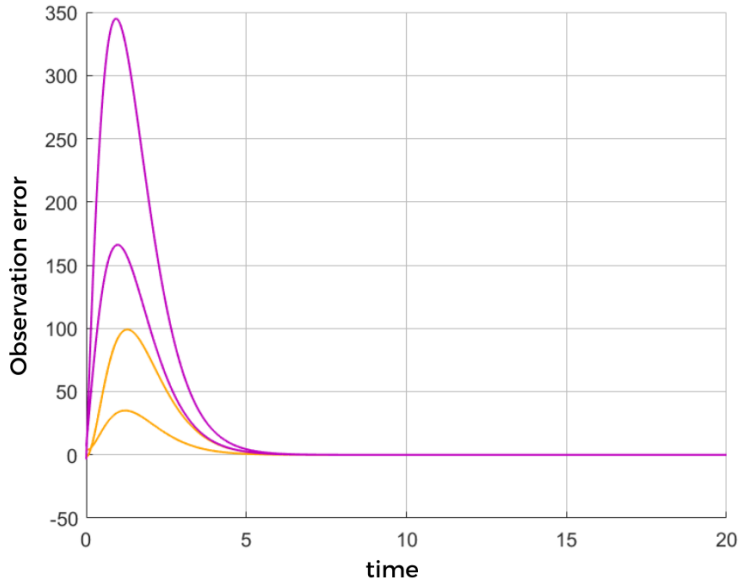

Figure 3.10: The observation error J_3 of DOLTI

Table 3.3: DOLTI comparison criteria

settling time (J_1) (s)	**Integral square error** ($\|\|J_2\|\|_\infty$)	**Steady-state error** ($J_3(t_{max})$)	**Overshoot**
7.14	–	0	350

3.4.4 Comparison between the different methods

3.4.4.1 Comparison criteria

According to the first criterion J_1, the DLO is the fastest observer; it has the shortest one-step settling time, followed by the DFTO, and finally the DOLTI, which is significantly slower than the other two observers. Furthermore, the DLO produced the most satisfactory results by far according to the second J_2 and third criterion J_3 (see figures 3.5, 3.4, 3.8, 3.7, and table 3.4).

Table 3.4: Results of the comparison of the different criteria.

	DLO	DFTO	DOLTI
Settling time J_1 (s)	**1.89**	2.13	7.14
Overshoot	**0**	**0**	350
Integral square error $\|\|J_2\|\|_\infty$	**0.122**	0.533	–

3.4.4.2 Robustness

We began by assessing the robustness of the observers to parametric noise. When parametric noise is introduced, all three observers converge, indicating that they are reliable concerning model inaccura-

cies.

To compare these observers, we decided to evaluate the first criterion J_1 in the presence of parametric errors. The results are presented in figures : 3.11, 3.12 and 3.13.

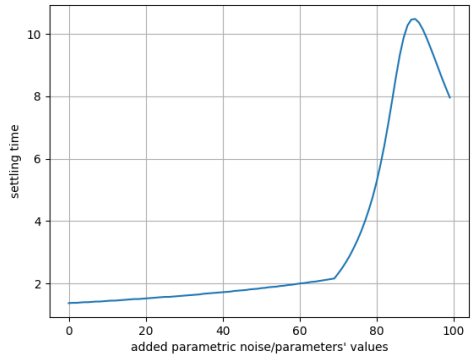

Figure 3.11: The settling time to a step versus the added parametric noise for DLO

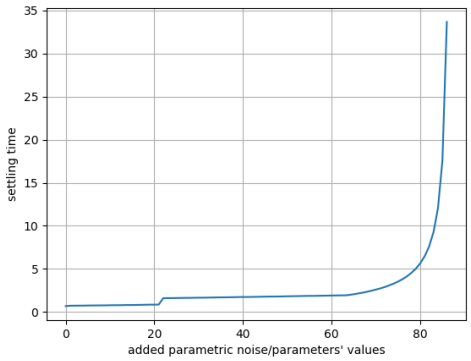

Figure 3.12: The settling time to a step versus the added parametric noise for DFTO

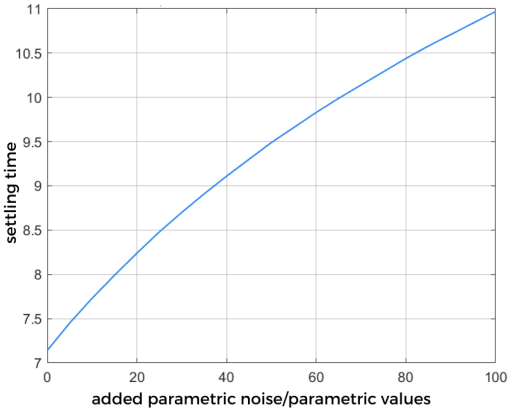

Figure 3.13: The settling time to a step versus the added parametric noise for DOLTI

Despite the additional disturbances, the distributed Luenberger observer has the shortest settling times. Up until the parametric error equals 80% of the parameter values, the DFTO's results are comparable to those of the DLO.

With a settling time beginning at 7s, the final observer produced the least desirable results. However, the DOLTI always manages to converge without any steady state error unlike the DLO (see figure 3.14).

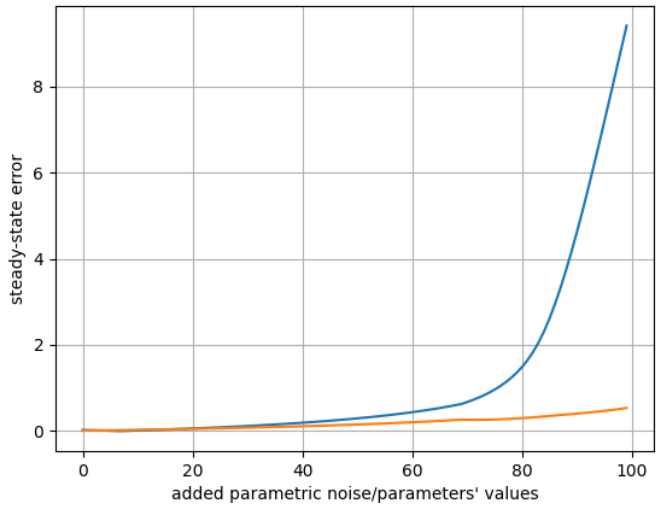

Figure 3.14: The steady-state error versus parametric noise when implementing the DLO

In conclusion, the DOLTI is the most robust observer since its settling time does not increase exponentially as a function of the parametric errors, unlike the other two observers.

However, it is important to note that, while the DOLTI can manage large parametric errors, it does so at the expense of large input values, as shown by plotting the overshoot obtained by the estimates against the proportion of parametric noise in figure 3.15. This type of overshoot is unbearable in practice.

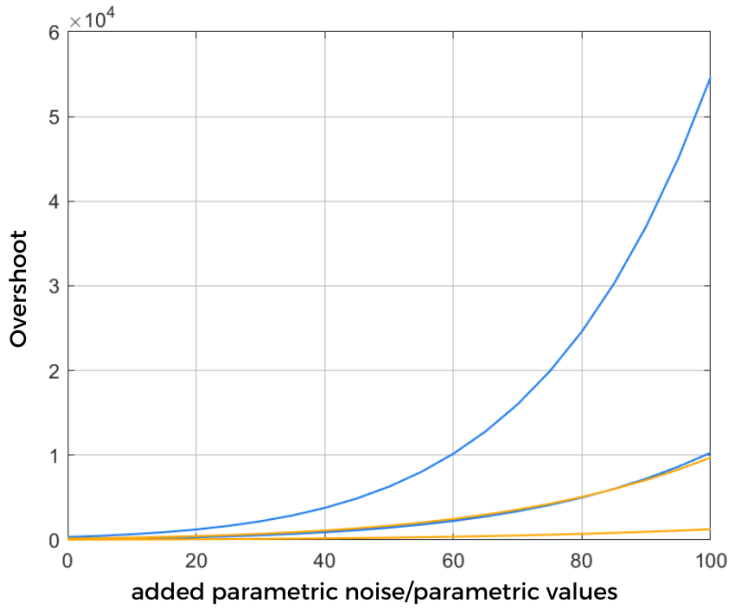

Figure 3.15: The overshoot of the estimates using the DOLTI versus the parametric noise

Furthermore, unlike the DOLTI, both the DLO and the DFTO are robust to output noise. The observers' estimates oscillate; this problem can be easily addressed by using an averaging filter.

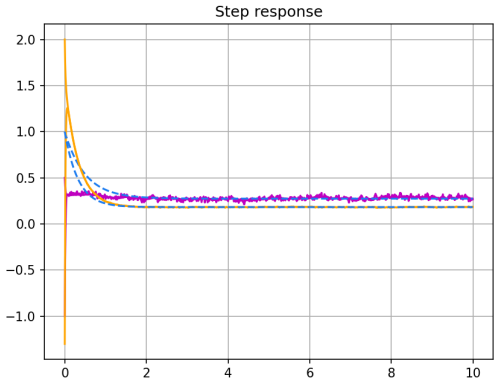

Figure 3.16: Trajectories of local estimates using the DLO

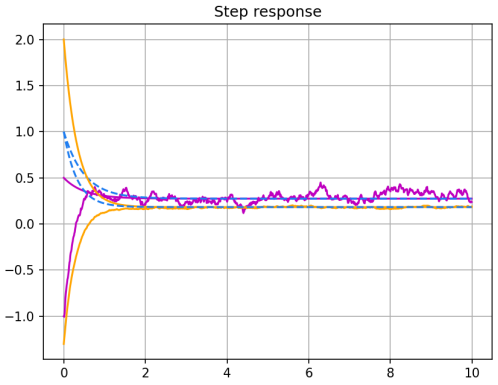

Figure 3.17: Trajectories of local estimates using DFTO.

Figures 3.16 and 3.17 show that the DLO's estimation yields a less noisy outcome than the DFTO estimation.

3.5 Conclusion

There are numerous approaches for observing a system of agents using distributed techniques. It becomes a question of knowing our aims when deciding which one to use in a given situation. Our goal can be to reduce the information sent between the agents if we have a large network for instance, or to optimize the performance of the estimation, in this case, we have seen that the DLO is the best choice for a linear system. In conclusion, deciding on the best distributed observer for a specific situation boils down to goals and trade-offs. .

CHAPTER 4

DISTRIBUTED OBSERVATION OF A MULTI-AGENT SYSTEM

4.1 Introduction

Multi-agent systems, in which each agent can make decisions and act autonomously, are becoming increasingly popular among scientists. The MAS can obtain the desired behavior by accumulating the various actions performed by each agent.

This field has spawned several control problems, including state estimation based on coupled or relative measurements between agents. This one is a continuation of prior research on distributed plant estimation.

In this chapter, we will cover existing approaches and algorithms for the distributed observation of a multi-agent system.

4.2 Problem statement

Throughout this chapter, we are interested in a **fixed** network of m agents which can communicate with their neighbors. The connections between the agents are defined using a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. We won't consider the agent to be its own neighbor; that is, $\boldsymbol{G}$ has no self-arcs. The agents' systems are modeled using a **linear time-invariant** state-space representation:

$$\begin{cases} \dot{x}_i = A_i x_i + B_i u \\ y^i = C^i x_i \end{cases} \tag{4.1}$$

With $x_i \in \mathbb{R}^{n_i}$, $y^i \in \mathbb{R}^{q_i}$, and $u_i \in \mathbb{R}^{r_i}$, the input of the system, with $i \in \{1, 2, \ldots, m\}$.

The goal is to define a suitable group of m observers that can jointly estimate the states of each agent using the coupled measurements between each other [4].

We can distinguish two categories of this problem:

- the observation of a group of **homogeneous** agents; that is, all the agents are defined using the same state-space model. Mathematically, $A_0 = A_1 = \cdots = A_i = \cdots = A_m$ and $B_0 = B_1 = \cdots = B_i = \cdots = B_m$.

- the observation of a group of **heterogeneous** agents; that is, the agents can have different state-space models.

Agents may not be aware of the structures of agents that are not in their general neighborhood if the system is heterogeneous. Hence, designing a group of observers that can perform estimation with this constraint in mind would be a significant contribution. Therefore, this objective is included to the list of goals specified in the previous chapter, section 3.2.

4.3 MAS state estimation

Let us examine how to convert a group of independent agents into a global system (MAS) and use the algorithms specified in the previous chapter.

4.3.1 From a multi-agent system to a plant

Since there is a substantial literature on employing multiple output matrices to solve the plant observation problem, we will rewrite the group of agents as one global plant.

Consider $x = \begin{bmatrix} x_0^T & x_1^T & \dots & x_m^T \end{bmatrix}^T$ and the system defined in 4.1.

$$\dot{x} = \begin{bmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \vdots \\ \dot{x}_m \end{bmatrix} = \begin{bmatrix} A_1 & 0 & \dots & 0 \\ 0 & A_2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & A_m \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} B_1 & 0 & \dots & 0 \\ 0 & B_2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & B_m \end{bmatrix} \begin{bmatrix} u^1 \\ u^2 \\ \vdots \\ u^m \end{bmatrix} \tag{4.2}$$

Therefore, we can write.

$$\dot{x} = Ax + BU \tag{4.3}$$

If the MAS is homogeneous, is equivalent to:

$$\dot{x} = (I_{m\times m} \otimes A_0)x + (I_{m\times m} \otimes B_0)U \tag{4.4}$$

with $U = \begin{bmatrix} u^{1T} & u^{2T} & \dots & u^{mT} \end{bmatrix}^T$, and A and B defined by the correspondence between equations 4.3 and 4.2.

In addition to that, it is possible to write:

$$y^i = \begin{bmatrix} 0 & \dots & 0 & C^i & 0 & \dots & 0 \end{bmatrix} x = (u_i \otimes C^i)x \tag{4.5}$$

with $u_i \in \mathbb{R}^m$ being the i^{th} unit vector.

Consider z_i, the coupled measurement between the agents, we write $z_i = \begin{bmatrix} C_0^i & \dots & C_i^i & \dots & C_m^i \end{bmatrix} x$.

Consider $y_i = \begin{bmatrix} y^{iT} & z_i^T \end{bmatrix}^T$.

$$y_i = \begin{bmatrix} y^i \\ z_i \end{bmatrix} = \begin{bmatrix} 0 & \dots & C_i & \dots & 0 \\ C_0^i & \dots & C_i^i & \dots & C_m^i \end{bmatrix} x \tag{4.6}$$

Let's put.

$$C_i = \begin{bmatrix} 0 & \dots & C_i & \dots & 0 \\ C_0^i & \dots & C_i^i & \dots & C_m^i \end{bmatrix} \tag{4.7}$$

Finally, we can write it as a MAS.

$$\begin{cases} \dot{x} = Ax + Bu \\ y_i = C_i x \end{cases} \tag{4.8}$$

Finally, we can use the algorithms defined in the previous chapter as long as we write the DSS in this format. The disadvantage of this approach is that carrying out an estimation requires knowledge of all the mathematical models of the agents. In addition, this method is computationally intensive. However, it can be sped up if the graph is sufficiently sparse and this is what we will examine in the section 4.3.2

Example

Let us consider the following example with three different agents connected in a topology of a cyclic graph (see figure 4.1). Let the individual dynamics of the agents be described by the state-space models defined in the following equations.

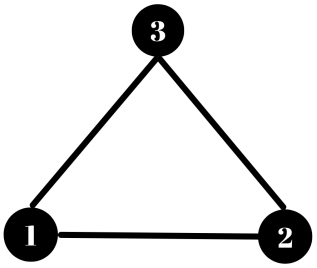

Figure 4.1: The cyclic graph

$$\begin{cases} \dot{x}_1 = \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix} x_1 + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u_1 \\ \\ y^1 = \begin{bmatrix} 1 & 0 \end{bmatrix} x_1 \\ \\ z_1 = \begin{bmatrix} 1 & 0 \end{bmatrix} x_2 + \begin{bmatrix} 2 & -1 \end{bmatrix} x_3 \end{cases} \tag{4.9}$$

$$\begin{cases} \dot{x}_2 = \begin{bmatrix} 4 & 0 \\ -2 & 1 \end{bmatrix} x_2 + \begin{bmatrix} 1 \\ 4 \end{bmatrix} u_2 \\ y^2 = \begin{bmatrix} 1 & 1 \end{bmatrix} x_2 \\ z_2 = \begin{bmatrix} 0 & 1 \end{bmatrix} x_1 + \begin{bmatrix} -1 & 0 \end{bmatrix} x_3 \end{cases} \tag{4.10}$$

$$\begin{cases} \dot{x}_3 = \begin{bmatrix} 1 & 0 \\ 0 & -2 \end{bmatrix} x + \begin{bmatrix} -1 \\ 1 \end{bmatrix} u_3 \\ y^3 = \begin{bmatrix} 1 & -1 \end{bmatrix} x_3 \\ z_1 = \begin{bmatrix} -2 & 1 \end{bmatrix} x_1 + \begin{bmatrix} 0 & -1 \end{bmatrix} x_2 \end{cases} \tag{4.11}$$

The first step is to define the output matrices of each agent. As we have seen, this is done by concatenating the output of the agents' systems with the coupled measurements.

We define x as the concatenation of all the states x_1, x_2, and x_3.

$$y_1 = \begin{bmatrix} y^1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = C_1 x \tag{4.12}$$

$$y_2 = \begin{bmatrix} y^2 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = C_2 x \tag{4.13}$$

$$y_3 = \begin{bmatrix} y^3 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = C_3 x \tag{4.14}$$

Finally, we define the state-space model of the multi-agent system.

$$\begin{cases} \dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = A \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + B \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \\ y_i = C_i x \end{cases} \tag{4.15}$$

With $i \in \{1, 2, 3\}$, and matrices A and B defined as follows.

$$A \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.16}$$

4.3.2 Cooperative Estimation of Multi-agent Systems With Coupled Measurements

We can estimate the states of a multi-agent system using a cooperative observer that relies on the state and parameters of the agent and its neighbors [4]. Each agent has access to the state-space model of its neighbors' and the measurements that are relative to them.

$$\begin{cases} \dot{x}_i = A_i x_i + B_i u \\ y_i = \sum_{j \in (i \cup \mathcal{N}_i)} C_{ji} x_j \end{cases} \tag{4.17}$$

The observer introduced in this section is a reduced size Luenberger observer; with each agent just estimating their own state and that of their neighbors. The state-space models and the agent's estimated states are the only pieces of information that need to be transmitted. In summary, each agent i estimates its own states and its neighbors' $\hat{x}^i_{j \in (i \cup \mathcal{N}_i)}$ using their neighbors' estimations $\hat{x}_i^{j \in (i \cup \mathcal{N}_i)}$ [4].

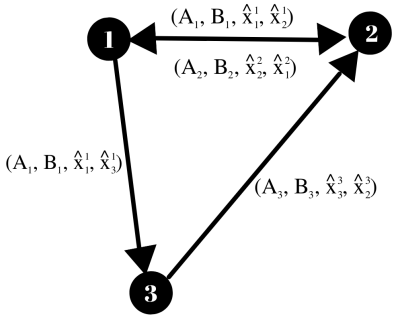

Figure 4.2: The information sent between the agents

Consider the corresponding estimation vector made by the agent i:

$$\hat{x}^i = \begin{bmatrix} \hat{x}^i_i \\ \hat{x}^i_{n_0} \\ \vdots \\ \hat{x}^i_{n_j} \\ \vdots \\ \hat{x}^i_{n_{m_i}} \end{bmatrix} \tag{4.18}$$

with $n_j \in \mathcal{N}_i$, $j \in \{1, 2, \dots, m_i\}$ and $m_i = Card(\mathcal{N}_i)$.

We apply the Distributed Luenberger observer on the state vectors defined in 4.18.

$$\dot{\hat{x}}^i = \begin{bmatrix} \dot{\hat{x}}^i_i \\ \dot{\hat{x}}^i_{n_0} \\ \vdots \\ \dot{\hat{x}}^i_{n_{m_i}} \end{bmatrix} = \begin{bmatrix} \hat{A}^i_i & 0 & \dots & 0 \\ 0 & \hat{A}^i_{n_0} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & \hat{A}^i_{n_{m_i}} \end{bmatrix} \begin{bmatrix} \hat{x}^i_i \\ \hat{x}^i_{n_0} \\ \vdots \\ \hat{x}^i_{n_{m_i}} \end{bmatrix} + L^i(y_i - \sum_{j \in (i \cup \mathcal{N}_i)} C_{ji}\hat{x}_j) + K^i \begin{bmatrix} \hat{x}^{n_0}_{n_0} - \hat{x}^i_{n_0} \\ \vdots \\ \hat{x}^{n_{m_i}}_{n_{m_i}} - \hat{x}^i_{n_{m_i}} \end{bmatrix}$$

$$\dot{\hat{x}}^i = A^i\hat{x}^i + L^i(y_i - \sum_{j \in (i \cup \mathcal{N}_i)} C_{ji}\hat{x}_j) + K^i \begin{bmatrix} \hat{x}^{n_0}_{n_0} - \hat{x}^i_{n_0} \\ \vdots \\ \hat{x}^{n_{m_i}}_{n_{m_i}} - \hat{x}^i_{n_{m_i}} \end{bmatrix} \tag{4.19}$$

With $i \in \{1, 2, \dots, m\}$.

L^i and K^i are gain matrices that will determine the convergence rate of the observer.

The optimal control method H_∞ is used to select the gain matrices, ensuring high performance in terms of the given criterion. Consider the estimation error e^i defined in the following equation.

$$e^i = \begin{bmatrix} x_i - \hat{x}^i_i \\ x_{n_0} - \hat{x}^i_{n_0} \\ \vdots \\ x_{n_j} - \hat{x}^i_{n_j} \\ \vdots \\ x_{n_{m_i}} - \hat{x}^i_{n_{m_i}} \end{bmatrix} \tag{4.20}$$

The goal is to obtain a quadratically finite estimation error. Therefore, we choose to minimise the error according to the following criterion [4].

$$\sum_{i=0}^{m} \int_0^\infty e^{iT} W^i e^i \, dx \tag{4.21}$$

With W^i weighting matrices that are positive semi-definite.

Using a well-defined Lyapunov function ($V = \sum_{i=0}^{m} e^{iT} P^i e^i$)), it was proven that there exist optimal gains L^i and K^i that will allow the observation error to converge to 0 exponentially in the absence of disturbances and noises [4] .

4.4 Conclusion

As in the case of estimating a plant using a distributed observer, there are several ways to estimate the states of a multi-agent system. Building on what we saw in the previous chapter, These techniques

have been improved to include cooperative measurements and maintain convergence while minimizing the dimensions of each observer and the information transferred between agents. In this regard, the cooperative observer is a remarkable improvement compared to the observers in the previous chapter, as it yields much-reduced observers when the graph is sparse.

.

CHAPTER 5

SENSOR FAULT CORRECTION USING DISTRIBUTED OBSERVERS

5.1 Introduction

Sensors have become an essential aspect of modern technology with the introduction of robotics, IoT and edge AI. These systems receive data from the environment via sensors and send it to a processing unit that extracts important information and takes action. Since the performance of these systems is highly dependent on the data collected, sensor corruption could have adverse implications [29]. This problem can have an even larger impact on a multi-agent system, as a single agent can cause the entire plant to malfunction. However, we know that the focus of a multi-agent system is not on the individual systems but on the group's ability to accomplish the task for which it was designed. Indeed, agents do not need to be independently reliable, but the system as a whole does. In this chapter, we will look at cooperative fault correction in multi-agent systems, and we will focus primarily on sensor failures.

5.2 Problem statement

We are interested in a *homogeneous* network of m agents that can communicate with their neighbors. An **undirected** graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ with no self-arcs, is used to define the connections among the agents. The systems are modeled using a **LTI** state-space representation:

$$\begin{cases} \dot{x}_i = Ax_i + Bu \\ y^i = C^i x_i \end{cases} \tag{5.1}$$

With $x_i \in \mathbb{R}^n$, and $i \in \{1, \ldots, m\}$.

The agents' models are such so that the system is observable. However, some lose one or more of their sensors functionality, rendering their system unobservable. Therefore, we need to find a group of agents and observers that can use relative measurements to estimate the states of the malfunctioning systems.

Consider a graph that shows how a group of agents is connected. Assume that L agents are defaulting; that is, one of the elements of their output matrix that was previously different from 0 has now been

annulled (see equation 5.2 and 5.3). Then we would have to figure out the following:

- How many agents do we need to combine in a graph to estimate their states including the defaulting ones? What is the structure of that graph?
- How to estimate the states of the defaulting agents using the least possible number of agents?
- What are the coupled or relative measurements we would need to have to perform the estimation?

The output matrix before the fault has occurred:

$$C^i = \begin{bmatrix} C_0^i & \dots & C_i^i \neq 0 & \dots & C_n^i \end{bmatrix} \tag{5.2}$$

The output matrix after the fault has occurred:

$$C^i = \begin{bmatrix} C_0^i & \dots & C_i^i = 0 & \dots & C_n^i \end{bmatrix} \tag{5.3}$$

Note that if a multi-agent system contains only observable agents; then it is jointly observable.

5.3 Solution overview

5.3.1 Summary

To solve this problem, we will expand on what we have seen in previous chapters, but first, we must identify the malfunctioning agents (this problem is sufficiently well documented [29] [30] [31]). The identification of failed sensors in an agent is done through an algorithm implemented in the agent's CPU, which is intrinsically dependent on it. Given that all agents have observable systems, to begin with, we are seeking means to find defective agents in the previously specified multi-agent system. As a result, it is sufficient to check each agent's observability and thus, identify those that are not observable. The latter systems are thought to be defective. Indeed, if the loss of a sensor does not render the system unobservable, it will be enough to estimate the states that are not directly accessible using a simple observer as we know it.

Finally, each agent communicates its observability index to its neighbors, therefore ensuring that all agents know which of their neighbors are deficient.

The next step is to estimate the states of the defaulting agents we identified beforehand. We will employ the previously stated distributed observation methods. However, we must first determine the relative measurements needed to design a jointly observable system (a necessary condition for applying the solutions introduced in Chapter 3), as well as the minimal subgraphs of $\boldsymbol{G}$ to which we will apply these algorithms.

5.3.2 Subgraphs definition

The sub-graphs are chosen locally; each defaulting agent searches for a group of agents in its neighborhood that includes at least one fault-free agent. The latter will take care of the relative measurements, and this set of agents will perform a distributed observation to estimate their states. The remaining fault-free agents will be considered as a group on their own.

Each defaulting agent i will execute the algorithm defined below (algorithm 1) on its CPU to find the sub-graph $\boldsymbol{G}_i$ of $\boldsymbol{G}$ that will be used to estimate its states. The defaulting systems apply algorithm 2; they look in their neighborhood for a non-faulty agent to group with.

Let us consider a multi-agent system with m agents, n_d of which have failed sensors. The relations of the MAS are defined using a graph $\mathbf{G = (E, V)}$. The set of edges is associated to a set of integers of same size $\mathbf{E} \leftrightarrow \{1, 2, \ldots, n\}$. The goal is to find a set of subgraphs $\{G_{d_1}, G_{d_2}, \ldots G_{d_{n_d}}\}$, one for each defaulting agents. We will denote the set of faulty agents E_D and the set of non-faulty agents E_{ND}. The group of edges E_i corresponding to the defaulting agent i is defined as follows.

$$E_i = \{i, min(E_{ND} \cup \mathcal{N}_i)\} \tag{5.4}$$

We will later use this resulting subgraph to jointly estimate the states of the defective agents.

Algorithm 1: Algorithm for clustering the defaulting agents

Input: i `/* index of the current agent. */`
ND_i `/* the set non-defaulting agents in the neighborhood of` i `*/`
Output: $\boldsymbol{G}_i$
Function `Find_minimal_sub_graph_D_agents(`i, ND_i`):`
  $\boldsymbol{G}_i \longleftarrow graph()$ `/* initialize empty graph. */`
  if ***Length***$(ND_i) \neq 0$ **then**
    $\boldsymbol{G}_i$.agents $\longleftarrow$ [i,ND$_i$[0]]
    $\boldsymbol{G}$**.send_to_neighbors(**$\boldsymbol{G}_i$.agents)
  else
    Print (*"It is impossible to jointly estimate the states of this multi-agent system."*)
    Break;
  end
  return $\boldsymbol{G}_i$;
End Function

Non-defaulting agents i, on the other hand, receive the groups formed by their defaulting neighbors $E_{\mathcal{N}_i} = \{E_{d_j}, \ldots, E_{d_k}, \ldots, E_{d_l}\}$ with $\{d_j, \ldots, d_k, \ldots, d_l\}$ the defaulting neighbors of agent i. If $Card(E_{\mathcal{N}_i}) > 1$, that is, they are part of more than one group, they establish a new subgraph that includes themselves and the agents that chose them to be a part of their group. Therefore, to find the subgraphs, we simply group the different sets sent by the neighbors and remove the duplicates from the resulting set.

Algorithm 2: Algorithm for clustering the non-defaulting agents

Input: i, D_i
Output: $\boldsymbol{G}_i$
Function `Find_minimal_sub_graph_ND_agents():`
  list_G_i_neighbors = $\longleftarrow$ $\boldsymbol{G}$**.receive_from_neighbors**()
  if $i \in$ *list_G_i_neighbors* **then**
    $\boldsymbol{G}_i \longleftarrow$ **reduce_set**(*list_G_i_neighbors*)
  else
    $\boldsymbol{G}_i \longleftarrow$ [i]
  end
  return $\boldsymbol{G}_i$;
End Function

If these non-failing agents are part of a group, their states are estimated using a distributed observer. Otherwise, the estimation is done centrally by the agent itself.

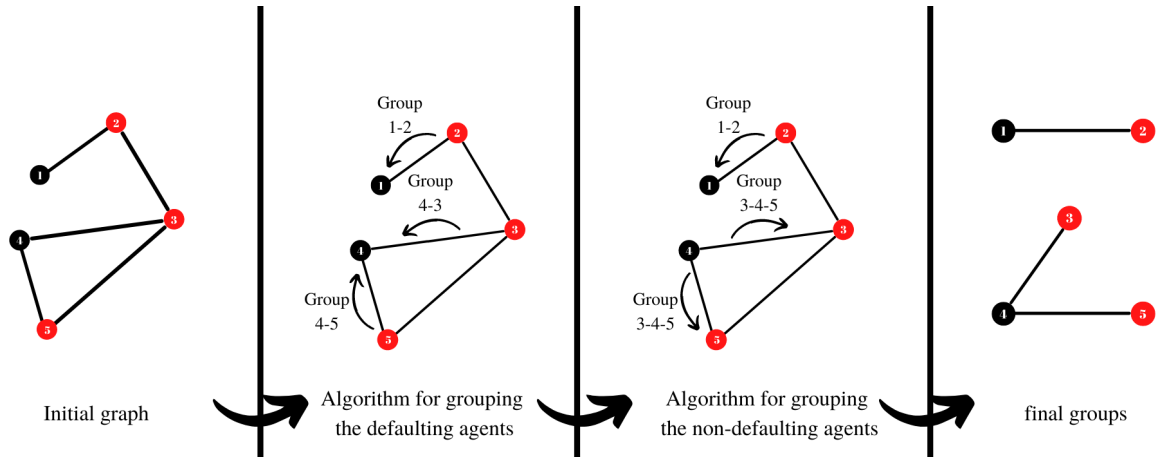

Figure 5.1: The newly formed sub-graphs

5.3.3 Output matrices generation

This section is devoted to the algorithm to find the relative measurements required to obtain a jointly observable system. To do so, we use the transformation matrices T_i of the staircase observability form of each agent's system. The last rows of this matrix represent the non-observable states. Hence we can raise the system's observability index by measuring them relative to their non-defective neighbors.

Consider the staircase transformation matrix T_i of a defective agent i. The agent has an observability index of v_i which means that the first v_i lines of T_i represent the states combinations that can be estimated or measured.

Applying the transformation matrix T_i on the system, we obtain $\bar{x}_i = T_i x_i = \begin{bmatrix} \bar{x}_{iO}^T & \bar{x}_{i\bar{O}}^T \end{bmatrix}^T$, and $y^i = \begin{bmatrix} C_O & 0 \end{bmatrix} T_i x_i$ (see equation 3.4).

$$T_i x_i = \left[\begin{array}{ccc} T_{11}^i & \dots & T_{1n}^i \\ \vdots & \ddots & \vdots \\ T_{v_i 1}^i & \dots & T_{v_i n}^i \\ \hline T_{(v_i+1)1}^i & \dots & T_{(v_i+1)n}^i \\ \vdots & \ddots & \vdots \\ T_{n1}^i & \dots & T_{nn}^i \end{array}\right] \left[\begin{array}{c} x_1^i \\ \vdots \\ x_{v_i}^i \\ \hline x_{v_i+1}^i \\ \vdots \\ x_n^i \end{array}\right] = \left[\begin{array}{c} \sum_{j=1}^n T_{1j}^i x_j^i \\ \vdots \\ \sum_{j=1}^n T_{v_i j}^i x_j^i \\ \hline \sum_{j=1}^n T_{(v_i+1)j}^i x_j^i \\ \vdots \\ \sum_{j=1}^n T_{nj}^i x_j^i \end{array}\right] = \left[\begin{array}{c} \bar{x}_O^i \\ \hline \bar{x}_{\bar{O}}^i \end{array}\right] \tag{5.5}$$

That means that the combination of states that can neither be measured nor estimated are $\bar{x}_{\bar{O}}^i = \begin{bmatrix} \sum_{j=1}^n T_{(v_i+1)j}^i x_j^i & \dots & \sum_{j=1}^n T_{nj}^i x_j^i \end{bmatrix}^T$. Therefore, these are the ones that should be measured relatively by another agent. Let's consider the matrices $\{T_{(v_i+1)}^i, \dots, T_n^i\}$ such that $T_l^i = \begin{bmatrix} T_{(v_i+1)1}^i & \dots & T_{(v_i+1)n}^i \end{bmatrix}$ is l^{th} line of the T_i.

The algorithm 3 generates the new output matrices C_i of the non-defaulting agents. It is done by concatenating the measurements of that agent (represented by matrix C^i) and the relative measurements of the unobservable states of the defaulting neighbors $\bar{x}_{\bar{O}}^j$ with $j \in \mathcal{N}_i$.

Consider a non-defaulting agent i connected to two agents $\{j, k\}$.

$$y_i = \begin{matrix} \\ \\ \\ \end{matrix} \begin{matrix} 1 & \dots & k & \dots & i & \dots & j & \dots & m \end{matrix} \begin{bmatrix} 0 & \dots & 0 & \dots & C^i x_i & \dots & 0 & \dots & 0 \\ 0 & \dots & \bar{x}^k_{\bar{O}} & \dots & -\bar{x}^i_{\bar{O}} & \dots & 0 & \dots & 0 \\ 0 & \dots & 0 & \dots & -\bar{x}^i_{\bar{O}} & \dots & \bar{x}^j_{\bar{O}} & \dots & 0 \end{bmatrix} \tag{5.6}$$

We can deduce the output matrix from equation 5.6, and $\bar{x}^i_{\bar{O}} = \begin{bmatrix} \sum_{j=1}^{n} T^i_{(v_i+1)j} x^i_j & \dots & \sum_{j=1}^{n} T^i_{nj} x^i_j \end{bmatrix}^T$.

$$y_i = \begin{matrix} \\ \\ \\ \end{matrix} \begin{matrix} 1 & \dots & k & \dots & i & \dots & j & \dots & m \end{matrix} \begin{bmatrix} 0 & \dots & 0 & \dots & C^i & \dots & 0 & \dots & 0 \\ 0 & \dots & T^k_{(v_k+1)\to n} & \dots & -T^k_{(v_k+1)\to n} & \dots & 0 & \dots & 0 \\ 0 & \dots & 0 & \dots & -T^j_{(v_j+1)} & \dots & T^j_{(v_j+1)\to n} & \dots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \tag{5.7}$$

With $T^i_{v_k \to n}$ the last $n - (v_k + 1)$ lines of the matrix T_i

Finally, the generated output matrix of the i non-defaulting agent that has two faulty neighbors ($\{j, k\}$) is deduced from equation 5.7.

$$C_i = \begin{matrix} \\ \\ \\ \end{matrix} \begin{matrix} 1 & \dots & k & \dots & i & \dots & j & \dots & m \end{matrix} \begin{bmatrix} 0 & \dots & 0 & \dots & C^i & \dots & 0 & \dots & 0 \\ 0 & \dots & T^k_{(v_k+1)\to n} & \dots & -T^k_{(v_k+1)\to n} & \dots & 0 & \dots & 0 \\ 0 & \dots & 0 & \dots & -T^j_{(v_j+1)} & \dots & T^j_{(v_j+1)\to n} & \dots & 0 \end{bmatrix} \tag{5.8}$$

Algorithm 3: Algorithm for generating the matrices C_i

Input: A, C^i, observability_index_list, m, D_i, max_iteration
Output: C_i
Function `Find_output_matrix`(A, C^i, *observability_index_list*, m, D_i, *max_iteration*)**:**
  $C_i \longleftarrow [\mathrm{u}_i \otimes C^i]$
  foreach *agent* $\in D_i$ **do**
    observability_index_agent = observability_index_i[agent]
    $C^i_{agent} \longleftarrow$ **zero_matrix_of_shape**((*size*(A)) - observability_index_agent, nbr_columns(C^i)))
    $T_i \longleftarrow$ **compute_staircase_transformation_matrix**(A,C^{agent})
    $C^i_{agent} \longleftarrow C^i_{agent} + T_i$ [observability_index_agent:, :]
    observability_index_agent
     $\longleftarrow$ (*rank*(**compute_observability_matrix**(A,[C^i, new_C_i]^T)) == size(A))
    it $\longleftarrow$ *it* + 1
    if *observability_index_agent* < *size*(A_i) **then**
      Print (*"It is impossible to find a set of output matrices* $\{C_{ij}\}$ *(With* $i \in \{1, 2, \ldots, m\}$ *and* $j \in \mathcal{N}_i$*) to jointly estimate the states of this multi-agent system."*)
      Break;
    end
    $C_i \longleftarrow [\mathrm{C}_i, \quad u_i \otimes (-C^i_{agent}) + u_{agent} \otimes (C^i_{agent})]^T$
  end
  return C_i;
End Function

5.3.4 General algorithm

Using the algorithms for finding the subgraphs and defining the output matrices, we can use the distributed observation techniques we have seen in chapter 3 to estimate the states of all the agents of the MAS.

After finding the subgraphs $\mathbf{G}_i$ and updating the output matrices of the non-defective agents with the coupled measurements, we can create a plant for each group using the techniques described in section 3, chapter 4. Each group estimates the states of all the agents that are part of its subgraph $\mathbf{G}_i$.

Consider a group composed of three agents i, j, and k, two of which have faulty sensors j, k. Their systems are mathematically modeled using the state-space representation. As for the plant model, it is computed according to the following equations.

$$A_{\mathbf{G}_i} = \mathbf{diag}(\begin{bmatrix} A_i & A_j & A_k \end{bmatrix}) = I_{size(\mathbf{G}_i)} \otimes A \tag{5.9}$$

$$B_{\mathbf{G}_i} = \mathbf{diag}(\begin{bmatrix} B & B_j & B_k \end{bmatrix}) = I_{size(\mathbf{G}_i)} \otimes B \tag{5.10}$$

As mentioned earlier, the output matrices of the non-defective agents are computed using Algorithm 3. For agents with defective sensors, they do not perform any relative measurements. However, since some distributed observers require output matrices with no null rows, we took the precaution of eliminating all null rows of the output matrices of the defaulting agents. These rows and columns do not affect the estimation anyway.

Algorithm 4: General estimation algorithm for a homogeneous multi-agent system

Input: i, A, B, C^i, m, max_iteration
Output: $\hat{x}_i$
observability_index_i $\longleftarrow$ (i, (*rank*(**compute_observability_matrix**(A,C^i)) == size(A)))
***G*.send_to_neighbors**(observability_index_i == size(A)) `/* sends its index and a Boolean (True if it is faulty, False otherwise). */`
ND_i, D_i, observability_index_list $\longleftarrow$ ***G*.receive_from_neighbors**()
if *observability_index_i < size(A)* **then**
  $G_i \longleftarrow$ **Find_minimal_sub_graph_D_agents**()
  $C_i \longleftarrow$ **remove_null_lines**(C^i)
else
  $G_i \longleftarrow$ **Find_minimal_sub_graph_ND_agents**()
  $C^i \longleftarrow$ **Find_output_matrix**(A,C^i, observability_index_list, m, D_i, max_iteration)
end
$A_{plant} \longleftarrow \mathrm{I}_{size(\mathbf{G}_i)} \otimes A$
$B_{plant} \longleftarrow \mathrm{I}_{size(\mathbf{G}_i)} \otimes B$
$y_i \longleftarrow$ **sensors_outputs**()
***G*$_i$.send_to_neighbors**(y_i)
$y_{j\in\mathcal{N}_i} \longleftarrow$ ***G*$_i$.receive_from_neighbors**()
$\hat{x}_i$, $\hat{x}_{j\in\mathcal{N}_i} = \longleftarrow$ **state_estimation**(A$_{plant}$, B_{plant}, C_i, ***G*$_i$**, $y_{j\in(i\cup\mathcal{N}_i)}$, initial_conditions)

Algorithm 4's **state_estimation**() function contains the estimator described in section 4.3.1 of the chapter 4 (using any of the observers of chapter 3). Indeed, we will use the chosen observer on a multi-agent system whose network is determined by the agents defined by Algorithms 1 and 2. The output matrices from Algorithm 3 and the y_j outputs measured by the non-faulty sensors are then used to estimate the states.

Depending on the sparsity of the associated graph, and the objectives we wish to achieve, we may choose different observers. It would be interesting to define an expert system that could choose for the estimator to use depending on a given set of objectives and conditions.

5.4 Heterogeneous MAS

Let us now consider the case where the multi-agent system is heterogeneous, that is:

$$\begin{cases} \dot{x}_i = A_i x_i + B_i u \\ y^i = C^i x_i \end{cases} \tag{5.11}$$

To identify the appropriate algorithm for estimating the states of a heterogeneous multi-agent system, we need to look at the results of the agent clustering algorithms defined in Algorithms 1 and 2. From there, we deduce that any defaulting agent must have one and only one non-faulty neighbor for estimating the states and that any group produced by these algorithms must have at least one non-faulty agent. This leads us to only one possible configuration for groups containing a faulty agent: a tree-like configuration containing only two levels with the non-faulty agent as the root of all other defective agents that are part of that group (note that the defaulting agents can have edges between each other).

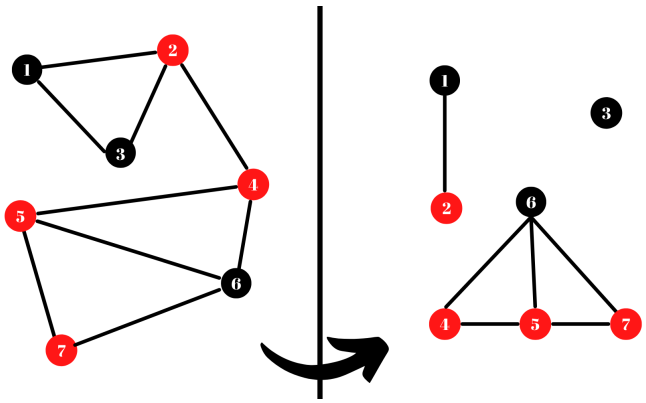

Figure 5.2: Examples of tree-like configurations for the agents groups

The tree configuration allows all agents in a group to access each other's state-space models in only two iterations, allowing the previous algorithm (4) to be used on a heterogeneous system.

Algorithm 5: General estimation algorithm for a heterogeneous multi-agent system

Input: i, A_i, B_i, C^i, m, max_iteration
Output: $\hat{x}_i$
observability_index_i $\longleftarrow$ (i, ($rank$(**compute_observability_matrix**(A,C^i)) == size(A)))
$\boldsymbol{G}$.**send_to_neighbors**(observability_index_i == size(A)) `/* sends its index and a Boolean (True if it is faulty, False otherwise). */`
ND_i, D_i $\longleftarrow$ $\boldsymbol{G}$.**receive_from_neighbors**()
if *observability_index_i* $<$ *size*(A_i) **then**
  G_i $\longleftarrow$ **Find_minimal_sub_graph_D_agents**()
  $\boldsymbol{G}_i$.**send_to_non_defaulting_neighbors**(A_i, B_i)
  A_{G_i}, B_{G_i} $\longleftarrow$G$_i$.**receive_from_neighbors**()
  C_i $\longleftarrow$ **Find_output_matrix**(A,C^i, $\boldsymbol{G}$.nbr_agents, ND_i, max_iteration)
else
  G_i $\longleftarrow$ **Find_minimal_sub_graph_ND_agents**()
  $A_{j\in G_i}$, $B_{j\in G_i}$ $\longleftarrow$G$_i$.**receive_from_neighbors**()
  $\boldsymbol{G}_i$.**send_to_neighbors**($\begin{bmatrix} A_i, & A_{j\mathcal{N}_i\cap G_i} \end{bmatrix}$, $\begin{bmatrix} B_i, & B_{j\mathcal{N}_i\cap G_i} \end{bmatrix}$))
  A_{G_i} $\longleftarrow$ $\begin{bmatrix} A_i, & A_{j\in\mathcal{N}_i\cap G_i} \end{bmatrix}$
  B_{G_i} $\longleftarrow$ $\begin{bmatrix} B_i, & B_{j\in\mathcal{N}_i\cap G_i} \end{bmatrix}$
  C_i $\longleftarrow$ **remove_null_lines**(C^i)
end
A_{plant} $\longleftarrow$**diag**(A$_{G_i}$)
B_{plant} $\longleftarrow$**diag**(B$_{G_i}$)
y_i $\longleftarrow$ **sensors_outputs**()
$\boldsymbol{G}_i$.**send_to_neighbors**(y_i)
$y_{j\in\mathcal{N}_i}$ $\longleftarrow$$\boldsymbol{G}_i$.**receive_from_neighbors**()
$\hat{x}_i$, $\hat{x}_{j\in\mathcal{N}_i}$ = $\longleftarrow$ **state_estimation**(A$_{plant}$, B_{plant}, C_i, $\boldsymbol{G}_i$, $y_{j\in(i\cup\mathcal{N}_i)}$, initial_conditions)

The previous algorithm (5) has been proposed to estimate the states of a heterogeneous multi-agent system. It is based on Algorithm 4 with slight modifications to obtain the state-space model of the whole plant. Indeed, in order to ensure that each agent receives the mathematical models for their group and because the graph has a tree-like configuration, it is sufficient to do the following:

1. Non-defective agents receive the state-space models of defective agents that are part of their group G_i and send this information back to all its defaulting neighbors.

2. Defaulting agents send their state-space model to the non-defaulting neighbor that is part of

their G_i group and wait for it to send back the list of all state-space models of the agents in their G_i group.

3. Using the list of state-space models of all the agents in the group G_i, determine the model of the plant.

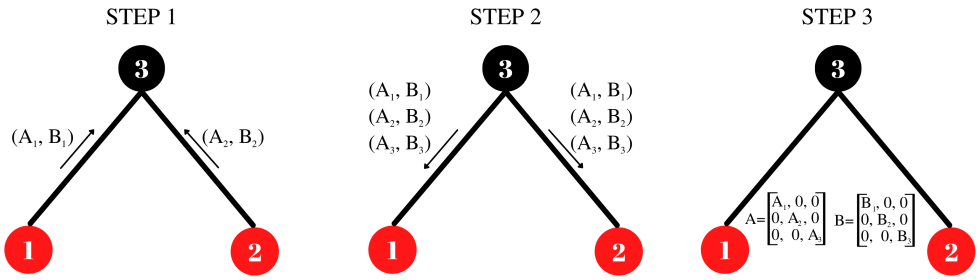

Figure 5.3: The steps that each agent must follow in order to obtain the state space model of the entire multi-agent system

A second way to solve the problem of estimating heterogeneous and defaulting multi-agent systems is to use the observer introduced in Chapter 4, section 4.3.2. Using that estimator makes the solution more straightforward since there is no need to search for the plant model.

Consider a subgraph $\mathbf{G}_i$ defined in figure 5.4 output from the agent clustering algorithms

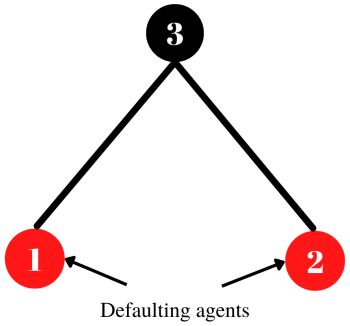

Figure 5.4: The subgraph $\mathbf{G}_i$

The state-space model of the three agents are defined accordingly.

$$\begin{cases} \dot{x}_i = A_i x_i + B_i u \\ y^i = C^i x_i \end{cases} \tag{5.12}$$

With $i \in \{1, 2, 3\}$

The output matrix is updated so that the system defined in equation 5.12 resembles the one defined in equation 4.17 of section 4.3.2, chapter 4. Consider the output matrix of the only non-defaulting agent, agent **3**.

$$y_3 = C_3 x = \begin{bmatrix} C_3^1 & C_3^2 & C_3^3 \\ 0 & 0 & C^3 \\ T^1_{(v_1+1)\to n} & & -T^1_{(v_1+1)\to n} \\ 0 & T^2_{(v_2+1)\to n} & -T^2_{(v_2+1)\to n} \end{bmatrix} x = \begin{bmatrix} C_3^1 & C_3^2 & C_3^3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \sum_{j=1}^{3} C_3^j x_j \tag{5.13}$$

The matrices C_1 and C_2 are simply computed by removing the null rows from the matrices C^1 and C^2 respectively. Finally, we can use the cooperative observer defined in Section 1 of Chapter 4, with agents 1 and 2 having observers of size 2 (this is the combination of agent 3 with one of them) and agent 3 having an observer of size 3. Let's take the example of agent 3's oberver.

$$\dot{\hat{x}}^3 = A^3\hat{x}^3 + L^3(y_3 - \sum_{j\in(3\cup\mathcal{N}_3)} C_j^3\hat{x}_j) + K^3 \begin{bmatrix} \hat{x}_1^1 - \hat{x}_1^3 \\ \hat{x}_2^2 - \hat{x}_2^3 \end{bmatrix} \tag{5.14}$$

Algorithm 6: General estimation algorithm for a heterogeneous multi-agent system using Cooperative Estimator of MAS With Coupled Measurements (see section 4.3.2)

Input: i, A, B, C^i, m, max_iteration
Output: $\hat{x}_i$
observability_index_i ⟵ (i, (*rank*(**compute_observability_matrix**(A,C^i)) == size(A)))
***G*.send_to_neighbors**(observability_index_i == size(A)) `/* sends its index and a Boolean (True if it is faulty, False otherwise). */`
ND_i, D_i ⟵ ***G*.receive_from_neighbors**()
if *observability_index_i < size(A)* **then**
  G_i ⟵ **Find_minimal_sub_graph_D_agents**()
  C_i ⟵ **Find_output_matrix**(A,C^i, ***G***.nbr_agents, ND_i, max_iteration)
else
  G_i ⟵ **Find_minimal_sub_graph_ND_agents**()
  C_i ⟵ **remove_null_lines**(C^i)
end
C_i ⟵ **Find_output_matrix**(A,C^i, ***G***.nbr_agents, ND_i, max_iteration)
***G*$_i$.send_to_neighbors**(A_i, B_i)
$A_{j\in G_i\cap\mathcal{N}_i}$, $B_{j\in G_i\cap\mathcal{N}_i}$= ⟵ ***G*$_i$.receive_from_neighbors**()
y_i ⟵ **sensors_outputs**()
***G*$_i$.send_to_neighbors**(y_i)
$y_{j\in\mathcal{N}_i}$ ⟵ ***G*$_i$.receive_from_neighbors**()
$\hat{x}_i$, $\hat{x}_{j\in\mathcal{N}_i}$ ⟵ **Cooperative_State_Estimation_Coupled_Measurements**(A$_{j\in G_i\cap\mathcal{N}_i}$, $B_{j\in G_i\cap\mathcal{N}_i}$, C_i, ***G*$_i$**, $y_{j\in(i\cup\mathcal{N}_i)}$, initial_conditions)

5.5 Conclusion

Using the distributed observers we saw in the previous chapters, we provided a collection of algorithms for estimating the states of agents with malfunctioning sensors that are part of a multi-agent system. By introducing relative measurements, we have established an algorithm for clustering agents into minimum groups while ensuring their joint observability. In summary, we have investigated the state estimation of homogeneous and heterogeneous multi-agent systems while correcting sensor-related defects.

Although the solution developed within the framework of this project is satisfactory, there are undoubtedly improvements to be added, particularly concerning the following points:

- Take into account the case where the agents are not observable on their own but the system is jointly observable.
- Allow groups containing only defaulting agents, if their failure is not due to the same sensor.

.

CHAPTER 6

SOLUTION IMPLEMENTATION AND SIMULATION

6.1 Introduction

Because of their simplicity and adaptability, unmanned aerial vehicles (UAVs) have sparked a lot of attention during the last decade. To the point where researchers and industries are now concentrating on UAV swarms. They are frequently used in critical situations such as rescue missions, military operations, or even outdoor shows in which they are surrounded by a huge number of civilians [32]. Therefore, they must be reliable as a group. Having a multi-agent system, on the other hand, raises the chances of one agent failing. As a result, we feel that multi-agent UAV systems are a suitable application of the sensor failure correction technique provided in the previous chapter.

In this chapter, we will use a quadrotor swarm to test the algorithm we provided in the previous chapter and discuss the results.

6.2 The quadrotor model

A quadrotor is an unnamed aerial vehicle that has two rotors rotating clockwise and two counter-clockwise [33]. They have several advantages, including their tiny size and stable hover. Recently, the research community has shown increasing interest in them, especially when it comes to quadrotor swarms.

The propellers are designed in such a way to control the quadrotor at six degrees of freedom. They are placed in a crossed configuration. In this configuration, the opposing rotors rotate in the same direction, which is necessary to allow independent control of yaw, pitch, and roll. These torques (U_{roll}, U_{pitch}, U_{yaw}) and the thrust force (U_{thrust}) are generated using the rotation speeds of the four propellers (see figure 6.1) and are determined accordingly:

$$\begin{cases} U_{thrust} = f_l(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ U_{roll} = f_l d(\Omega_4^2 - \Omega_2^2) \\ U_{pitch} = f_l d(\Omega_3^2 - \Omega_1^2) \\ U_{yaw} = f_d(\Omega_2^2 - \Omega_1^2 + \Omega_4^2 - \Omega_3^2) \end{cases} \tag{6.1}$$

With Ω_i the rotation speed of motor i, f_l and f_d are the lift and drag factor respectively, d is the

distance from the center of mass to the center of each rotor, and U_{thrust}, U_{roll}, U_{pitch}, U_{yaw} are the four control inputs.

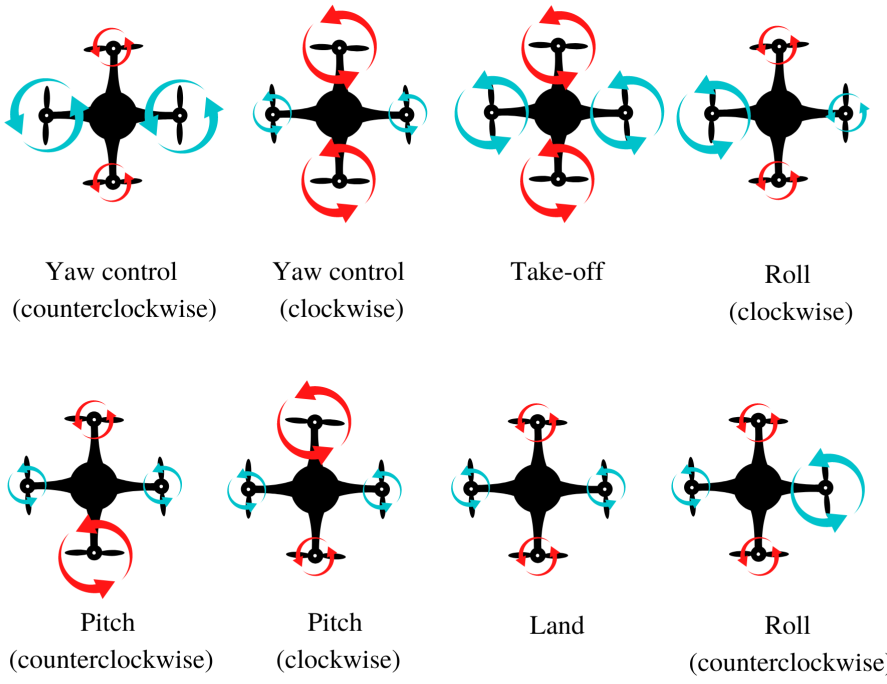

Figure 6.1: Motors configuration needed to obtain the different movements

Using Newton's second law of motion, we can easily define the nonlinear model of the quadrotor [34], consider a quadrotor defined in the 3 dimentional space using the Euclidean position (x, y, z) and the Euler angles (ϕ, θ, ψ).

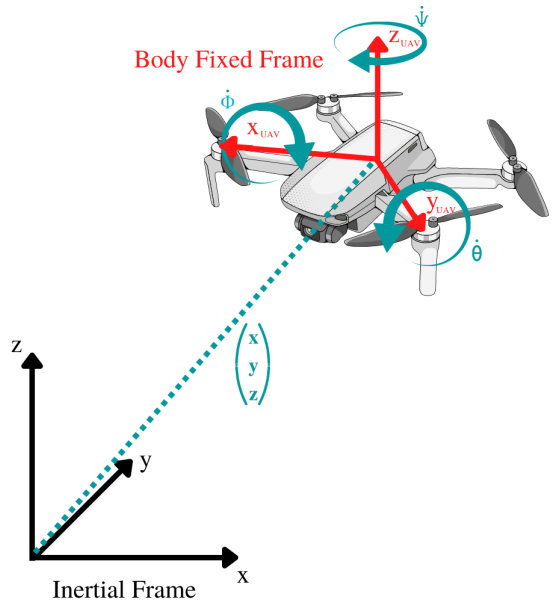

Figure 6.2: Quadrotor model

We define

m	Total mass of the quadrotor.
g	Force of gravity.
$X \in \mathbb{R}^3$	Position of the quadrotor (x, y, z).
$V \in \mathbb{R}^3$	Velocity of the quadrotor $(\dot{x}, \dot{y}, \dot{z})$.
$\Omega \in \mathbb{R}^3$	Angular velocity of the quadrotor (p, q, r)
R	Rotational matrix of the quadrotor from the inertial frame.
$J \in \mathbb{R}^{3\times3}$	Inertia matrix $\mathbf{diag}(\begin{bmatrix} J_{xx} & J_{yy} & J_{zz} \end{bmatrix})$.

$$\dot{X} = V \tag{6.2}$$

$$m\dot{V} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ U_{thrust} \end{bmatrix} \tag{6.3}$$

$$\dot{R} = R\hat{\Omega} \tag{6.4}$$

$$\begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} \dot{\Omega} + \Omega \times \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} \Omega = \begin{bmatrix} U_{roll} \\ U_{pitch} \\ U_{yaw} \end{bmatrix} \tag{6.5}$$

Where $\hat{\cdot}$ is the hat map: $\mathbb{R}^3 \to so(3)$. Suppose that $\Omega^T = \begin{bmatrix} p & q & r \end{bmatrix}$ than:

$$\hat{\Omega} = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}$$

The rotational matrix R is the rotational matrix of the quadrotor from the body-fixed frame to its inertial frame [35].

$$R = \begin{bmatrix} \cos\phi\cos\theta & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \sin\phi\cos\theta & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi \\ -\sin\theta & \cos\theta\sin\psi & \cos\theta\cos\psi \end{bmatrix} \tag{6.6}$$

Finally, we can define the relation between the angular velocities in the inertial frame (p, q, r) and those of the body-fixed frame $(\dot{\phi}, \dot{\theta}, \dot{\psi})$.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{6.7}$$

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = R_{\mathbf{x}\to\mathbf{x}_{UAV}} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{6.8}$$

The inverse of matrix $R_{\mathbf{x}\to\mathbf{x}_{UAV}}$ is used to define the dynamics of the angular velocities.

Finally, the nonlinear state-space model of the quadrotor can be deduced using equations 6.2 to 6.5 and rotation matrix $R_{\mathbf{x}\to\mathbf{x}_{UAV}}$. Matrix $R_{\mathbf{x}\to\mathbf{x}_{UAV}}$ is introduced in equation 6.5 to obtain the dynamic of the rotational angles in the inertial frame. To simplify the nonlinear model, we consider the Euler angles to be close to zero. Therefore, we can consider $p \approx \dot{\phi}$, $q \approx \dot{\theta}$, and $r \approx \dot{\psi}$.

$$\begin{cases} \ddot{\phi} = \dfrac{J_{yy} - J_{zz}}{J_{xx}} \dot{\theta}\dot{\psi} + \dfrac{U_{roll}}{J_{xx}} \\ \ddot{\theta} = \dfrac{J_{zz} - J_{xx}}{J_{yy}} \dot{\phi}\dot{\psi} + \dfrac{U_{pitch}}{J_{yy}} \\ \ddot{\psi} = \dfrac{J_{xx} - J_{yy}}{J_{zz}} \dot{\phi}\dot{\theta} + \dfrac{U_{yaw}}{J_{zz}} \end{cases} \tag{6.9}$$

Now, we can linearize the model by considering that the quadrotor is around an operating point. We will use the approximations and the linear model designed by [36]. They considered the following assumptions.

$$\begin{cases} p \approx q \approx r \approx 0 \\ sin(\phi) \approx 0 \\ sin(\theta) \approx \theta \\ sin(\psi) \approx \psi \end{cases} \tag{6.10}$$

Using these approximations, we can now define the linear model of the quadrotor.

$$\begin{cases} \dot{\mathbf{x}} = A\mathbf{x} + B \begin{bmatrix} U_{thrust} \\ U_{roll} \\ U_{pitch} \\ U_{yaw} \end{bmatrix} \\ \mathbf{y} = C\mathbf{x} \end{cases} \tag{6.11}$$

Such that: $\mathbf{x} = \begin{bmatrix} x^T & y^T & z^T & \phi^T & \theta^T & \psi^T & \dot{x}^T & \dot{y}^T & \dot{z}^T & \dot{\phi}^T & \dot{\theta}^T & \dot{\psi}^T \end{bmatrix}^T$

With the state-space matrices are defined as follows.

$$A = \begin{bmatrix} 0_{3\times4} & 0_{3\times1} & 0_{3\times1} & I_{3\times3} & 0_{3\times3} \\ 0_{3\times4} & 0_{3\times1} & 0_{3\times1} & 0_{3\times3} & I_{3\times3} \\ 0_{1\times4} & 0 & g & 0_{1\times3} & 0_{1\times3} \\ 0_{1\times4} & -g & 0 & 0_{1\times3} & 0_{1\times3} \\ 0_{4\times4} & 0_{4\times1} & 0_{4\times1} & 0_{4\times3} & 0_{4\times3} \end{bmatrix} \tag{6.12}$$

$$B = \begin{bmatrix} 0_{8\times1} & 0_{8\times1} & 0_{8\times1} & 0_{8\times1} \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{J_{xx}} & 0 & 0 \\ 0 & 0 & \frac{1}{J_{yy}} & 0 \\ 0 & 0 & 0 & \frac{1}{J_{zz}} \end{bmatrix} \tag{6.13}$$

This drone usually contains several sensors to allow it to be fully observable and measure its position and rotation in 3D space, that is $\mathbf{y} = \begin{bmatrix} x^T & y^T & z^T & \phi^T & \theta^T & \psi^T & \dot{x}^T & \dot{y}^T & \dot{z}^T \end{bmatrix}^T$, therefore, the output matrix is written as follows.

$$C = \begin{bmatrix} I_{9\times9} & 0_{9\times3} \end{bmatrix} \tag{6.14}$$

This linearized model ensures that the system remains controllable and observable.

However, the quadrotor is not a naturally stable system, it will need to be stabilized.

For the purposes of our example, we have picked the following values for our drone's constants:

$$\begin{aligned} J = \mathbf{diag}(\begin{bmatrix} 0.0820 & 0.0845 & 0.1377 \end{bmatrix}) \\ m = 4.34Kg \\ d = 0.315m \\ f_l = 2 \times 10^{-4} \\ f_d = 7 \times 10^{-5} \end{aligned}$$

We borrowed these values from a UAV designed by [37].

6.3 Multi-agent UAV system

We will implement the approach proposed in the previous chapter on a multi-agent quadrotor system that we will present in the current section.

Consider a set of three quadrotors that, according to a graph $\boldsymbol{G}$, can communicate with their neighbors. The network is static, that is it won't change over time. Furthermore, the quadrotors are modeled using the linearized state-space model defined in the previous section (see equation 6.12 and 6.13). All the UAVs have the same mathematical model. Therefore, the MAS is homogeneous.

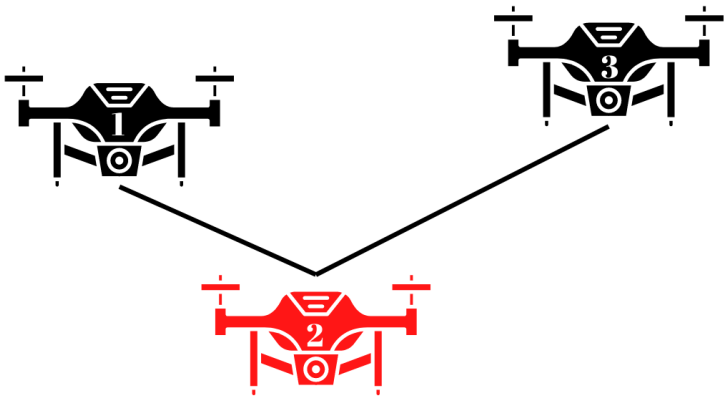

Figure 6.3: The graph associated with the set of quadrotors

Agent 2 is the one that has defective sensors, and it is connected to both the other agents.

The groups depicted in figure 6.4 were created using the clustering algorithms 2 and 1 described in the previous chapter. These algorithms generated two subgraphs $\mathbf{G}_1$ composed of agents **1** and **2**, and $\mathbf{G}_2$ composed of only agent **3**.

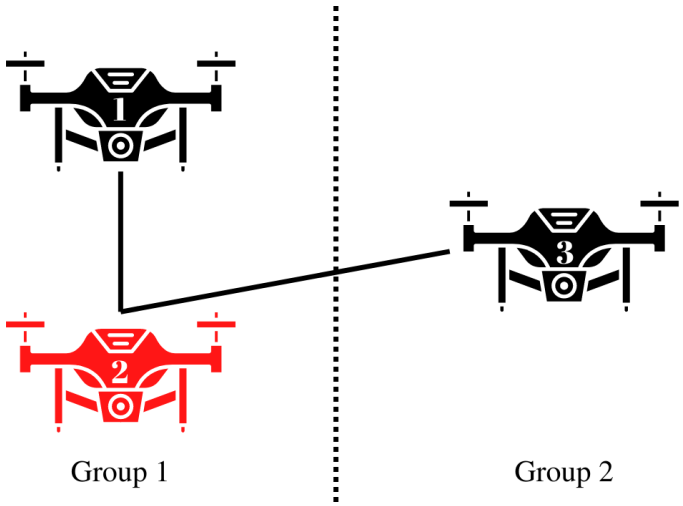

Figure 6.4: The groups formed after applying algorithm 1 and 2

Quadrotor **2** has randomly selected one of its non-failing neighbors. The remaining two agents wait for Quadrotor **2** to send them information about their cluster and reorganize a new one based on what they received (see Algorithm 2). Finally, Quadrotor **1** receives a signal from the second agent, and Quadrotor **3** receives no feedback and ends up in a cluster by itself.

6.4 Defaulting agent

The defaulting system has lost the functionality of several sensors; it cannot measure its position on the $x-$ and $y-$ axes and its rotations θ and ψ. Its output matrix is defined in the following equation, and the remaining outputs are $\mathbf{y}_2 = \begin{bmatrix} z_2^T & \phi_2^T & \dot{x}_2^T & \dot{y}_2^T & \dot{z}_2^T \end{bmatrix}$.

$$\mathbf{y}_2 = C_2 \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \tag{6.15}$$

$$C_2 = \begin{bmatrix} 0_{2\times14} & I_{2\times2} & 0_{2\times2} & 0_{2\times3} & 0_{2\times3} \\ 0_{3\times14} & 0_{3\times2} & 0_{3\times2} & I_{3\times3} & 0_{3\times3} \end{bmatrix} \tag{6.16}$$

The system (C_2, A) is unobservable, it has an observability index 10, which means that the last two lines of the staircase transformation matrix T_2 of that system are used to define the states that need to be relatively measured to obtain a jointly observable multi-agent quadrotor system composed of quadrotor **1** and **2**.

The transformation matrix T_2 is defined as follows.

$$T_2 = \left[\begin{array}{cccccc} 0_{2\times2} & -I_{2\times2} & 0_{2\times2} & 0_{2\times3} & 0_{2\times2} & 0_{2\times1} \\ 0_{3\times2} & 0_{3\times2} & 0_{3\times2} & -I_{3\times3} & 0_{3\times2} & 0_{3\times1} \\ 0_{2\times2} & 0_{2\times2} & J_{2\times2} & 0_{2\times3} & 0_{2\times2} & 0_{2\times1} \\ 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times3} & I_{2\times2} & 0_{2\times1} \\ 0_{1\times2} & 0_{1\times2} & 0_{1\times2} & 0_{1\times3} & 0_{1\times2} & -1 \\ \hline J_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times3} & 0_{2\times2} & 0_{2\times1} \end{array}\right] \tag{6.17}$$

Using T_2, we can finally deduce which states of the agent **2** should be measured relative to the agent **1**. We find: $\mathbf{y}_1 = \begin{bmatrix} x_1^T & y_1^T & z_1^T & \phi_1^T & \theta_1^T & \psi_1^T & \dot{x}_1^T & \dot{y}_1^T & \dot{z}_1^T & y_2^T - y_1^T & x_2^T - x_1^T \end{bmatrix}$.

$$\mathbf{y}_1 = C_1 \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \tag{6.18}$$

$$C_1 = \begin{bmatrix} 1 & 0 & 0_{1\times7} & 0 & 0 & 0_{1\times7} \\ 0 & 1 & 0_{1\times7} & 0 & 0 & 0_{1\times7} \\ 0_{7\times1} & 0_{7\times1} & I_{7\times7} & 0_{7\times1} & 0_{7\times1} & 0_{7\times7} \\ 0 & -1 & 0_{1\times7} & 0 & 1 & 0_{1\times7} \\ -1 & 0 & 0_{1\times7} & 1 & 0 & 0_{1\times7} \end{bmatrix} \tag{6.19}$$

Measuring the position of a neighbor on the x- and y- axes is simple in practice. In fact, it is likely that a quadrotor has a camera that can be used to quickly locate neighboring drones.

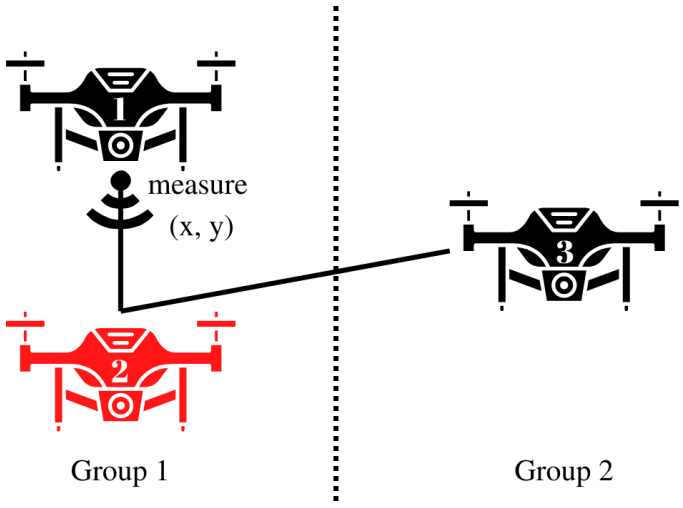

Figure 6.5: The relative measurements made by agent **1**

Agent 3 has been clustered alone, so its output matrix does not change, it can measure its position (x, y, z), its angular rotation (ϕ, θ, ψ), and its speed $(\dot{x}, \dot{y}, \dot{z})$.

$$C_3 = \begin{bmatrix} I_{9\times9} & 0_{9\times3} \end{bmatrix} \tag{6.20}$$

6.5 Distributed observer

Now it's only a matter of deciding which distributed estimator to utilize in these conditions.

We chose to use the Distributed Luenberger Observer for its simplicity, speed, and robustness, as we have seen in chapter 3.

Since the subgrapsh $\mathbf{G}_1$ and $\mathbf{G}_2$ are not sparse, the Cooperative Estimator Using Coupled Measurements, which was presented in section 4.3.2, Chapter 4, does not reduce the size of the observer. Hence, favoring simplicity and performance (see table 3.4), we have chosen to implement the distributed Luenberger observer. Finally, for the group containing a single agent, the observer is no longer distributed. Therefore, the DLO becomes a simple Luenberger estimator.

In addition, since the quadrotor is an unstable plant, we implemented two simulations, one involving the free system and the other containing an **observer-based** feedback controller. Furthermore, to ensure that the estimated state values used in the controller, approximate the actual state values before the controller takes effect, we made the dynamics of the observer faster than that of the controller.

It should be noted that the implementation of the feedback control is theoretical; in reality, this type of control is not adequate for a UAV (see [38], [39], [40], and [41] for more detail on how to control a UAV).

We define

(A, B)	The state-space model of the quadrotor.
$\hat{x}_i^j$	The estimate of agent i's states made by agent j.
K_i	The feedback controller gain, null matrix if the controller is not implemented.
L_i, M_i, k_i, and γ	The parameters of the DLO (see equation 3.5 and section 3.3.1).
C_i	The output matrices (see equation 6.19, 6.16, 6.20).

$$\begin{bmatrix}\dot{\hat{x}}_1^1\\ \dot{\hat{x}}_2^1\end{bmatrix} = (I_2 \otimes A)\begin{bmatrix}\hat{x}_1^1\\ \hat{x}_2^1\end{bmatrix} - (I_2 \otimes B)K_1\begin{bmatrix}\hat{x}_1^1\\ \hat{x}_2^1\end{bmatrix} + L_1(y_1 - C_1\begin{bmatrix}\hat{x}_1^1\\ \hat{x}_2^1\end{bmatrix}) + \gamma M_1^{-1}(k_1)\begin{bmatrix}\hat{x}_1^2 - \hat{x}_1^1\\ \hat{x}_2^2 - \hat{x}_2^1\end{bmatrix} \tag{6.21}$$

$$\begin{bmatrix}\dot{\hat{x}}_1^2\\ \dot{\hat{x}}_2^2\end{bmatrix} = (I_2 \otimes A)\begin{bmatrix}\hat{x}_1^2\\ \hat{x}_2^2\end{bmatrix} - (I_2 \otimes B)K_2\begin{bmatrix}\hat{x}_1^2\\ \hat{x}_2^2\end{bmatrix} + L_2(y_2 - C_2\begin{bmatrix}\hat{x}_1^2\\ \hat{x}_2^2\end{bmatrix}) + \gamma M_2^{-1}(k_2)\begin{bmatrix}\hat{x}_1^1 - \hat{x}_1^2\\ \hat{x}_2^1 - \hat{x}_2^2\end{bmatrix} \tag{6.22}$$

With L_i, and M_i, $i \in \{1, 2\}$ defined according to the laws introduced in chapter 3 (see equations 3.6, 3.7, and 3.5). Indeed, using the **place**() function from Scipy Python, we compute the value of the gain L_i.

$$\dot{\hat{x}}_3 = A\hat{x}_3 - BK_3\hat{x}_3 + L_3(y_3 - C_3\hat{x}_3) \tag{6.23}$$

When implementing the observer-based controller, we chose the gains K_i for $i \in \{1, 2, 3\}$ using the **place**() function from Scipy Python. The eigenvalues were taken such that the dynamic of the controller would be slower than that of the observer.

Note: In our case, we chose to work with the Luenberger distributed observer, but depending on the size of the groups, the sparsity of the associated graphs, and the objectives we wish to achieve, we may make a different choice. It would be interesting to define an expert system that could choose for the estimator to use depending on some objectives and conditions.

6.6 Simulations

We will present the outcomes of our simulations in this section.

6.6.1 Initialization

This table summarizes the initial values chosen for our drones and their estimates.

Table 6.1: Initial values of $\mathbf{x}$(t) and (t)

i	x (m)	y (m)	z (m)	ϕ (°)	θ (°)	ψ (°)	$\hat{x}$ (m)	$\hat{y}$ (m)	$\hat{z}$ (m)	$\hat{\phi}$ (°)	$\hat{\theta}$ (°)	$\hat{\psi}$ (°)
1	0	0	0	0	0	0	-2	-3	-2.5	0	0	0
2	1.5	-3	-3	0	0	0	2.8	0	-1.5	0	0	0
3	0	0	0	0	0	0	-1	3	0.5	0	0	0

These values have been chosen so that the response x(t) is zero. Indeed, since the elements of the matrix A are null when multiplied by the position (x, y, z), we can afford to modify the initial values of the first three states, contrary to the others.

The parameters of the distributed observer were chosen according to the laws presented in section 3.3.1 of chapter 3. We took $\gamma = 0.1$ and $k_i = 0.1$. While these parameters were chosen by calibration, the matrix gains L_i and M_i, for $i \in \{1, 2\}$ were taken according to laws 3.6 and 3.7. We selected the eigenvalues of the associated matrix (defined in equation 3.6) to randomly vary from -1 to -0.5, such that

$$\lambda_{L_1} = [-0.559 \quad -0.819 \quad -0.571 \quad -0.972 \quad -0.760 \quad -0.707 \quad -0.632 \quad -0.887 \quad -0.728 \quad -0.784 \quad].$$

$$\lambda_{L_2} = [-0.509 \quad -0.808 \quad -0.806 \quad -0.808 \quad -0.971 \quad -0.840 \quad -0.679 \quad -0.718 \quad -0.848 \quad -0.5301 \quad -0.833 \quad -0.835 \quad -0.605 \quad -0.564 \quad -0.657 \quad -0.681 \quad -0.785 \quad -0.719 \quad -0.994 \quad -0.551].$$

On the other hand, M_i is computed using the gain L_i through equation 3.7.

For the third quadrotor, these parameters are not needed, and L_3 is computed by pole placement with the eigenvalues randomly varying from -1 to -0.5.

$$\lambda_{L_3} = [-0.784 \quad -0.962 \quad -0.535 \quad -0.543 \quad -0.510 \quad -0.916 \quad -0.889 \quad -0.935 \quad -0.989 \quad -0.899 \quad -0.730 \quad -0.890 \quad].$$

6.6.2 Distributed observation

The results are illustrated in the figures below. The estimated trajectories $\hat{x}_i^j$, $j, i \in \{1, 2\}$ are in colored solid curves while $x_i(t)$, $i \in \{1, 2\}$ are in blue dashed curves.

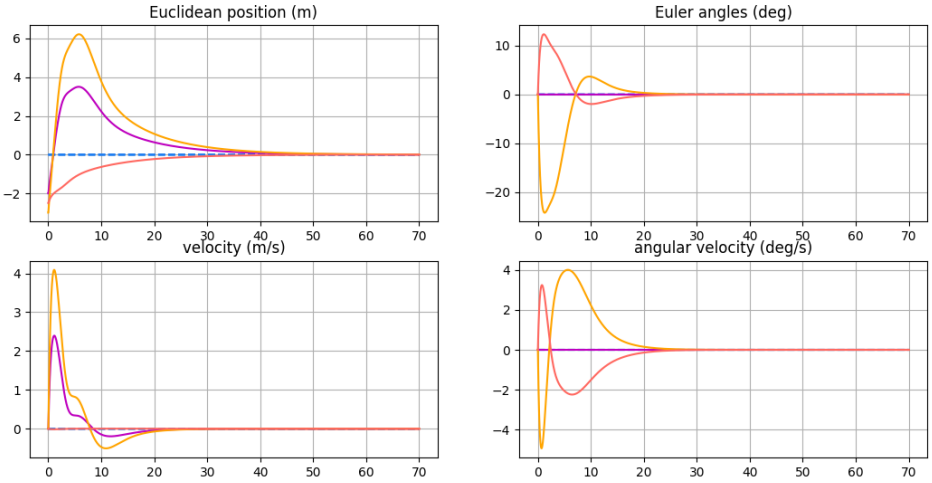

Figure 6.6: States $x_1(t)$ and their average estimates $\mathbf{avg}(\hat{x}_1^1(t), \hat{x}_1^2(t))$ of quadrotor **1**

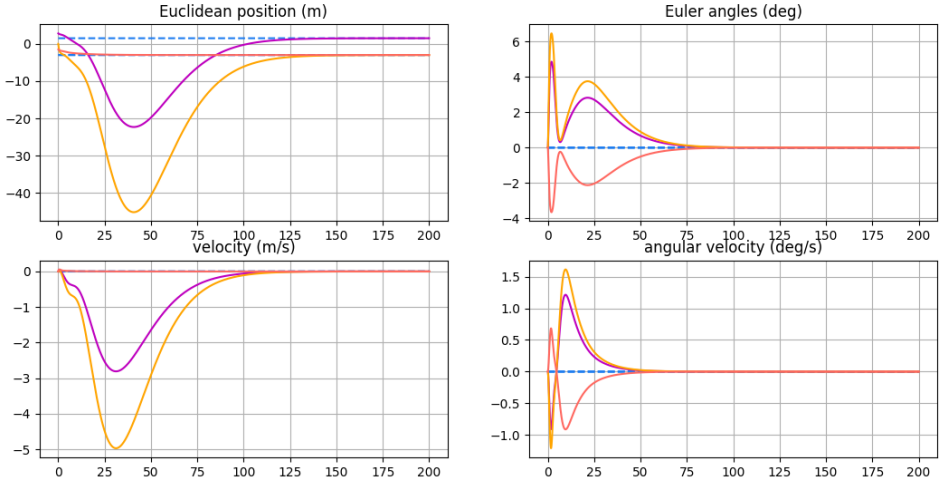

Figure 6.7: States $x_2(t)$ and their average estimates $\mathbf{avg}(\hat{x}_2^1(t), \hat{x}_2^2(t))$ of quadrotor **1**

The observer converges, and we can see that the color curves overlap on the blue dotted lines. The settling time of the state estimation of the first quadrotor is 63s, while that of the second is 176s (settling time is the time after which the observation error becomes lower than 10^{-2}).

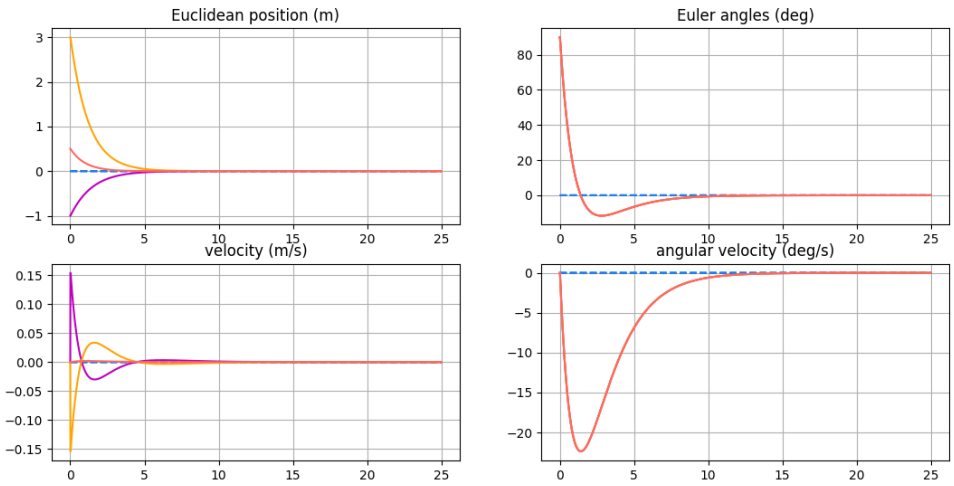

Figure 6.8: States $x_3(t)$ and their estimates $\hat{x}_3(t)$ of quadrotor **3**

6.6.3 Observer-based controller

Let us consider the same system and observer, and let us implement an observer-based controller to it. The goal is for the system to reach a desired value at infinity, which is not the case when the system is allowed to evolve freely.

We designed the controller by choosing the eigenvalues of $I_2 \otimes A - (I_2 \otimes B)K_i$, for $i \in \{1, 2, 3\}$ to randomly vary between -0.1 to -0.2

We took for the first two agents ($i \in \{1, 2\}$):

$$\lambda_{K_i} = [-0.155 \quad -0.171 \quad -0.160 \quad -0.154 \quad -0.142 \quad -0.165 \quad -0.143 \quad -0.189...$$
$$-0.196 \quad -0.138 \quad -0.179 \quad -0.152].$$

for the third agent ($i = 3$):

$$\lambda_{K_3} = [-0.2097 \quad -0.2430 \quad -0.2206 \quad -0.2089 \quad -0.1847 \quad -0.2292 \quad -0.1875 \quad -0.2784...$$
$$-0.2927 \quad -0.1767 \quad -0.2583 \quad -0.2058].$$

For the group $\mathbf{1} - \mathbf{2}$, the results are illustrated in the figures below. The estimated trajectories $\hat{x}_i^j$, $j, i \in \{1, 2\}$ are in colored solid curves while $x_i(t)$, $i \in \{1, 2\}$ are in blue dashed curves.

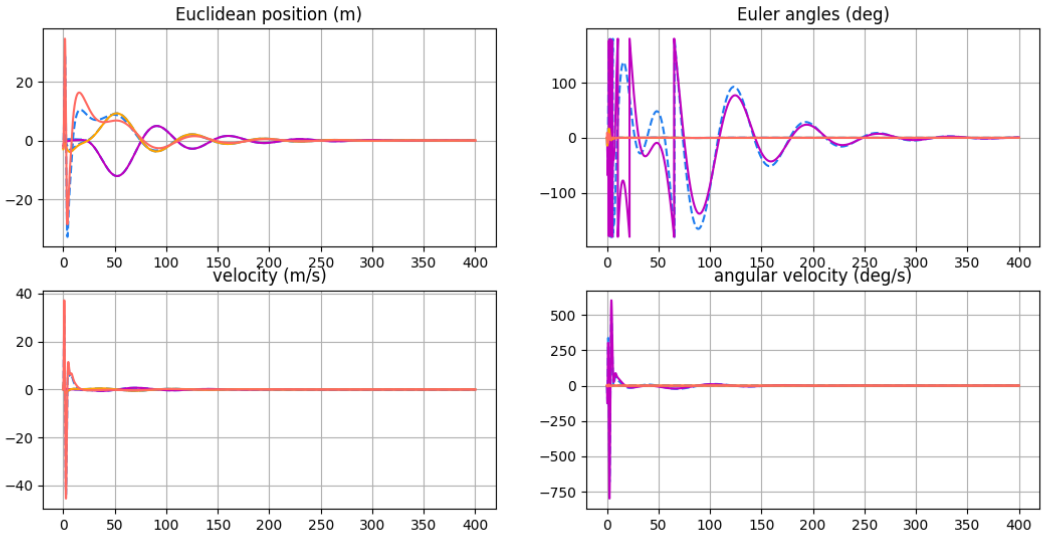

Figure 6.9: States $x_1(t)$ and their average estimates $\mathbf{avg}(\hat{x}_1^1(t), \hat{x}_1^2(t))$ of quadrotor **1**

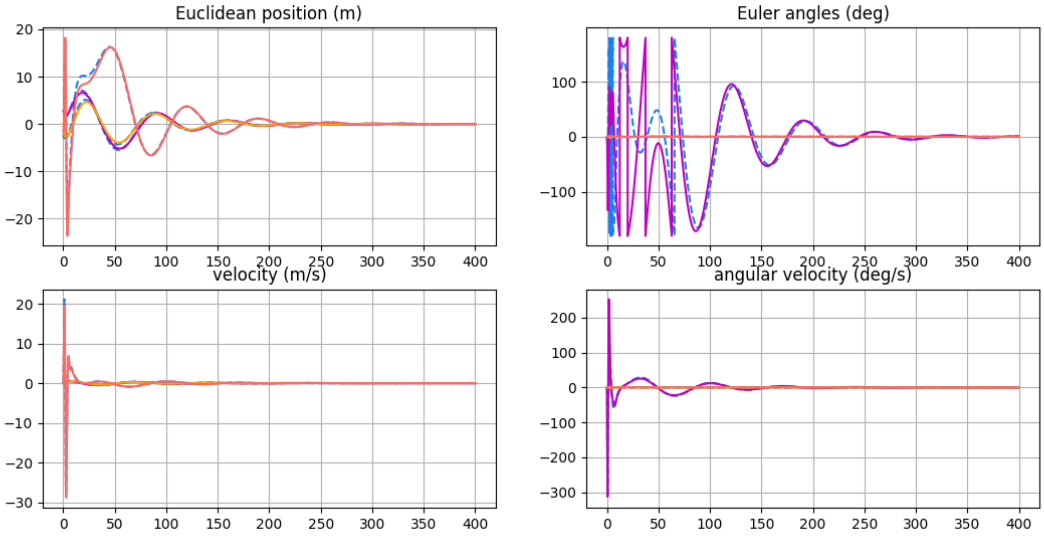

Figure 6.10: States $x_2(t)$ and their average estimates $\mathbf{avg}(\hat{x}_2^1(t), \hat{x}_2^2(t))$ of quadrotor **2**

Estimates and actual values overlap quickly for all variables, but rotation angles take more time. We also see at the beginning the quadrotor rotating around itself, this is unrealistic. This is due to the implementation of a control that is not appropriate for this type of systems (the problem of the control of quadrotors is widely studied in the literature, see [38], [39], [40], and [41]).

For the system composed of agent 3, the estimation is straightforward, it is performed using a simple Luenberger observer. The results are illustrated in figure 6.11.

We took the initial values of the Euler angles to be equal to $(0, 0, 0)$ and kept the same initial values as the one presented in table 6.1 for the remaining states. The estimated trajectories $\hat{x}_3$, $j, i \in \{1, 2\}$ are in colored solid curves while $x_3(t)$, $i \in \{1, 2\}$ are in blue dashed curves.

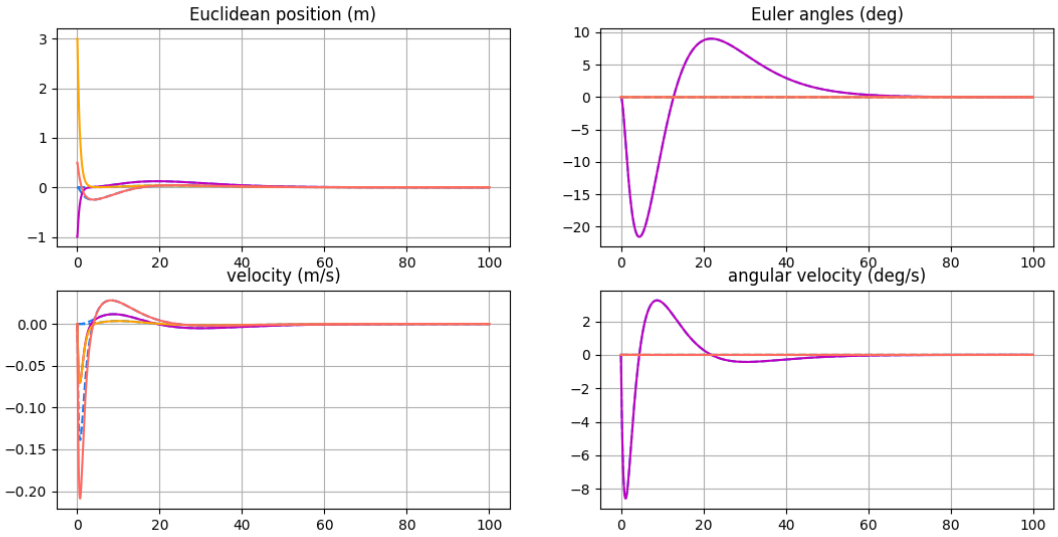

Figure 6.11: States $x_3(t)$ and their estimates $\hat{x}_3(t)$ of quadrotor **3**

6.6.3.1 Noisy outputs

In reality, relative measurements are often noisier than values measured by an onboard sensor. For this reason, we decided to test the observer with measurement noise for the last two outputs of the quadrotor **1** system ($x_2 - x_1$ and $y_2 - y_1$). We tested the observer with three levels of noise: $(\pm 0.1m, \pm 5°, \pm 0.1m/s, \pm 5°)$, $(\pm 0.2m, \pm 10°, \pm 0.2m/s, \pm 10°)$, and $(\pm 0.5m, \pm 25°, \pm 1m/s, \pm 25°)$ and illustrated the observation errors in figures 6.15 to 6.17. It converges in the first two cases but with oscillations around the final value. This can be corrected by using an averaging filter. As for the third level, the output is too noisy, it might be too difficult to recover the estimates.

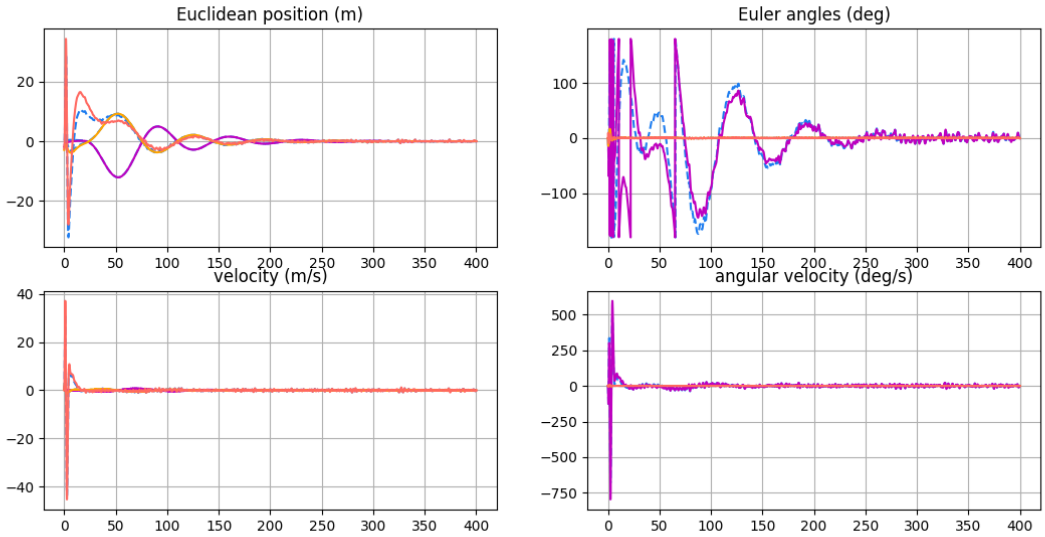

Figure 6.12: Observation errors of the quadrotor **1** in the first case of noisy relative measurements

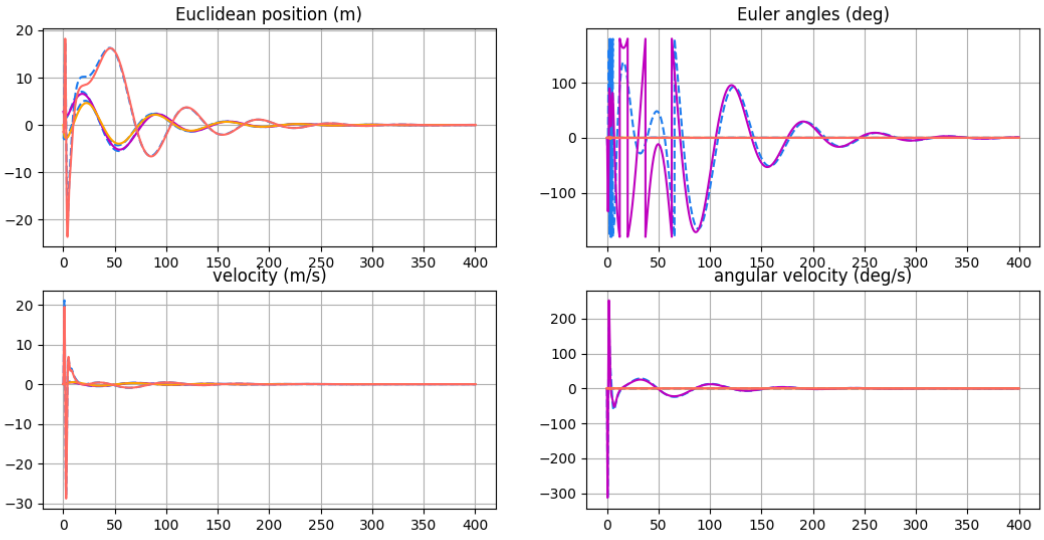

Figure 6.13: Observation errors of the quadrotor **2** in the first case of noisy relative measurements

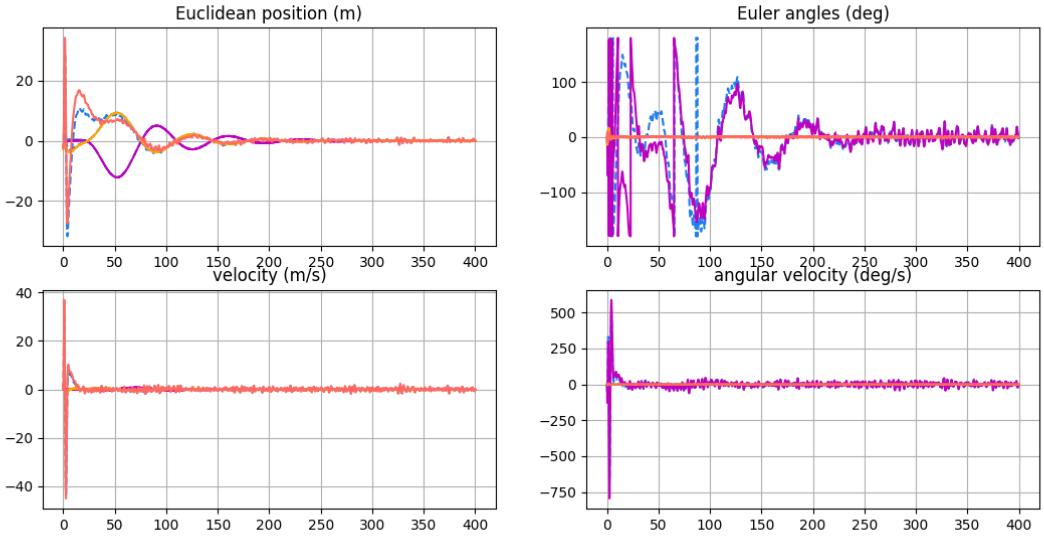

Figure 6.14: Observation errors of the quadrotor **1** in the second case of noisy relative measurements

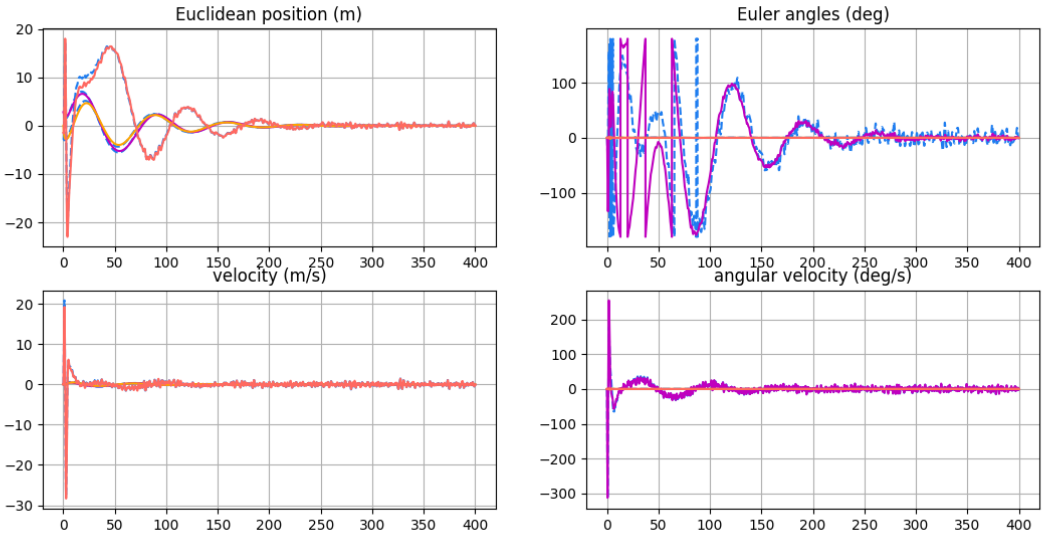

Figure 6.15: Observation errors of the quadrotor **2** in the second case of noisy relative measurements

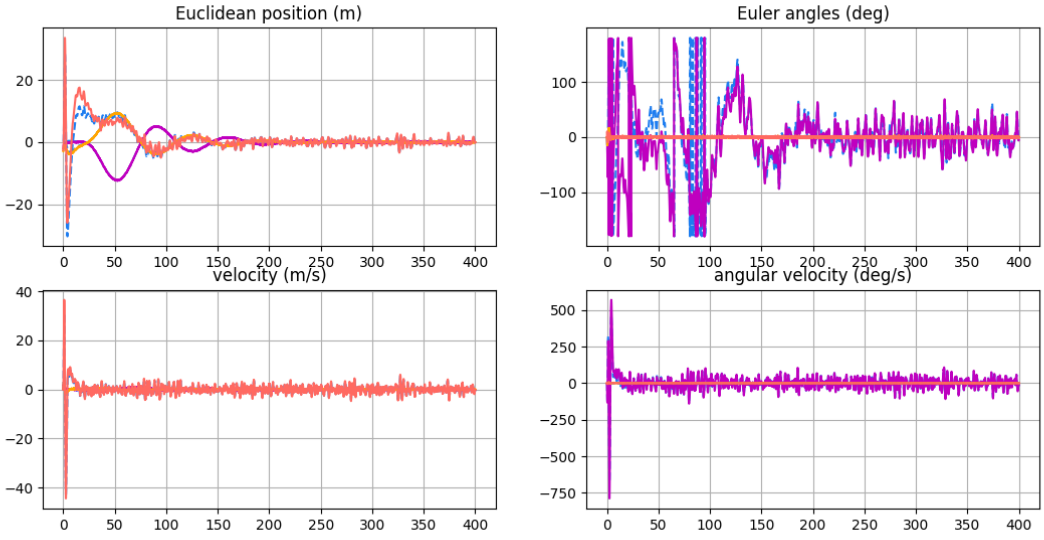

Figure 6.16: Observation errors of quadrotor **1** in the last case of noisy relative measurements

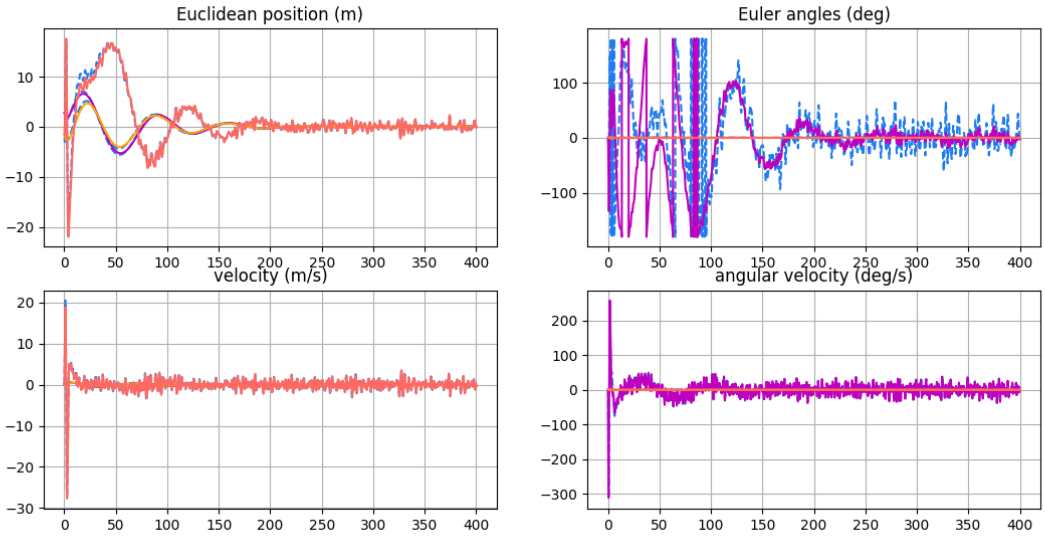

Figure 6.17: Observation errors of quadrotor **2** in the last case of noisy relative measurements

6.6.3.2 Noisy parameters

Since the system is linearized, the mathematical model on which we tested the observer is not the actual model. Therefore, to simulate the effect the observer would have on the nonlinear system, we implemented parametric noise to the state-space model of the quadrotor. This is called a robustness test, and we chose three levels of noise: 5%, 10%, and 20% of the parameters.

To obtain

We have illustrated the results in figures 6.18 to 6.23. In all three cases, the results converge, although the system is unstable, and we have carried out a controller based on the observer. However, adding noise slows down the convergence rate of the observer and the controller.

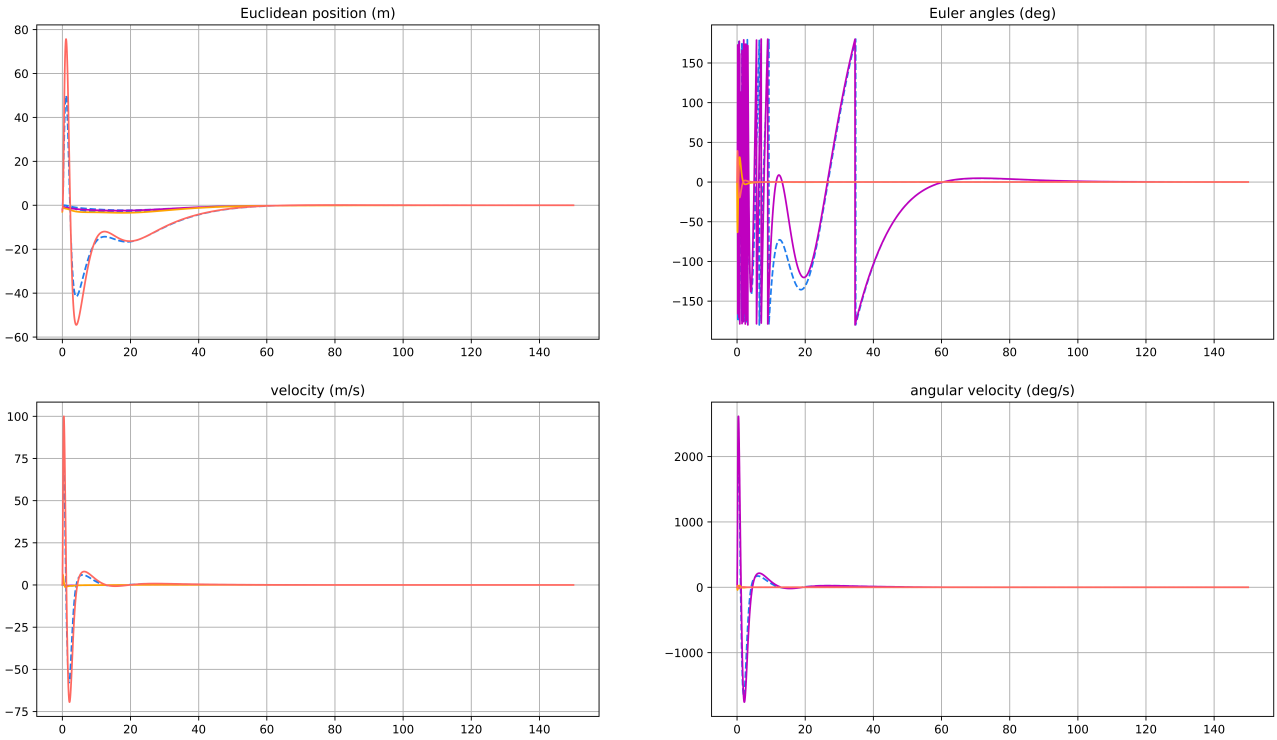

Figure 6.18: Observation errors of quadrotor **1** in the case of noisy parameters (5%) noise

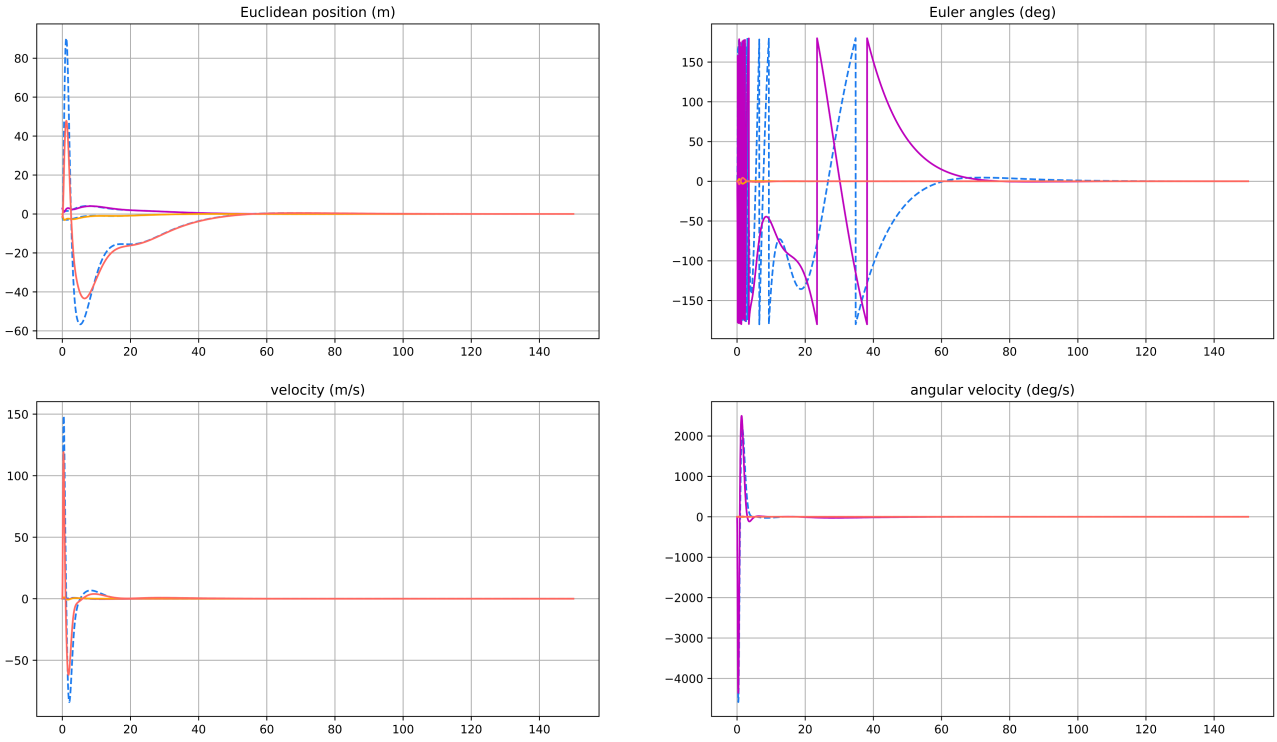

Figure 6.19: Observation errors of quadrotor **2** in the case of noisy parameters (5%) noise

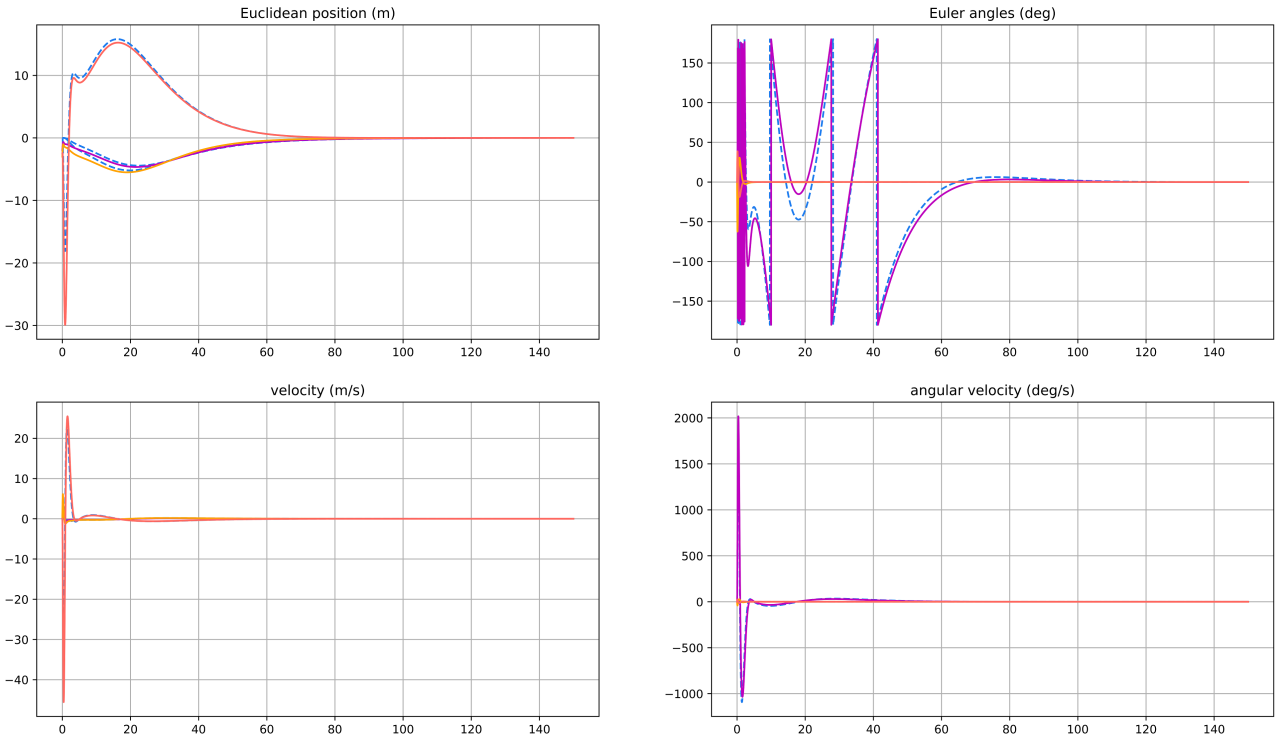

Figure 6.20: Observation errors of quadrotor **1** in the case of noisy parameters (10%) noise

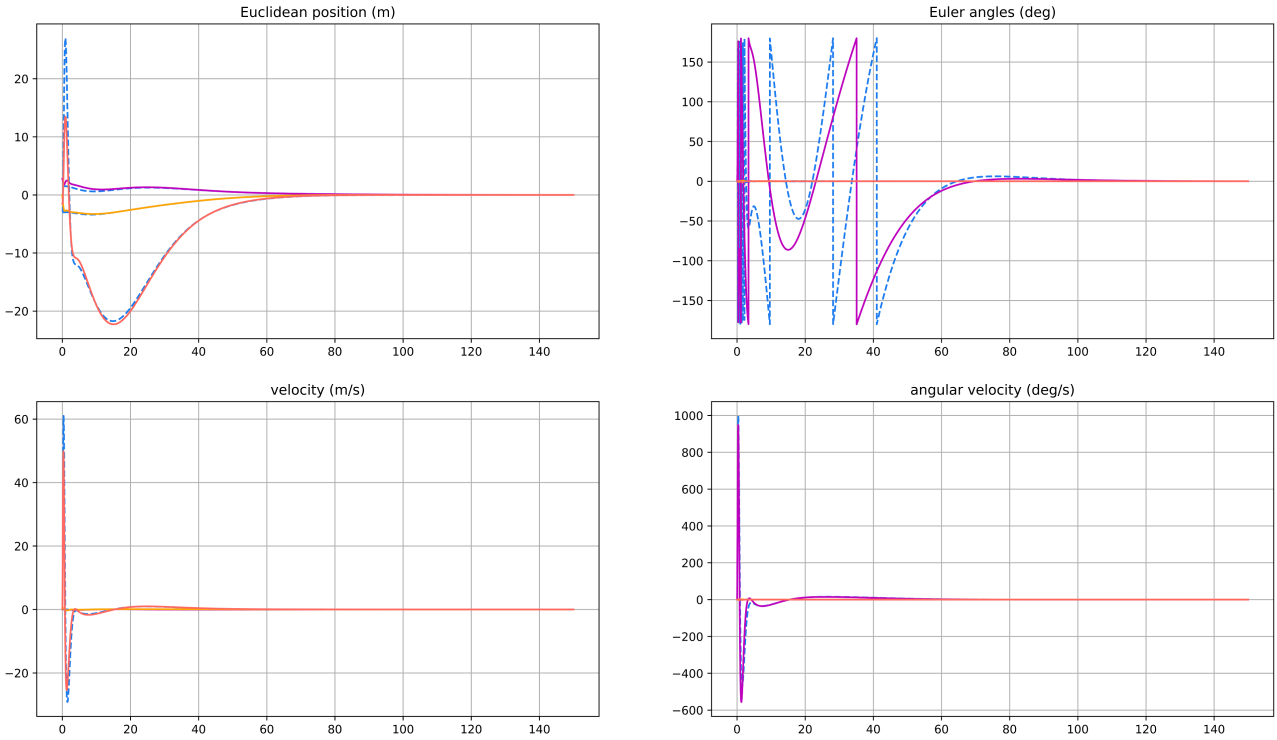

Figure 6.21: Observation errors of quadrotor **2** in the case of noisy parameters (10%) noise

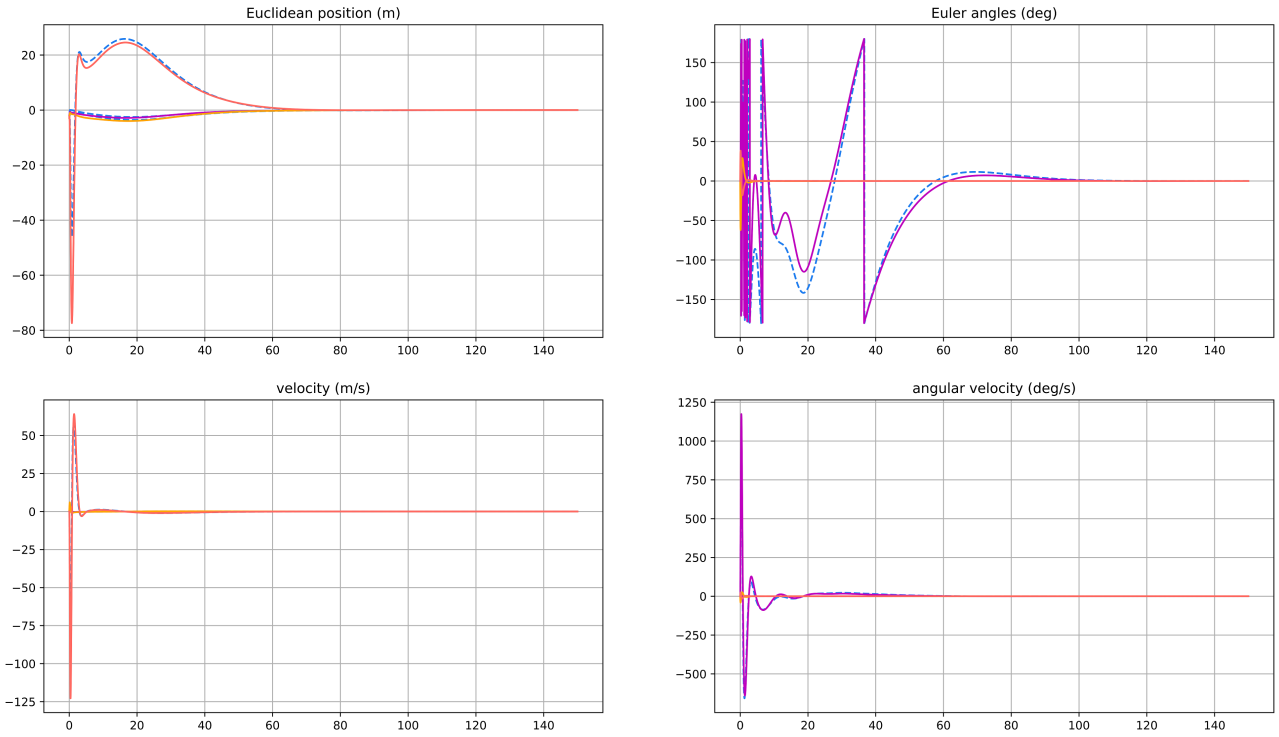

Figure 6.22: Observation errors of quadrotor **1** in the case of noisy parameters (20%) noise

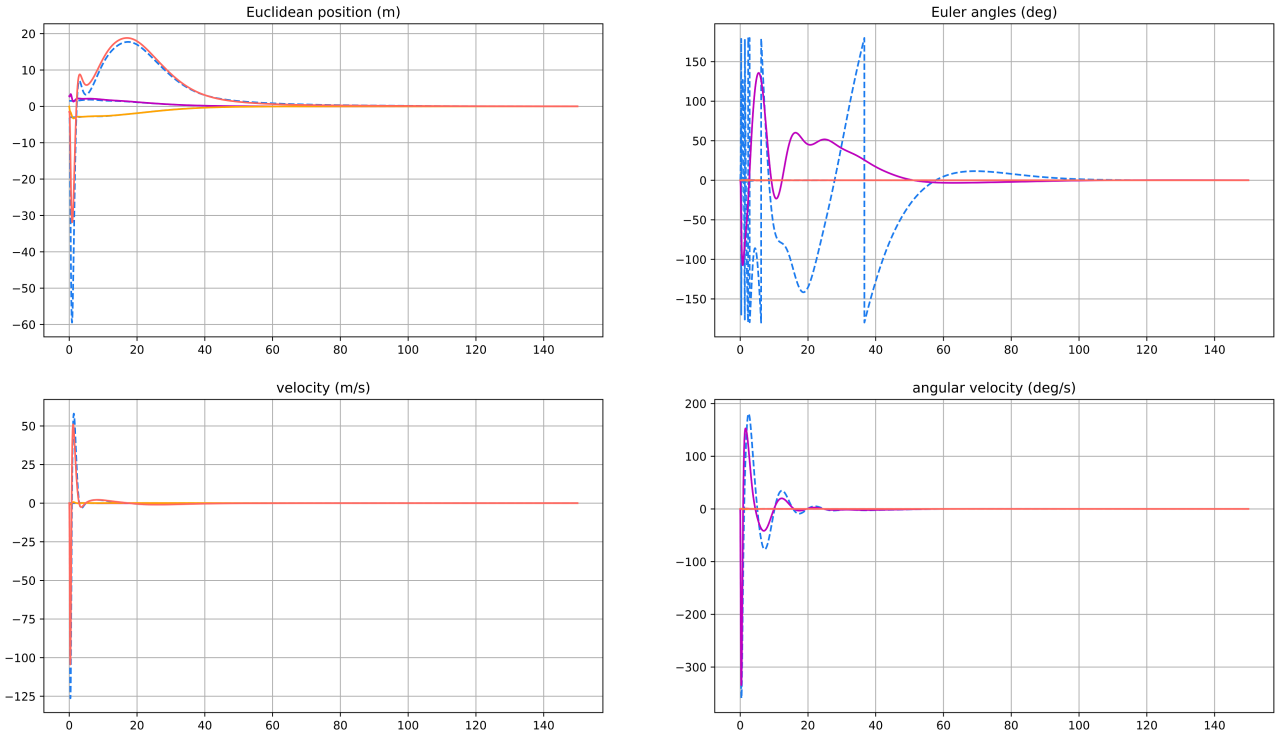

Figure 6.23: Observation errors of quadrotor **2** in the case of noisy parameters (20%) noise

Table 6.2: Settling time when adding noise to the parameters

Parametric error (% of the parameters)	5	10	20
Settling time for observation (s)	80	105	124
Settling time for control (s)	103	122	128

Note that for multi-agent systems evolving in the same environment, it is necessary to add an obstacle avoidance algorithm to avoid collisions. Furthermore, none of the trajectories and controllers presented in this section are feasible in practice. For example, speeds such as 1000deg/s are impossible to achieve, and a quadrotor rotating indefinitely on one of its axes is unrealistic.

Consider U_{thrust}. It is the control that affects the motion of the UAV vertically. Since the motors are all placed on one side of the quadrotor, it is impossible to create a negative thrust. However, using a feedback controller cannot guarantee that this force always remains positive. We can see this for our example in Figures 6.24 and 6.25 which represents the thrust applied to the quadrotor **1** and **2** for a system without any noise, that the thrust is sometimes negative. Therefore, we can say that this controller is impossible to achieve in practice.

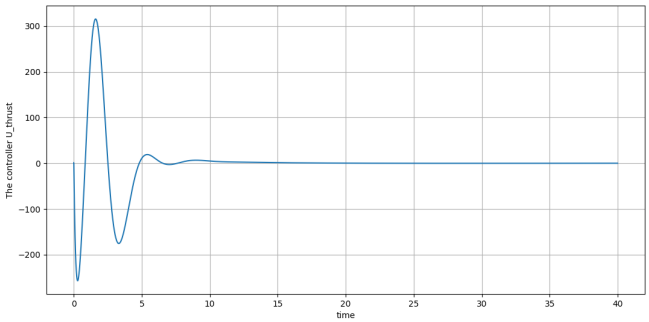

Figure 6.24: The control U_{thrust} of the quadrotor **1** for a system without any disturbances

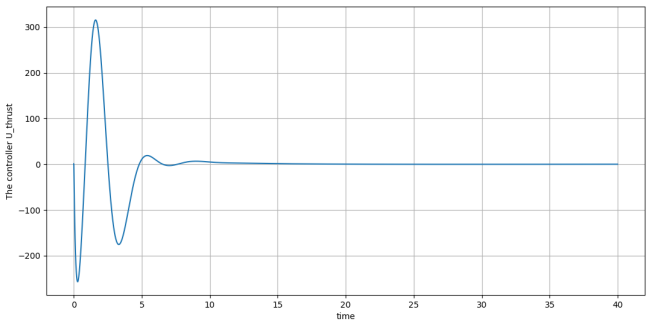

Figure 6.25: The control U_{thrust} of the quadrotor **2** for a system without any disturbances

6.7 Conclusion

In this chapter, we have tested the algorithms and the sensor fault correction estimator presented in the last chapter on a multi-agent UAV system. We first defined the mathematical model of the quadrotor. Then, considering a MAS of 3 drones, we selected one of them to be defective and estimated the states of the defaulting agent using relative measurements and a distributed observer on the clusters we have generated. We have decided to use the Distributed Luenberger Observer for the estimations for its simplicity and robustness and implemented an observer-based controller to stabilize the systems.

While testing the sensor fault correction algorithms and the distributed observer on a quadrotor, we did not design a suitable controller for a UAV, as it would be too complex. Therefore, a future perspective for this work would be to implement an observer-based controller based on Lie groups or nonlinear techniques to test the robustness of this method and monitor its performance in practice. Thus, to see if its settling time is short enough, for instance.

.

CHAPTER 7

PYTHON LIBRARY IMPLEMENTATION

7.1 Introduction

Scientific research is essential to generate new knowledge to understanding better a given phenomenon, develop new ideas and build new technologies. It is more than crucial in today's world. However, it cannot evolve without the technological and computer tools to help it do so. Therefore, we decided to develop a library in Python to assist control engineers, students, and researchers in distributed observation.

In this chapter, we will present the Python module we created to simulate multi-agent systems and the various existing distributed observers. We will review the packages we used and how the classes and functions are organized in this package.

7.2 Overview of the library

To achieve our goal of supporting the scientific community, we have implemented a package in Python that allows the simulation and testing of distributed observers and multi-agent systems. This package is divided into several classes and was implemented using the software presented in section 2.7, chapter 2.

7.2.1 Graph

This class creates and manipulates graphs as defined in section 2.2.1. It can verify if the graph is strongly connected and compute its incidence, adjacency, and degree matrices.

To create a graph, we need to call the class with the chosen number of nodes and optionally an adjacency matrix. If no adjacency matrix were input to the object, this class instantiates a graph with random weighted edges. Upon initialization, the degree matrix, the Laplacian matrix, and the incidence matrix are calculated and can be referred to using the commands $\boldsymbol{G}.Deg$, $\boldsymbol{G}.Lap$ and $\boldsymbol{G}.Incid$ respectively.

This class includes many other methods. For instance, it contains a function that, given a set of vertices, returns a subgraph containing only those vertices and their associated edges. It was designed for the purposes of our project.

To see an illustrative code, see the appendix, figure A.1.

7.2.2 Multi-agent system

This class defines a multi-agent system. It allows computing its joint observability, implementing a multi-agent feedback controller, adding parametric noise, and testing it with different inputs (a step, an impulse, etc). The two types of multi-agent systems defined in 4.8 and 5.1 can be designed using this class.

The class is initialized with the state-space models of the agents and a graph $\mathbf{G}$ associated with the MAS. It contains several functions that can test some of its properties. For instance, the step response, the impulse response, and the forced response can be simulated by choosing parameters such as the time of the simulation, the input u, and the initial values of the states (see figure 3.2, chapter 3). Furthermore, the multi-agent system class can measure the observability index of each agent, define which agents are faulty, and add a feedback controller to the system.

A code is presented in the appendix, figure A.2 to illustrate the functioning of this class.

7.2.3 Quadrotor

This class defines the quadrotor's linear mathematical model, as defined in the previous chapter. It can graph the trajectories and the rotational movements of a quadrotor.

Using parameters such as the mass of the quadrotor, its lift and drag factors, the distances between its wings, its inertial matrix, this class can simulate a quadrotor on a 3D plane.

The way of creating a quadrotor as an object in Python using this class is illustrated in the appendix, figure A.3.

7.2.4 Distributed observer

This class designs different types of distributed observers, calculates the necessary parameters to ensure exponential convergence of the observer, and adds an observer-based feedback controller to the inputted multi-agent system. The distributed estimators can be tested, and their responses plotted with and without parametric or output disturbances. This class also allows to measure some performance criteria and display them.

An object from this class is initialized using an object from the multi-agent class and several other parameters including the initial values of states and their estimates, the type of observer (DLO, DFTO ...), and the standard deviation of the parametric disturbances. The chosen distributed observer can be run with an observer-based controller and with a noisy output.

When initializing the observer, the user enters the multi-agent system on which he desires to apply his observer. He mentions the values of the parameters (these depend on the type of observer chosen) and initializes the states and their estimates. Finally, the choice of noise rates is defined, and the observer is run using the *run_observer()* command. If the system is unstable, an observer-based feedback control law can be implemented in the system.

This class also allows the results of the different observers to be visualized (see figures 6.15 to 6.23, chapter 6). Additionally, it makes it possible to retrieve performance indicators such as the settling time, the steady-state error, or the observation error.

We have illustrated an example of the implementation on this class in the appendix, figure A.4.

.

CHAPTER 8

CONCLUSION

Over the last twenty years, scientists have become aware of the strength of multi-agent systems, and much research has been carried out in this area. control system theory has not failed to adapt to this trend, and several multi-agent control and estimation algorithms have been developed.

Throughout this thesis, we have focused our efforts on the distributed estimation of multi-agent systems. In particular, we have worked on sensor faults correction in MAS; we have presented an algorithm and tested it on a MAS quadrotor under different conditions.

In the first part of this work, we presented four distributed observers, each with its advantages and drawbacks. Three of these estimators were originally designed to estimate the global states of a multi-agent system. In contrast, the last one is a cooperative observer that introduces the notion of coupled measurements. In addition, we proposed a technique to correct sensor faults in a MAS using distributed estimators. We then examined the proposed solution for homogeneous and heterogeneous multi-agent systems. Finally, we simulated our solution on a linearized quadrotor multi-agent system and tested the robustness of the solution. The simulations were implemented in Python using a package we wrote to complement the Python Control library with the distributed estimation problem.

In summary, our contribution is twofold:

- The realization of algorithms for correcting sensor faults in multi-agent systems, using different distributed estimators existing in the literature.
- The realization of a package in Python for simulating multi-agent systems and their estimators.

Several additional studies can be carried out to complement this thesis, such as implementing a distributed estimator that corrects for temporary sensor failures, performing distributed estimation on a group of systems that are not individually observable, or developing an algorithm to define relative measurements based on the availability of sensors in practice.

BIBLIOGRAPHY

[1] Taekyoo Kim, Chanhwa Lee, and Hyungbo Shim. Completely decentralized design of distributed observer for linear systems. *IEEE Transactions on Automatic Control*, 65(11):4664–4678, 2019.

[2] Weixin Han, Harry L Trentelman, Zhenhua Wang, and Yi Shen. A simple approach to distributed observer design for linear systems. *IEEE Transactions on Automatic Control*, 64(1):329–336, 2018.

[3] Taekyoo Kim, Hyungbo Shim, and Dongil Dan Cho. Distributed luenberger observer design. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6928–6933. IEEE, 2016.

[4] Jingbo Wu, Valery Ugrinovskii, and Frank Allgöwer. Cooperative estimation and robust synchronization of heterogeneous multiagent systems with coupled measurements. *IEEE Transactions on Control of Network Systems*, 5(4):1597–1607, 2017.

[5] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[6] Francesco Bullo. *Lectures on network systems*. Kindle Direct Publishing, 2020.

[7] Rudolf E Kalman. On the general theory of control systems. In *Proceedings First International Conference on Automatic Control, Moscow, USSR*, pages 481–492, 1960.

[8] Bong Wie. *Space vehicle dynamics and control*. Aiaa, 1998.

[9] William L. Thompson, Gary C. White, and Charles Gowan. Chapter 3 - enumeration methods. In William L. Thompson, Gary C. White, and Charles Gowan, editors, *Monitoring Vertebrate Populations*, pages 75–121. Academic Press, San Diego, 1998.

[10] Julio H. Braslavsky. Elec4410: Control systems design, lecture 16, controllability and observability, canonical decompositions.

[11] Brian Douglas MATLAB Tech Talk. What is robust control? | robust control, part 1, 2020.

[12] Vipin Jain Electrical 4 u. Steady state error: What is it? (steady-state gain, value formula), 2020.

[13] Katsuhiko Ogata et al. *Modern control engineering*, volume 5. Prentice hall Upper Saddle River, NJ, 2010.

[14] Karl Johan Åström and Richard M Murray. Feedback systems. *An Introduction for Scientists and Engineers, Karl Johan Åström and Richard M. Murray*, 2007.

[15] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.

[16] Wei Ren and Randal W Beard. *Distributed consensus in multi-vehicle cooperative control*, volume 27. Springer, 2008.

[17] What is python? executive summary | python.org. https://www.python.org/doc/essays/blurb/. (Accessed on 06/29/2022).

[18] Numpy. https://numpy.org/. (Accessed on 07/03/2022).

[19] Scipy. https://scipy.org/. (Accessed on 07/03/2022).

[20] Matplotlib — visualization with python. https://matplotlib.org/. (Accessed on 07/03/2022).

[21] control · pypi. https://pypi.org/project/control/. (Accessed on 07/03/2022).

[22] harold/harold at main · ilayn/harold. https://github.com/ilayn/harold/tree/main/harold. (Accessed on 07/03/2022).

[23] Matthew E Taylor, Yang Yu, Edith Elkind, and Yang Gao. *Distributed Artificial Intelligence: Second International Conference, DAI 2020, Nanjing, China, October 24-27, 2020, Proceedings*, volume 12547. Springer Nature, 2020.

[24] Lili Wang and A Stephen Morse. A distributed observer for a time-invariant linear system. *IEEE Transactions on Automatic Control*, 63(7):2123–2130, 2017.

[25] Valery Ugrinovskii. Distributed robust filtering with h consensus of estimates. *Automatica*, 47(1):1–13, 2011.

[26] Haik Silm, Rosane Ushirobira, Denis Efimov, Jean-Pierre Richard, and Wim Michiels. A note on distributed finite-time observers. *IEEE Transactions on Automatic Control*, 64(2):759–766, 2018.

[27] Shih-Ho Wang and Edward Davison. On the stabilization of decentralized control systems. *IEEE Transactions on Automatic Control*, 18(5):473–478, 1973.

[28] Jean-Pierre Corfmat and A Stephen Morse. Decentralized control of linear multivariable systems. *Automatica*, 12(5):479–495, 1976.

[29] Sana Ullah Jan, Young Doo Lee, and In Soo Koo. A distributed sensor-fault detection and diagnosis framework using machine learning. *Information Sciences*, 547:777–796, 2021.

[30] Nasir Mehranbod, Masoud Soroush, and Chanin Panjapornpon. A method of sensor fault detection and identification. *Journal of Process Control*, 15(3):321–339, 2005.

[31] Keith Worden and AP Burrows. Optimal sensor placement for fault detection. *Engineering structures*, 23(8):885–901, 2001.

[32] Omid Shakernia, Yi Ma, T John Koo, and Shankar Sastry. Landing an unmanned air vehicle: Vision based motion estimation and nonlinear control. *Asian journal of control*, 1(3):128–145, 1999.

[33] Omid Shakernia, Yi Ma, T John Koo, Joao Hespanha, and S Shankar Sastry. Vision guided landing of an unmanned air vehicle. In *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No. 99CH36304)*, volume 4, pages 4143–4148. IEEE, 1999.

[34] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. Geometric tracking control of a quadrotor uav on se (3). In *49th IEEE conference on decision and control (CDC)*, pages 5420–5425. IEEE, 2010.

[35] Randal W Beard. Quadrotor dynamics and control. *Brigham Young University*, 19(3):46–56, 2008.

[36] Oualid Araar and Nabil Aouf. Full linear control of a quadrotor uav, lq vs h. In *2014 UKACC International Conference on Control (CONTROL)*, pages 133–138. IEEE, 2014.

[37] Gabriel Hoffmann, Haomiao Huang, Steven Waslander, and Claire Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *AIAA guidance, navigation and control conference and exhibit*, page 6461, 2007.

[38] Samir Bouabdallah and Roland Siegwart. Full control of a quadrotor. In *2007 IEEE/RSJ international conference on intelligent robots and systems*, pages 153–158. Ieee, 2007.

[39] Young-Cheol Choi and Hyo-Sung Ahn. Nonlinear control of quadrotor for point tracking: Actual implementation and experimental tests. *IEEE/ASME transactions on mechatronics*, 20(3):1179–1192, 2014.

[40] Jemie Muliadi and Benyamin Kusumoputro. Neural network control system of uav altitude dynamics and its comparison with the pid control system. *Journal of Advanced Transportation*, 2018, 2018.

[41] Haibo Wang, Zheng Li, Hongyun Xiong, and Xiaohong Nian. Robust h attitude tracking control of a quadrotor uav on so (3) via variation-based linearization and interval matrix approach. *ISA transactions*, 87:10–16, 2019.

APPENDIX A

PYTHON CODES

```
G = Graph(nbr_agent= 3,
            Adj= [[0, 1, 0], [1, 0, 1], [0, 1, 0]])

print("Degree matrix:\n", G.Deg)
print("Laplacian matrix:\n", G.Lap)
print("Incidence matrix:\n", G.Incid)
```

```
This graph is strongly connected
Degree matrix:
 [[1. 0. 0.]
 [0. 2. 0.]
 [0. 0. 1.]]
Laplacian matrix:
 [[ 1. -1.  0.]
 [-1.  2. -1.]
 [ 0. -1.  1.]]
Incidence matrix:
 [[-1.  1.  0.  0.]
 [ 1. -1. -1.  1.]
 [ 0.  0.  1. -1.]]
```

Figure A.1: Example of a code from the Graph class

```
nbr_agent = 2

G = Graph(nbr_agent, [[0, 1], [1, 0]])

A_sys = np.array([[-2, 0],
                  [0, -3]])

B_sys = np.array([[1], [1]])

C1 = np.array([[1, 0]])
C2 = np.array([[0, 1]])

C_sys = (C1, C2)

MAS = MultiAgentSystem(A_sys, B_sys, C_sys, G)

print("It is jointly observable:", MAS.is_jointly_obsv())
print("obsv index", MAS.obsv_index())

MA.step_response(t_max = 7,
                 x0= [-1, 0, 1, 2])
```

Figure A.2: Example of a code from the Multi-agent System class

```
J = [0.0820, 0.0845, 0.1377]
m = 4.34
d = 0.315
lift_factor = 2*10**(-4)
drag_factor = 7*10**(-5)

drone = Quadrotor(m, d, J, lift_factor, drag_factor)

A_sys, B_sys, C_sys = drone.get_state_space()
```

Figure A.3: Example of a code from the Quadrotor class

```
observer = ObserverDesign(multi_agent_system= MA,
                          t_max= 3,
                          x0= x0,
                          x_hat_0= x_hat_0,
                          desired_states= x_d_0,
                          gamma= 6,
                          k0= np.ones(2),
                          std_noise_parameters= 0,
                          std_noise_sensor= 0,
                          std_noise_relative_sensor = 0,
                          input= "impulse")

eig = [-0.12744068, -0.13575947, -0.13013817, -0.12724416, -0.12118274, -0.13229471,
       -0.12187936, -0.14458865, -0.14818314, -0.11917208, -0.13958625, -0.12644475]
observer.feedback_control_with_observer(desired_eig= eig)

observer.run_observer(type_observer = "DLO")

print(observer.steady-state_error)
print(observer.settling_time)
```

Figure A.4: Example of a code from the Distributed Observer class

Figure A.5: Github QR code of the repository of the entire project