RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE

Département d'Automatique

End of Studies Project

For the attainment of an Engineer degree in Control Engineering

---

Fault Tolerant control of nonlinear systems
Iterative observer based approach

HIRECHE Zakaria & HABIA Abdechafi

Under the supervision of **Dr. ACHOUR Hakim** and **Pr. BOUDANA Djamel**

Publicly presented and discussed on (21/06/2025)

Composition of the jury:

President:	Pr. BOUDJEMA Farès	ENP
Promoter:	Dr. ACHOUR Hakim	ENP
Promoter:	Pr. BOUDANA Djamel	ENP
Examiner:	Pr. TADJINE Mohamed	ENP

ENP 2025

Département / ENP, 10 Rue des Frères OUDEK, El Harrach 16200 Alger Algérie.

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE

Département d'Automatique

End of Studies Project

For the attainment of an Engineer degree in Control Engineering

---

Fault Tolerant control of nonlinear systems
Iterative observer based approach

HIRECHE Zakaria & HABIA Abdechafi

Under the supervision of **Dr. ACHOUR Hakim** and **Pr. BOUDANA Djamel**

Publicly presented and discussed on (21/06/2025)

Composition of the jury:

President:	Pr. BOUDJEMA Farès	ENP
Promoter:	Dr. ACHOUR Hakim	ENP
Promoter:	Pr. BOUDANA Djamel	ENP
Examiner:	Pr. TADJINE Mohamed	ENP

ENP 2025

Département / ENP, 10 Rue des Frères OUDEK, El Harrach 16200 Alger Algérie.

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

مخبر التحكم في العمليات التصنيعية
Laboratoire de Commande des Processus

Département d'Automatique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'ingénieur d'état en Automatique

---

Commande tolérante aux défauts des systèmes non linéaires
Approche basée sur les observateurs itératifs

HIRECHE Zakaria & HABIA Abdechafi

Sous la supervision de **Dr. ACHOUR Hakim** et **Pr. BOUDANA Djamel**

Présenté et soutenu publiquement le (21/06/2025)

Membres du jury:

Président:	Pr. BOUDJEMA Farès	ENP
Promoteur:	Dr. ACHOUR Hakim	ENP
Promoteur:	Pr. BOUDANA Djamel	ENP
Examinateur:	Pr. TADJINE Mohamed	ENP

ENP 2025

Département / ENP, 10 Rue des Frères OUDEK, El Harrach 16200 Alger Algérie.

ملخص :

يتناول هذا المشروع موضوع التحكم المقاوم للأعطال (Fault-Tolerant Control) في الأنظمة غير الخطية، باستخدام مقاربة تقديرية تعتمد على المراقبين (Observers). تم تطوير إطار تحكم متكامل من خلال الجمع بين نمذجة تاكاغي-سوجينو الضبابية (Takagi-Sugeno) ومراقبين تكراريين سريعين من نوع k-خطوة، من أجل تقدير أعطال المشغلات وتعويضها في الزمن الحقيقي. تم تصميم مراقبين مركزيين وموزعين لكل من الأنظمة ذات العامل الوحيد والأنظمة متعددة العوامل، مع ضمان الاستقرار باستخدام نظرية ليابونوف (Lyapunov) واللامساواة الخطية المصفوفية (LMI) تم تطبيق المنهجية على روبوتات متحركة ذات دفع تفاضلي، حيث حافظ النظام على تتبع المسار بدقة رغم وجود أعطال في المشغلات. تم تنفيذ بنية التحكم في بيئة ROS واختبارها في محاكاة روبوتية واقعية. تؤكد النتائج موثوقية الطريقة وإمكانية تطبيقها عملياً. يساهم هذا العمل في تطوير استراتيجيات تحكم قوية وقابلة للتكيّف للأنظمة الذاتية التشغيل في ظل وجود أعطال.

الكلمات المفتاحية: التحكم المقاوم للأعطال، المراقبون التكراريون، نماذج تاكاغي-سوجينو الضبابية، تقدير أعطال المشغلات، ROS، الروبوتات ذات الدفع التفاضلي، الأنظمة متعددة العوامل، التقدير الموزع

Résumé

Ce projet traite la commande tolérante aux défauts (FTC) pour les systèmes non linéaires en utilisant une stratégie d'estimation basée sur les observateurs itératifs. Un cadre de commande complet est développé en combinant la modélisation floue de Takagi-Sugeno avec des observateurs itératifs rapides de type k-étapes pour l'estimation et la compensation en temps réel des défauts d'actionneurs. Des observateurs centralisés et distribués sont conçus pour des systèmes mono-agent et multi-agents, avec des garanties de stabilité assurées à l'aide de la théorie de Lyapunov et des inégalités matricielles linéaires. La méthode est appliquée à des robots différentiels afin d'assurer le suivi de trajectoire même en présence de défauts. L'architecture de commande est implémentée sous ROS et testée dans un environnement de simulation robotique réaliste. Les résultats confirment la fiabilité et la faisabilité pratique de la méthode. Ce travail contribue à l'évolution des stratégies de commande robustes et adaptatives pour les systèmes autonomes soumis à des défauts.

Mots-clés : Commande tolérante aux défauts, Observateurs itératifs, Modèles flous de Takagi-Sugeno, Estimation de défauts d'actionneurs, ROS, Robots différentiels, Systèmes multi agents, Estimation distribuée

Abstract

This project deals with fault-tolerant control (FTC) for nonlinear systems using an observer-based estimation strategy. A complete framework is developed by combining Takagi-Sugeno fuzzy modeling with fast iterative k-step observers for real-time actuator fault estimation and compensation. Centralized and distributed observers are designed for both single-agent and multi-agent systems, with stability ensured using Lyapunov theory and Linear Matrix Inequalities. The method is applied to differential drive mobile robots, maintaining trajectory tracking despite actuator faults. The control architecture is implemented in ROS and tested in a realistic robotic simulation. Results confirm the method's reliability and practical applicability. This work advances robust and adaptive control strategies for autonomous systems under fault conditions.

Keywords : Fault Tolerant Control, Iterative Observers, Takagi-Sugeno Fuzzy Models, Actuator Fault Estimation, ROS, Differential Drive Robots, Multi Agent systems, distributed estimation

Acknowledgements

First and foremost, I would like to express my deepest gratitude to ALLAH for His guidance and support. Without His will, I would not have had the strength to move forward or accomplish anything. All praise and thanks be to Him for everything.

Secondly, I extend my heartfelt thanks and dedicate this achievement to my parents, who have supported me tirelessly throughout my journey. Their joy means more to me than all the sleepless nights and efforts I've endured. I only hope to continue making them proud for the rest of my life.

I also wish to sincerely thank my sister. Without her, I would not have been able to complete this project. Her unwavering support, encouragement, and guidance from my earliest days until now have been a true source of strength. You are my role model, and I continue to follow your lead.

I would also like to express my sincere gratitude to Chikh Amine, thank you for raising me and showing me how to live with purpose. Your teachings and values have shaped who I am, and they will remain with me for the rest of my life, Mourad, thank you for your unwavering support, you have been a true companion throughout my journey, and your presence has been a source of strength.

I am deeply grateful to all my teachers and educators, from primary school to this very year in national polytechnic school. I still remember the valuable advice and lessons they shared. I hope to be a worthy result of their dedication and hard work.

Lastly, I want to thank all my loyal friends and colleagues who stood by me during difficult times. Each one of you who contributed, in any way, to what I have achieved today thank you from the bottom of my heart. You are simply the best.

HIRECHE Zakaria

Acknowledgements

First and foremost, I would like to express my deepest gratitude to ALLAH for His countless blessings and for granting us the strength and patience to complete this work. Truly, He is the source of all grace and goodness.

I am sincerely thankful to my parents for their unwavering support, their numerous sacrifices, and for raising me with strong values and respect. Their encouragement has been a foundation throughout my journey.

I would also like to extend my heartfelt thanks to my sisters for their continuous support, their words of encouragement, and the care they have always shown me during my studies.

I wish to express my sincere gratitude to my extended family for their presence and kind words, and to my friends for their companionship, assistance, and unwavering encouragement throughout my academic path. Their support has played an important role in helping me stay motivated and balanced.

I am especially grateful to my close friends Aïssa, Zaki, and Abdeldjebar for their brotherhood, their sincere friendship, and the precious help they provided me. Their presence has been a great source of strength, support, and loyalty throughout this journey.

I would also like to thank my fellow classmates and study colleagues for their valuable help and collaboration throughout these years. Sharing this academic journey with them has been a source of mutual support and growth.

Finally, I am deeply grateful to all my teachers and educators, from primary school to my current studies at the National Polytechnic School. I still carry with me the valuable lessons and advice they have shared. Thank you for your guidance, your generosity in sharing knowledge, and for your support, patience, and empathy throughout the years.

HABIA Abdechafi

Contents

List of Tables

List of Figures

Acronym list

- LPV : Linear Parameter Varying
- LTI : Linear Time Invariant
- FTC : Fault Tolerant Control
- AFTC : Active Fault Tolerant Control
- PFTC : Passive Fault Tolerant Control
- FDI : Fault Detection and Isolation
- T-S : Takagi-Sugeno
- MPV : Measurable Premise Variables
- NMPV : Non-Measurable Premise Variables
- PDC : Parallel Distributed Compensation
- PI : Proportional Integral
- PMI : Proportional Multi Integral
- LMI : Linear Matrix Inequalities
- FE : Fault Estimation
- MAS : Multi-agent systems
- ROS : Robot Operating System
- RViz : ROS Visualization
- RQT : ROS Qt-based framework
- LIDAR : Light Detection and Ranging
- IMU : Inertial Measurement Unit
- GPS : Global Positioning System
- URDF : Unified Robot Description Format
- UAV : Unmanned Aerial Vehicle
- DDS : Data Distribution Service
- GUI : graphical user interface

General Introduction

Modern control systems are expected to operate with high reliability and stability, even in the presence of internal faults and external uncertainties. This requirement is particularly critical in domains such as autonomous vehicles, robotics, and aerospace systems, where failures can have significant consequences on safety and system performance. As systems become more complex and interconnected, it is no longer sufficient to rely solely on redundancy or manual intervention for fault management. This has led to the emergence and development of Fault-Tolerant Control (FTC), a field dedicated to designing control strategies capable of detecting, estimating, and compensating for faults in real time.

Observer-based FTC has proven to be a promising and efficient approach, especially when redundancy in hardware is not feasible. By leveraging mathematical models and system outputs, observers can estimate faults dynamically and enable the controller to take corrective actions without disrupting system operation. However, when dealing with nonlinear systems, traditional linear observers fall short due to the inherent complexity and variability of the dynamics involved. To address this, Takagi-Sugeno (T-S) fuzzy models have been widely adopted. These models offer a flexible way to represent nonlinear behavior through a convex combination of linear models, while still allowing for the use of powerful analysis and design tools.

In this work, we propose an FTC method tailored for nonlinear systems modeled by T-S fuzzy representations. The core idea is to use iterative observers to estimate actuator faults with high speed and accuracy. These observers can be implemented both in centralized and distributed architectures, making the method suitable for a wide range of applications including multi-agent systems. The approach is validated through theoretical analysis, simulations, and practical implementation in robotic platforms using the Robot Operating System (ROS).

The report is structured as follows:

- **Chapter 1** introduces the theoretical foundations of Takagi-Sugeno fuzzy modeling and stability analysis. It explains how nonlinear systems can be represented using fuzzy rules and local linear models, and discusses stability criteria using Lyapunov-based methods.

- **Chapter 2** presents the concept of fault-tolerant control in the context of T-S fuzzy systems. It classifies FTC techniques into passive and active approaches and discusses their integration with fuzzy modeling frameworks. This chapter prepares the ground for the observer-based strategy developed in the following sections.

- **Chapter 3** focuses on the design of iterative k-step observers for fault estimation. The chapter first addresses the single-agent case, then extends the approach to distributed multi-agent systems using graph theory and Kronecker products. Observer gain synthesis conditions are derived via linear matrix inequalities, and simulation results are presented to validate the estimation performance.

- **Chapter 4** applies the proposed FTC method to differential drive mobile robots. It shows how actuator faults can be effectively estimated and compensated during trajectory

tracking tasks. The simulation scenarios highlight the effectiveness of the control scheme in realistic nonlinear dynamics.

- **Chapter 5** details the implementation on ROS of the method using the Robot Operating System. It describes the simulation setup, communication structure, and deployment process. The results demonstrate the practical feasibility and performance of the proposed FTC approach in a robotic environment.

From the existing literature, the findings of this work bridges theoretical developments in fault estimation of nonlinear systems with practical applications in robotics and automation. The proposed observer-based control scheme offers a fast, lightweight, and reliable solution to the growing need for autonomy and fault resilience in modern control systems.

Chapter 1

Generalities on Modeling and Stability of Fuzzy Models

Introduction

In modern control systems, the stability and robustness of nonlinear models are essential aspects of system performance. Among the various modeling approaches, Takagi-Sugeno (T-S) fuzzy models offer an effective framework for representing complex nonlinear systems using a set of local linear models weighted by membership functions. This chapter explores the fundamental principles of T-S modeling, stability analysis using Linear Matrix Inequalities (LMIs), and control synthesis techniques such as Parallel Distributed Compensation (PDC). The objective is to develop strategies that ensure system stability while maintaining high performance in the presence of uncertainties and external disturbances.

The chapter is structured as follows: we first address the fuzzy modeling of nonlinear systems, providing the necessary theoretical background and formulation. Next, we analyze the stability of T-S systems and discuss stabilization techniques, including the reference tracking problem. Following this, we introduce the concept of fault diagnosis, outlining key terminologies and methodologies relevant to this project. Through this structured approach, we aim to provide a comprehensive understanding of the stability and control of fuzzy systems while considering fault diagnosis as an integral aspect of robust system design.

1.1 State space representation

Physical processes are often represented by models described in the following form (explicit state-space representation)

$$\begin{cases} \dot{x} = f(x(t), u(t)) \\ y = h(x(t)) \end{cases} \tag{1.1}$$

where x represents the state variables describing the internal state of the system, u and y are the system's input and output variables, respectively, and f and h represent linear and/or nonlinear functions.

Controlling complex nonlinear systems is challenging due to modeling difficulties. Simplified models may lack accuracy, while detailed models can be impractical. The multi-model approach

offers a balanced alternative by representing system behavior across operating regions [1]. Two main methods exist: direct construction, which may lose information, and the more common sector-based nonlinear representation.

1.2 Multi-model approach

The multi-model approach has garnered significant interest since the publication of the works by Johansen and Foss [2] it consists more precisely, of reducing the complexity of the system by decomposing its operating space into a finite number of operating zones.

The T-S approach decomposes a nonlinear system into r linear sub-models, each valid in a specific operating region. As illustrated in Figure 1.1, the global behavior is approximated by interpolating these local models using activation functions. The union of all subdomains forms the overall operating domain, enabling a global multi-model representation [3].

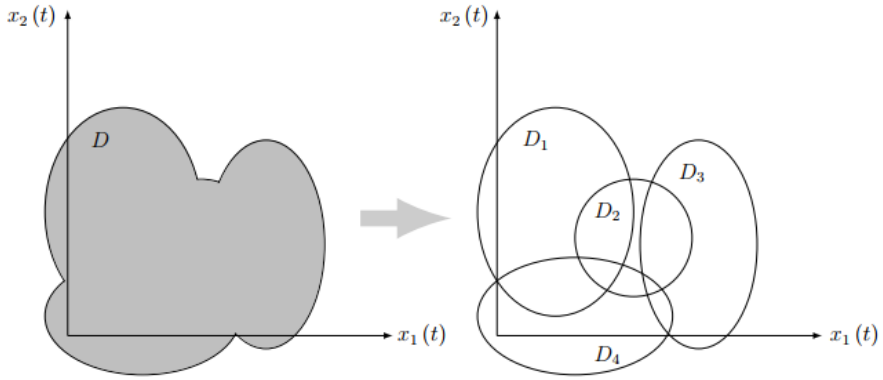


Figure 1.1: Principle of the Takagi-Sugeno approach

- **Sub-model:** It is the model that represents the behavior of the nonlinear system in a specific operating region.
- **Premise variable:** Also known as the decision variable $z(t)$, it influences the activation functions $h(t)$ and may include measurable or unmeasurable internal or external system variables.
- **Activation function:** Determines the contribution level of each local sub-model based on the system's operating region. It ensures smooth transitions between models and depends on decision variables.

$$\mu_i(\xi(t)) = \frac{h_i(z(t))}{\sum_{i=1}^{n} h_i(z(t))} \tag{1.2}$$

The activation functions can be constructed either from discontinuous derivative functions (such as triangular or trapezoidal functions) or from continuous derivative functions (such as Gaussian functions). They are chosen to satisfy the following convex sum properties:

$$\begin{cases} 0 \leq \mu_i(z(t)) \leq 1 \\ \sum_{i=1}^{n} \mu_i(z(t)) = 1 \end{cases} \tag{1.3}$$

Activation functions built from an exponential law are often used for the continuous case.

1.3 Fuzzy modeling and Takagi-Sugeno models

Coupled-state multi-models or Takagi-Sugeno (T-S) fuzzy models [4] are also referred to as TS dynamic fuzzy models. This model is based on the use of a set of fuzzy rules to describe a global nonlinear system in terms of a set of local linear models interpolated by fuzzy membership functions.

The TS fuzzy modeling approach effectively describes complex nonlinear systems while reducing the number of required rules. It also enables systematic stability analysis and control law synthesis by combining fuzzy logic with classical control theory.

TS fuzzy models consist of fuzzy If-Then rules, where the antecedent represents the operating conditions, and the consequent is a local linear or affine model that describes the nonlinear system at that operating point. The i^{th} fuzzy rule is of the form:

If z_i is F_i, then:

$$\begin{cases} \dot{x}_i(t) = A_i x(t) + B_i u(t) \\ y_i(t) = C_i x(t) \end{cases} \tag{1.4}$$

where $z(t)$ is the decision variable, F_i the i^{th} fuzzy set. $x(t) \in \mathbb{R}^n$ is the state vector, $u(t) \in \mathbb{R}^n$ the control vector, $A_i \in \mathbb{R}^{n \times n}$ the local dynamic matrix, and $B_i \in \mathbb{R}^{n \times m}$ the input matrix.

We obtain the following expression for $\dot{x}(t)$:

$$\begin{cases} \dot{x}(t) = \sum_{i=1}^{r} \mu_i(z(t))(A_i x(t) + B_i u(t)) \\ y(t) = \sum_{i=1}^{r} \mu_i((t)) C_i x(t) \end{cases} \tag{1.5}$$

where $\mu_i(z(t))$ is the membership function (or activation function) of the i^{th} fuzzy set.

r is the number of fuzzy sets (i.e., the number of local linear models).

1.3.1 Multi-model structures

A multi-model is a set of sub-models aggregated through an interpolation mechanism that characterizes the overall dynamic behavior of a system. A multi-model is defined by the number of its sub-models, their structure, and the choice of activation functions.

In the literature, two main families of multi-models are identified based on the use of the state vector [5].

1.3.1.1 Decoupled structure

The decoupled structure, or multi-models with decoupled states, are multi-models proposed by [6] where the sub-models each have an independent state vector figure 1.2.

They are governed by the following state equations:

$$\dot{x}_i(t) = A_i x_i(t) + B_i u(t) \tag{1.6}$$

$$y_i(t) = C_i x_i(t) \tag{1.7}$$

The state vector will be given by:

$$\dot{x}(t) = \sum_{i=1}^{r} \mu_i(z(t)) x_i(t) \tag{1.8}$$

And the global output will be given by:

$$y(t) = \sum_{i=1}^{r} \mu_i(z(t)) C_i x_i(t) \tag{1.9}$$

The structure of the decoupled multi-model, in turn, introduces a certain flexibility in the modeling stage. Indeed, it allows the introduction of sub-models whose state vectors can be of different dimensions and thus stands out from the classically used multi-model structures.

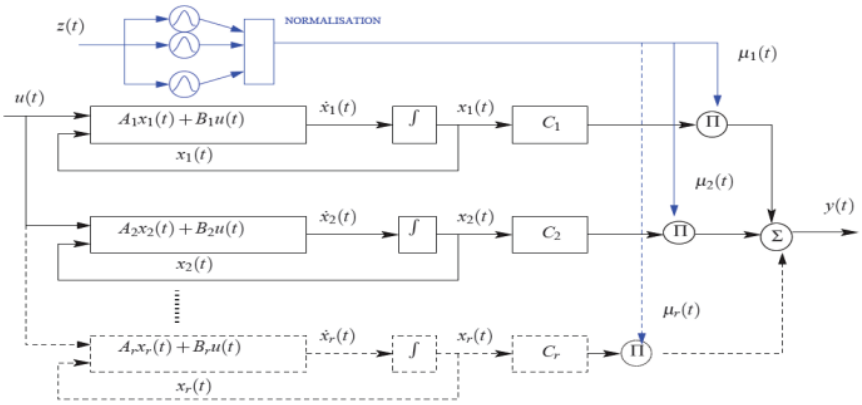


Figure 1.2: Decoupled multi-model structure

1.3.1.2 Coupled structure

The coupled state multi-model structure, or Takagi-Sugeno fuzzy models, are widely used for stability analysis, control, or synthesis of fuzzy observers. In this structure, the sub-models share a common state and are governed by the following equations:

$$\dot{x}(t) = \sum_{i=1}^{r} \mu_i(z(t)) \left(A_i x(t) + B_i u(t)\right) \tag{1.10}$$

$$y(t) = \sum_{i=1}^{r} \mu_i(z(t)) C_i x(t) \tag{1.11}$$

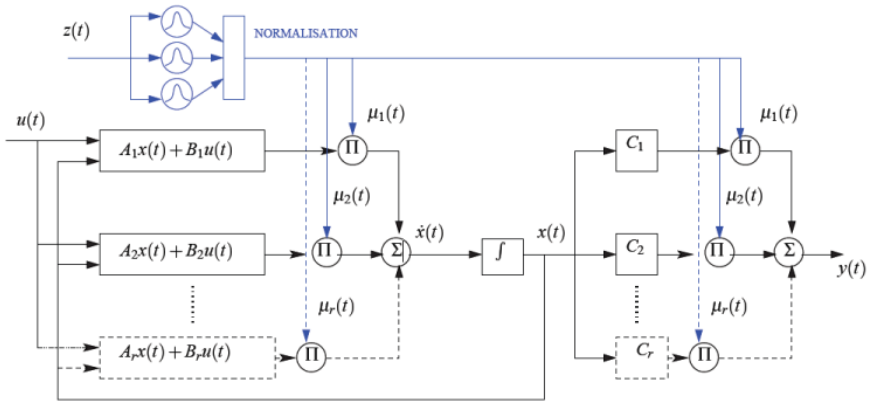


Figure 1.3: Coupled multi-model structure

1.3.2 Techniques for Obtaining T-S Multimodels

Many research studies have been dedicated to the modeling of nonlinear systems using a multi-model approach. Indeed, there is no specific methodology capable of leading to a unique multi-model representation of a system [7]. In all cases, the development of a multi-model raises four major issues:

1. The choice of the decision variable $z(t)$ of the system, which allows indexing the activation functions.
2. The decomposition of the system's operating space into a finite number of operating regions.
3. The determination of the structure of the multi-model and the parametric identification of each sub-model.
4. The choice of the method for obtaining the multi-model.

The real challenge with fuzzy systems is determining how to obtain local linear models. In the literature, there are three main ways to derive these models.

1.3.2.1 Models Obtained through Identification

Representing a nonlinear system as a multi-model simplifies its identification to estimating local linear sub-models and activation functions. Parameter estimation is performed by numerical optimization methods that minimize the error between the estimated output $\hat{y}(t)$ and the measured output $y_m(t)$.

1.3.2.2 Models Obtained through Linearization

In this case, the analytical form of the nonlinear model of the physical process is available. The system is linearized around different appropriately chosen operating points.

We will represent the nonlinear system 1.1 using a multi-model approach, consisting of several local linear or affine models obtained by linearizing the nonlinear system around an arbitrary operating point $(x_i, u_i) \in \mathbb{R}^n \times \mathbb{R}^m$,[8]

$$\begin{cases} \dot{x}_m(t) = \sum_{i=1}^{r} \mu_i(z(t))(A_i x_m(t) + B_i u(t) + D_i) \\ y_m(t) = \sum_{i=1}^{r} \mu_i(z(t))(C_i x_m(t) + E_i u(t) + N_i) \end{cases} \tag{1.12}$$

with

$$A_i = \left.\frac{\partial f(x,u)}{\partial x}\right|_{(x,u)=(x_i,u_i)}, \quad B_i = \left.\frac{\partial f(x,u)}{\partial u}\right|_{(x,u)=(x_i,u_i)} \tag{1.13}$$

$$C_i = \left.\frac{\partial h(x,u)}{\partial x}\right|_{(x,u)=(x_i,u_i)}, \quad E_i = \left.\frac{\partial h(x,u)}{\partial u}\right|_{(x,u)=(x_i,u_i)} \tag{1.14}$$

$$D_i = f(x_i, u_i) - A_i x_i - B_i u_i, \quad N_i = h(x_i, u_i) - C_i x_i - E_i u_i \tag{1.15}$$

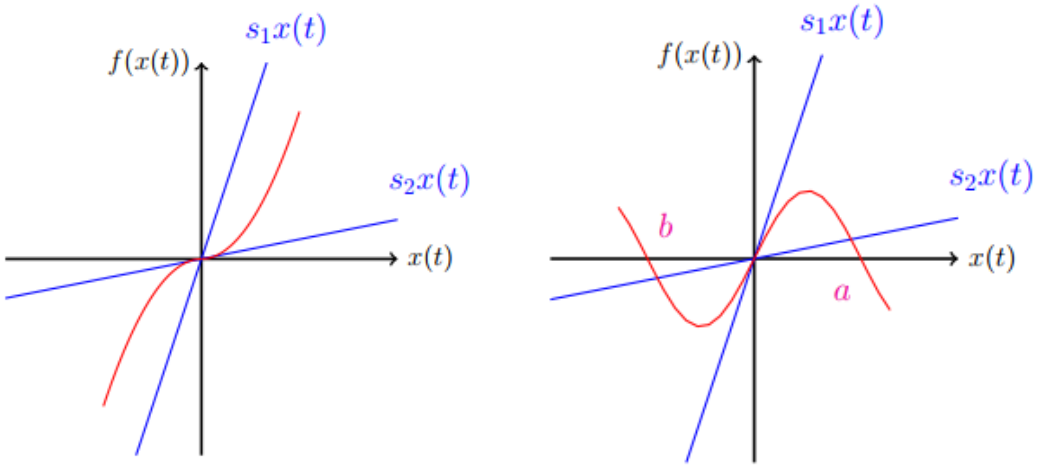
Note that in this case, the number of local models (r) depends on the desired modeling accuracy, the complexity of the nonlinear system, and the choice of activation function.

1.3.2.3 Models Obtained through Nonlinear Sector

This approach aims to accurately represent the nonlinear system within a compact state variable space. In this context, it is sometimes challenging to find a global sector for the nonlinear system.

For this reason, a local nonlinear sector is considered, as illustrated in the figure 1.4.

The advantage of such a method is that it does not introduce approximation errors and reduces the number of models compared to the linearization method [9].



(a) Global sector (b) Local sector

Figure 1.4: Non-linear sectors

Consider the continuous nonlinear system :

$$\dot{x} = f(x(t)) + g(x(t)) \cdot u(t) \tag{1.16}$$

where: $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$, $f(x(t)) \in \mathbb{R}^p$, and $g \in \mathbb{R}^{p \times m}$.

Lemma 1.1 [10] Let $z(x(t))$ be a bounded function from $[a, b]$ to $\mathbb{R}$ for all $x \in [a, b]$ with $[a, b] \in \mathbb{R}^+$. Then there exist two functions $F^1(x(t))$ and $F^2(x(t))$ and two scalars α and β such that:

$$z(x(t)) = \alpha * F^1(x(t)) + \beta * F^2(x(t)) \tag{1.17}$$

with:

$$F^1(x(t)) + F^2(x(t)) = 1, \quad F^1(x(t)) \geq 0, \quad F^2(x(t)) \geq 0 \tag{1.18}$$

A decomposition of $z(x(t))$ over $[a, b]$ is given by:

$$\begin{cases} \beta = \min_{x \in [a,b]} z(x(t)) \\ \alpha = \max_{x \in [a,b]} z(x(t)) \end{cases} \tag{1.19}$$

$$\begin{cases} F^1(x(t)) = \frac{z(x(t)) - \beta}{\alpha - \beta} \\ F^2(x(t)) = \frac{\alpha - z(x(t))}{\alpha - \beta} \end{cases} \tag{1.20}$$

Under the assumptions of continuity and boundedness of the functions $f(x(t))$ and $g(x(t))$ in model 1.16, with $f(0) = 0$ and $g(0) = 0$, they can be rewritten in the following form:

$$\begin{cases} f(x(t)) = \sum_{i=1}^{r} \mu_i(z(t)) A_i x(t) \\ g(x(t)) = \sum_{i=1}^{r} \mu_i(z(t)) C_i x(t) \end{cases} \tag{1.21}$$

The model 1.17 becomes:

$$\begin{cases} \dot{x}(t) = \sum_{i=1}^{r} \mu_i(z(t))(A_i x(t) + B_i u(t)) \\ y(t) = \sum_{i=1}^{r} \mu_i(z(t))(C_i x(t) + D_i u(t)) \end{cases} \tag{1.22}$$

In this case, the obtained multi-model representation exactly matches the nonlinear model over the considered compact interval.

1.3.2.4 Example 1.1

We consider the following 2-dimensional nonlinear model [11] :

$$\dot{x}(t) = f(x(t)) = \begin{cases} \dot{x}_1(t) = \sin(x_1(t)) \\ \dot{x}_2(t) = -x_2^3(t) \end{cases} \tag{1.23}$$

This model can be rewritten as follows:

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}\frac{\sin(x_1(t))}{x_1(t)} & 0\\ 0 & -x_2^2(t)\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} \tag{1.24}$$

The equation 1.24 presents two nonlinearities denoted as $z_1(x_1(t))$ and $z_2(x_2(t))$ respectively, such that:

$$z_1(x_1(t)) = \frac{\sin(x_1(t))}{x_1(t)}, \quad z_2(x_2(t)) = -x_2^2(t)$$

We observe that the nonlinear term $z_1(x_1(t))$ is bounded $\forall x(t) \in \mathbb{R}^n$, i.e.,

$$z_1(x_1(t)) \in \left[\frac{\sin(x_{10})}{x_{10}}, 1\right] \quad (\text{where} \quad \frac{\sin(x_{10})}{x_{10}} = \min(\sin(x_1)/x_1) \approx -0.217) \tag{1.25}$$

while the term

$$z_2(x_2(t)) \quad \text{can only be bounded on a compact set defined by} \quad x_2(t) \in [-a, a], \quad \text{with} \quad a > 0. \tag{1.26}$$

We can transform the nonlinear terms $z_1(t)$ and $z_2(t)$ $\forall x(t) \in \mathbb{R}^n \times [-a, a], \quad a > 0, \quad$ such that:

$$z_1(x_1(t)) = F_1^1(x_1(t)) \cdot 1 + F_1^2(x_1(t)) \cdot \frac{\sin(x_{1_0})}{x_{1_0}}$$

$$z_2(x_2(t)) = F_2^1(x_2(t)) \cdot 0 - F_2^2(x_2(t)) \cdot a^2$$

The fuzzy sets are given by:

$$F_1^1(x_1(t)) = \frac{(\sin(x_1(t))/x_1(t)) - (\sin(x_{10})/x_{10})}{1 - (\sin(x_{10})/x_{10})}, \quad F_1^2(x_1(t)) = \frac{1 - (\sin(x_1(t))/x_1(t))}{1 - (\sin(x_{10})/x_{10})}$$

$$F_2^1(x_2(t)) = 1 - \frac{x_2(t)^2}{a^2}, \quad F_2^2(x_2(t)) = \frac{x_2(t)^2}{a^2}$$

We then obtain the corresponding TS model from the four possible combinations of the bounds of the nonlinear terms $z_1(x(t))$ and $z_2(x(t))$, described by the following matrices f_i:

1. If $x_1(t)$ is $F_1^1(x_1(t))$ and $x_2(t)$ is $F_2^1(x_2(t))$ then:

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & a^2\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} \tag{1.27}$$

2. If $x_1(t)$ is $F_1^1(x_1(t))$ and $x_2(t)$ is $F_2^2(x_2(t))$ then:

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} \tag{1.28}$$

3. If $x_1(t)$ is $F_1^2(x_1(t))$ and $x_2(t)$ is $F_2^2(x_2(t))$ then:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} \frac{\sin(x_{1_0})}{x_{1_0}} & 0 \\ 0 & a^2 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{1.29}$$

4. If $x_1(t)$ is $F_1^2(x_1(t))$ and $x_2(t)$ is $F_2^2(x_2(t))$ then:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} \frac{\sin(x_{1_0})}{x_{1_0}} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{1.30}$$

The inference of the fuzzy system is given by:

$$\dot{x}(t) = \sum_{i=1}^{4} \mu_i(z(t)) A_i x(t) \tag{1.31}$$

where

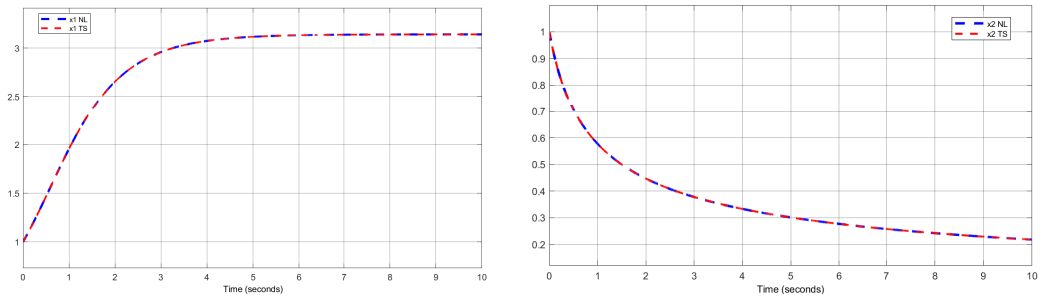
$$\begin{cases} \mu_1(z(t)) = F_1^1(z_1(t)) \times F_2^1(z_2(t)) \\ \mu_2(z(t)) = F_1^2(z_1(t)) \times F_2^2(z_2(t)) \\ \mu_3(z(t)) = F_1^2(z_1(t)) \times F_2^1(z_2(t)) \\ \mu_4(z(t)) = F_1^2(z_1(t)) \times F_2^2(z_2(t)) \end{cases} \tag{1.32}$$

This transformation leads to a certain number of local LTI (Linear Time Invariant) models depending on the number of nonlinearities contained in the function $f(t)$. In general, if $f(t)$ has k nonlinear terms, then the TS model consists of at most 2^k local models.

It should also be noted that the adopted transformation strongly influences the analysis results since the matrices of the local models directly depend on it according to predefined objectives.

From this example, we have shown that the number of rules in an exact TS model increases based on the nonlinearities considered in the nonlinear model, which leads to greater conservativeness in the results.

The results of the simulation of the fuzzy and real models for $x(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ are shown in 1.5a and 1.5b. It is evident that the two models are identical, meaning that the fuzzy model can exactly represent the original system in the pre-specified domains.



(a) x_{1NL} *and* x_{1TS} (b) x_{2NL} *and* x_{2TS}

Figure 1.5: Simulation of the real system and the fuzzy system example 1.1

1.3.3 Criteria for choosing premise variables

The selection of premise variables plays a key role in ensuring stability [12], controllability, and observability, without affecting the resolution of LMIs due to the convexity of interpolation functions. As shown in [13] and [14], the choice influences the number of sub-models and the global model structure, with methods proposed to reduce LMI complexity and computation time. However, as noted in [14], the complexity of LMI conditions increases with the number of state variables used as premise variables.

The four guiding principles mentioned in [13] are:

- The control matrix of the quasi-LPV system must not be a null matrix. This is a necessary condition for controlling the system.
- A quasi-LPV form with a minimal number of premise variables is preferable.
- A T-S model with a minimal number of submodels should be chosen.
- The premise variable vector should depend on a minimal number of state variables [14].

1.4 Stability of Takagi-Sugeno fuzzy models

Our objective is to ensure the stability of Takagi-Sugeno fuzzy systems, and we have prioritized the use of quadratic stabilization of the system through Lyapunov's second method [13]. This stability is guaranteed if the conditions, formulated as a set of Linear Matrix Inequalities LMI from the following theorems, are satisfied. We consider the following fuzzy TS system:

$$\dot{x}(t) = \sum_{i=1}^{r} \mu_i(z(t)) A_i x(t) \tag{1.33}$$

Consider the classical quadratic Lyapunov function given by:

$$V(x(t)) = x(t)^T P x(t) \tag{1.34}$$

Theorem 1. *[10] The TS fuzzy multi-model described by 1.33 is asymptotically stable if there exists a positive definite matrix P such that the following LMIs are satisfied:*

$$A_i^T P + P A_i < 0, \quad i = 1, \ldots, r \tag{1.35}$$

We draw the reader's attention to the fact that numerous examples show that a T.S fuzzy system may contain unstable sub-models while still being stable as a whole, and vice versa. The stability conditions in Theorem 1 are conservative since the premise variables are not taken into account. The conservatism issue of these stability conditions is mitigated at the cost of a significant number of LMIs.

1.5 Stabilization of a T-S type multi-model

To ensure the stability of a T-S model, we rely on the synthesis of a stabilizing control law. To achieve this, drawing inspiration from stability analysis results of dynamic systems [7], we derive state-feedback control synthesis conditions. The conditions on the control gains obtained are not necessarily formulated directly as an LMI problem. Indeed, in some cases, nonlinear matrix inequalities arise, requiring a set of matrix transformations to linearize them. In this context, several fuzzy control laws have been proposed in the literature. Among the most commonly used is the state-feedback control law, known as Parallel Distributed Compensation (PDC).

1.5.1 Stabilizing control by the PDC approach with state feedback

The main idea behind the design of the PDC controller is primarily based on stability analysis using a quadratic Lyapunov function. The concept is to assign a control rule to each controller based on the corresponding rule of the T-S fuzzy model, ensuring compensation of the model. As a result, a fuzzy controller is obtained, which also shares the same fuzzy set as the fuzzy model. Figure 1.6 illustrates the operating principle of the PDC control strategy [10].

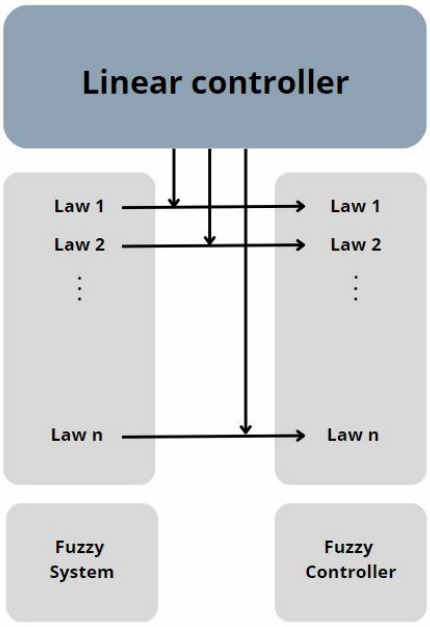


Figure 1.6: Principle of the PDC controller

Let the continuous T-S model 1.5 in closed-loop.

The PDC controller rules can be written as follows: Rule i: If $z_1(t)$ and F_i^1 and ... and $z_p(t)$ and F_i^p Then $u(t) = -K_i x(t)$ $i = 1, \ldots, r$, which is a state feedback controller in the consequence part. The fuzzy controller is represented by:

$$u(t) = -\sum_{i=1}^{r} \mu_i(z(t)) K_i x(t) \tag{1.36}$$

where $K_i : i = 1, \ldots, r$ is the local feedback gain relative to the i-th model, with the same $\mu_i(z(t))$ as those of the fuzzy model .

By combining 1.22 and 1.36, the representation of the global closed-loop model with a PDC control law is given by:

$$\dot{x}(t) = \sum_{i=1}^{r}\sum_{j=1}^{r} \mu_i(z(t))\mu_j(z(t))(A_i - B_iK_j)x(t) \tag{1.37}$$

We can write 1.37 as follows:

$$\dot{x}(t) = \sum_{i=1}^{r}\sum_{j=1}^{r} \mu_i(z(t))\mu_j G_{ij}x(t) \tag{1.38}$$

With :

$$G_{ij} = A_i - B_iK_j$$

Theorem 2. *[10] The continuous fuzzy model 1.33 is globally asymptotically stable via the PDC control law 1.36 if there exists a common positive definite matrix $P = P^T > 0$ that satisfies the following matrix inequalities:*

$$G_{ii}^T P + PG_{ii} < 0, \quad \forall i = 1, \ldots, r \tag{1.39}$$

$$\left(\frac{G_{ij} + G_{ji}}{2}\right)^T P + P\left(\frac{G_{ij} + G_{ji}}{2}\right) \leq 0, \quad i < j \leq r \tag{1.40}$$

Consider the linear matrix inequalities (LMIs) with variables P and K_i. To make the inequalities linear, we define a new variable $X = P^{-1}$ and use $M_i = K_iX$. The LMIs are then expressed in terms of X and M_i:

$$\begin{cases} X > 0 \\ XA_i^T + A_iX - B_iM_i - M_i^TB_i^T < 0 \quad \forall i = 1, \ldots, r \\ X(A_i + A_j)^T + (A_i + A_j)X - (B_iM_j + B_jM_i) - (B_iM_j + B_jM_i)^T < 0 \quad i < j \leq r \end{cases} \tag{1.41}$$

The state feedback gains are given by:

$$K_i = M_iX^{-1} \quad i = 1, \ldots, r \tag{1.42}$$

1.5.1.1 Example 1.2

We will consider an example to apply the PDC method, aiming at the synthesis of a stabilizing state feedback control law 1.36.

Let the nonlinear system be represented by a Takagi-Sugeno fuzzy multi-model, defined as follows [15]:

$$\dot{x}(t) = \sum_{i=1}^{2} \mu_i(z(t)) \left(A_ix(t) + B_iu(t)\right)$$

Such that :

$$A_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -109.8 & 100 & 0 & 0 \\ 100 & -100 & 0 & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 100 & 0 & 0 \\ 100 & -100 & 0 & 0 \end{bmatrix} \quad B_1 = B_2 = B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The initial vector x_0 at $t = 0$ is given by: $x_0 = [1, 3, 2, -1]^T$:

The activation functions μ_i, $i = 1, 2$ are given by:

$$\begin{cases} \mu_1(x) = \frac{1-\tanh(x_1)}{2} \\ \mu_2(x) = 1 - \mu_1(x) \end{cases}$$

Before applying the PDC method, the system was unstable. We first verify its controllability. The controllability matrices associated with the pairs (A_1, B_1) and (A_2, B_2) both have full rank (rank $= n = 4$).The resolution of 1.41 give us the gains of state feedback :

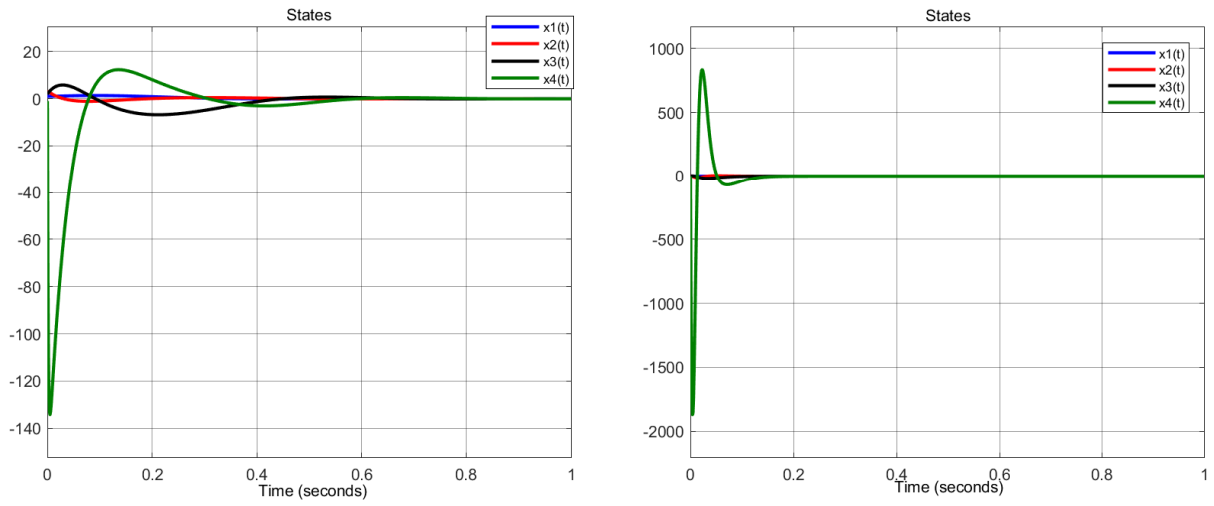
$$K_1 = 10^4 \times \begin{bmatrix} 1.5515 & 2.2845 & 0.2908 & 0.0613 \end{bmatrix} \; ; \; K_2 = 10^4 \times \begin{bmatrix} 1.7714 & 2.5925 & 0.3320 & 0.0697 \end{bmatrix}$$

To make the system dynamics faster, we can impose poles in a specific region. For example, we can constrain them to the left half-plane below a given constant, such as $\alpha = 40$, to ensure a desired performance.

$$K_1 = 10^5 \times \begin{bmatrix} 8.5963 & 0.6613 & 0.4122 & 0.0040 \end{bmatrix} \; ; \; K_2 = 10^5 \times \begin{bmatrix} 7.8324 & 0.6146 & 0.3791 & 0.0038 \end{bmatrix} \tag{1.43}$$

Figure 1.7 illustrates the result of the applied control. We observe that the system stabilization is well established. The graph shows that, without pole placement, the system states stabilize around the origin after a transient period of approximately 0.6 seconds 1.7a. However, when poles are placed in the previously defined region, the convergence to the origin is significantly faster, occurring within 0.2 seconds 1.7b.

This result validates the proper functioning of the implemented PDC controller, as it successfully achieves the regulation and stabilization objectives. We can conclude that the tuning of the PDC gains is satisfactory, ensuring both stabilization and the desired level of precision.



(a) Without pole imposition (b) With pole imposition

Figure 1.7: Temporal evolution of the states of the closed-loop system with a PDC control

1.6 Tracking trajectories of T-S model :

Most research on multi-model systems has focused on stabilization strategies, ensuring local or global stability. However, few studies have addressed reference or trajectory tracking problems, an approach based on a combination of sliding mode control and PDC control for TS models has been proposed by [7],[10] and [16]

The trajectory tracking problem for a nonlinear system modeled by the T-S model (1.5) involves designing a control law $u(t)$ that enables the system to follow a desired reference trajectory. This problem is typically reformulated as a stabilization problem. The tracking error is defined as the difference between the state vector of the system and that of the reference model, as follows: $e(t) = x(t) - x_d(t)$

The dynamics of the tracking error is given by:

$$\dot{e}(t) = \dot{x(t)} - \dot{x_d}(t) \tag{1.44}$$

By replacing 1.5 into 1.44 and adding the term $\sum_{i=1}^{r} \mu_i(z(t))A_i \left(x_d(t) - x_d(t)\right)$.
Equation 1.44 becomes:

$$\dot{e}(t) = \sum_{i=1}^{r} \mu_i(z) \left(A_i\tilde{x}(t) + B_i u(t) + A_i x_d(t)\right) - \dot{x}_d(t) \tag{1.45}$$

In equation 1.45, we introduce a new variable $\tau(t)$ satisfying the following relation:

$$\sum_{i=1}^{r} \mu_i(z(t))B_i\tau(t) = \sum_{i=1}^{r} \mu_i(z(t))B_i u(t) + \sum_{i=1}^{r} \mu_i(z(t))A_i x_d(t) - \dot{x}_d(t) \tag{1.46}$$

where $\tau(t)$ is a new fuzzy controller that will be synthesized based on the PDC technique. Using the last equation, the derivative of the tracking error 1.45 can be written as follows:

$$\dot{e}(t) = \sum_{i=1}^{r} \mu_i(z(t)) \left(A_i e(t) + B_i\tau(t)\right) \tag{1.47}$$

The output of the fuzzy controller is determined by the following summation:

$$\tau(t) = -\sum_{i=1}^{r} \mu_i(z(t))K_i e(t) \tag{1.48}$$

We substitute into equation 1.45 and obtain:

$$\dot{e}(t) = \sum_{i=1}^{r}\sum_{j=1}^{r} \mu_i(z(t))\mu_j(z(t))G_{ij}e(t) \tag{1.49}$$

with:

$$G_{ij} = A_i - B_i K_j \tag{1.50}$$

The problem thus reduces to a stabilization problem, where the goal is to compute the gains K_i to stabilize $e(t)$ at the origin. The LMIs are obtained in a similar manner to those in the

previous section concerning stabilization.
If we manage to stabilize $e(t) = x(t) - x_d(t)$ at the origin, then the state $x(t)$ tends to follow the trajectory of the reference model $x_d(t)$.

Remark : The stability conditions for trajectory tracking are the same as those for the stabilization problem of T-S fuzzy models using a PDC-type control law. This means that the state feedback gains K_i can be directly obtained by solving the stabilization problem. However, to ensure accurate tracking, it is necessary to impose specific pole placements.

1.6.1 Nonlinear Control Law :

There are two different methods for determining the control law: the first one proposed by [10], and the second by [7]. In the following, we will detail the control law using both methods.
. **1st method :** To determine the desired variables $x_d(t)$ and the control law $u(t)$, we use equation 1.46:

$$\sum_{i=1}^{r} \mu_i(z(t))B_i(u(t) - \tau(t)) = -\sum_{i=1}^{r} \mu_i(z(t))A_i x_d(t) + \dot{x}_d(t). \tag{1.51}$$

By defining:

$$A(x) = \sum_{i=1}^{r} \mu_i(z(t))A_i, \quad g(x) = \sum_{i=1}^{r} \mu_i(z(t))B_i \tag{1.52}$$

Equation 1.51 can be rewritten in the following form:

$$g(x)(u(t) - \tau(t)) = -A(x)x_d(t) + \dot{x}_d(t). \tag{1.53}$$

The existence of the control $u(t)$ depends on the form of $g(x)$. The input matrix $g(x)$ is assumed to be full column rank.

$$g(x) = \begin{bmatrix} 0_{n-m} \\ \vdots \\ B(x) \end{bmatrix}, \quad A(x) = \begin{bmatrix} A_{n-m} \\ \vdots \\ A_m \end{bmatrix}, \quad x_d(x) = \begin{bmatrix} x_{d_{n-m}} \\ \vdots \\ x_{d_m} \end{bmatrix} \tag{1.54}$$

Equation 1.53 can be rewritten in matrix form as:

$$\begin{bmatrix} 0_{n-m} \\ \vdots \\ B(x) \end{bmatrix} (u - \tau) = \begin{bmatrix} \dot{x}_{d_{n-m}} - A(x)_{n-m}x_d(t) \\ \vdots \\ \dot{x}_{d_m} - A_m(x)x_d(t) \end{bmatrix} \tag{1.55}$$

From the second equation of 1.55, the nonlinear control law is given by:

$$u(t) = -\sum_{i=1}^{r} \mu_i(z(t))K_i\tilde{x}(t) + B^{-1}(x)(\dot{x}_{d_m}(t) - A_m(x)x_d(t)). \tag{1.56}$$

The variables $x_d(t)$ can be easily extracted from the first equation of 1.55.
. **2nd method :** There is also another method, presented as follows.
We consider this reference tracking problem by designing a control law that ensures both stabilization and good reference tracking.
To achieve this objective, we have adopted an improved PDC control law :

The classic PDC control law is given by:

$$u_{PDC}(t) = \sum_{j=1}^{r} \mu_j(x(t)) \left(-K_j x(t)\right) \tag{1.57}$$

And a fuzzy pre-compensator

$$u_{pc}(t) = \sum_{j=1}^{r} \mu_j(x(t)) \left(P_j y_{ref}(t)\right) \tag{1.58}$$

This pre-compensator is an approximation obtained by computing the gain P_j for each sub-model while neglecting the interactions between them. Thus, P_j is computed exactly as in a linear system.

The proposed improved PDC control law is given as follows:

$$u(t) = \sum_{j=1}^{r} \mu_j(x(t)) \left(-K_j x(t) + P_j y_{ref}(t)\right) \tag{1.59}$$

where K_j are the gains of the PDC law, computed by solving the LMIs 1.41

The local gains P_j of the pre-compensator are computed using the following formula :

$$P_j = \left(-C_j (A_j - BK_j)^{-1} B_j\right)^{-1} \tag{1.60}$$

Finally, $y_{ref}(t)$ is the desired reference.

1.6.2 Example :

In this example, we will consider the same system analyzed in Example (1.2), with pole placement and trajectory tracking using both methods.
With :

$$A(x) = \sum_{i=1}^{r} \mu_i(z(t)) A_i = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -109.8\mu_1 & 100 & 0 & 0 \\ 100 & -100 & 0 & 0 \end{bmatrix}$$

$$g(x) = \sum_{i=1}^{r} \mu_i(z(t)) B_i = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \mu_1 + \mu_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \implies B = 1$$

$$x_d = \begin{bmatrix} x_{1d} \\ x_{2d} \\ x_{3d} \\ x_{4d} \end{bmatrix}$$

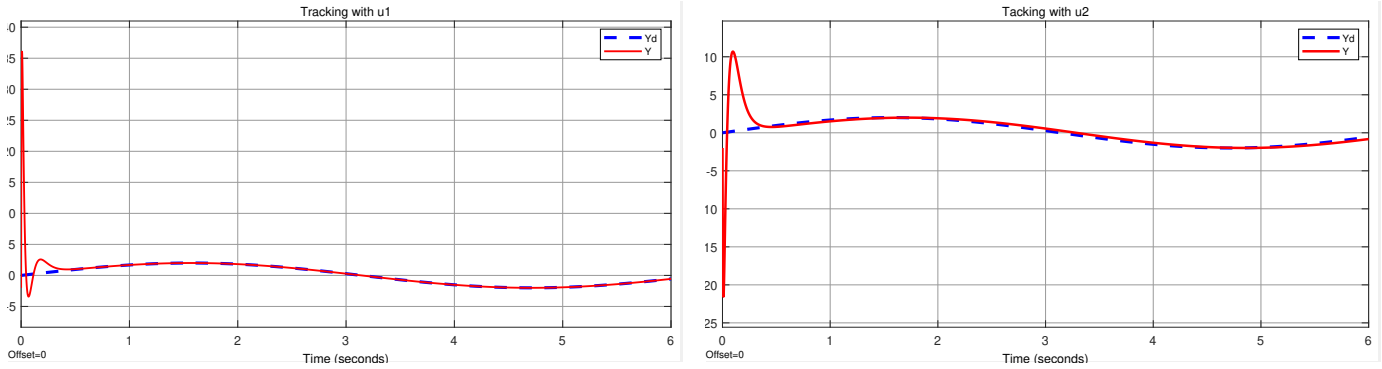
$$u_1 = -\mu_1 * K1 * e - \mu_2 * K2 * e + \dot{x}_{dm} - Am * xd$$

$$u_2 = \mu_1 \left(-K_1 x + P_1 y_d\right) + \mu_2 \left(-K_2 x + P_2 y_d\right)$$

From the figure 1.8, we can observe that both methods are capable of tracking a reference.

In the figure 1.8a, we used first method [10] by applying Control Law 1.56.

In the figure 1.8b we used second method [7] by applying Control Law 1.59.



(a) Tracking using 1.56 (b) Tracking using 1.59

Figure 1.8: Tracking reference

However, it is important to note that the second method exhibits a shift in the case of high-frequency periodic reference signals.

Conclusion

In this chapter, we studied the stability of Takagi-Sugeno fuzzy systems using a quadratic Lyapunov function and designed a PDC-based control law to ensure system stabilization. The stability conditions were formulated as LMIs and solved using convex optimization tools. While the conventional PDC approach guarantees stability, it does not inherently ensure reference tracking. To overcome this limitation, we introduced a fuzzy pre-compensator that improves both stability and tracking performance. The results demonstrate the effectiveness of this approach in enhancing the robustness of fuzzy control systems. Future work could focus on optimizing the pre-compensator design and exploring adaptive strategies to further refine system performance.

Chapter 2

Fault tolerant control for T-S systems

Introduction

Faults in actuators, sensors, or internal components are common in real-world control systems and can lead to performance degradation or instability. Fault-Tolerant Control (FTC) aims to maintain acceptable system behavior despite such faults, either by designing robust control laws (passive approach) or by detecting and compensating for faults in real time.

In this chapter, we focus on observer-based FTC strategies for nonlinear systems represented by Takagi-Sugeno (T-S) models. After reviewing the classification of FTC techniques, we present the synthesis of observers under measurable premise variables (MPV), including Proportional-Integral (PI) and Proportional Multi-Integral (PMI) observers. These observers enable both state estimation and fault reconstruction, allowing for real-time compensation of constant and time-varying actuator faults.

2.1 Generalities of Fault-Tolerant Control

In most practical engineering systems, sensor, actuator, and component faults are inevitable events that may occur at any time. Once faults appear, the control system may experience performance degradation and even instability. Therefore, it is crucial to study Fault-Tolerant Control (FTC) and its related challenges [17].

Since its introduction in 1971, the goal has been to enhance the safety of modern industrial technologies by maintaining system stability and ensuring acceptable control performance in the presence of failures. With FTC, the impact of faults can be mitigated without halting the production process [18].

2.2 Classification of fault-tolerant control techniques

2.2.0.1 Classification of Fault-Tolerant Control FTC

FTC has been classified in references [19, 18, 20] into two approaches: Passive FTC (PFTC) and Active FTC (AFTC). This classification is based on the design structure, the mathematical method used, the control performance, and the severity of the fault.

As shown in Figure 2.1, the active approach involves system reconfiguration or fault accommodation, whereas the passive approach relies on robust control.

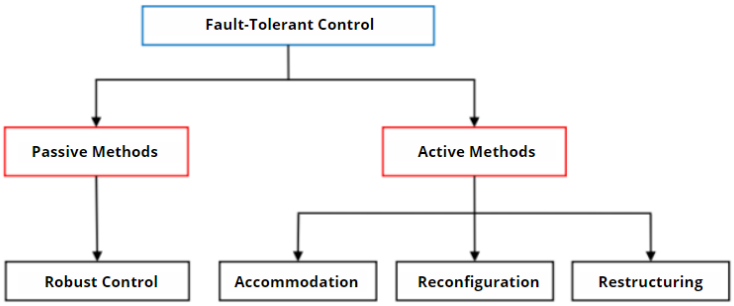


Figure 2.1: Classification of FTC techniques

2.2.1 Passive approach

The passive approach uses robust control techniques, such as H_∞ and sliding mode control, to handle faults treated as disturbances. It requires no fault detection or control reconfiguration but offers limited tolerance to small-magnitude faults.

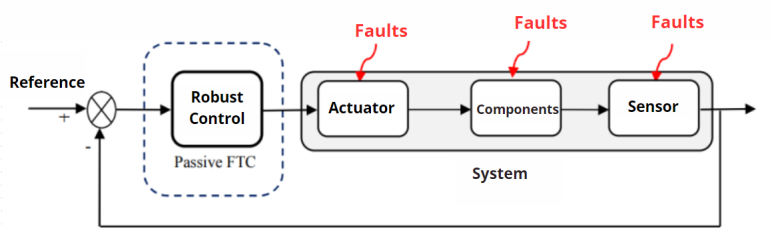


Figure 2.2: Passive FTC

2.2.2 Active approach

AFTC methods are more prevalent in the literature than passive methods due to their superior performance and ability to handle a wide range of faults. AFTC methods react proactively by reconfiguring the control law to maintain system stability and performance, even in the presence of unexpected faults. This process requires a Fault Detection and Isolation (FDI) module and a reconfiguration mechanism.

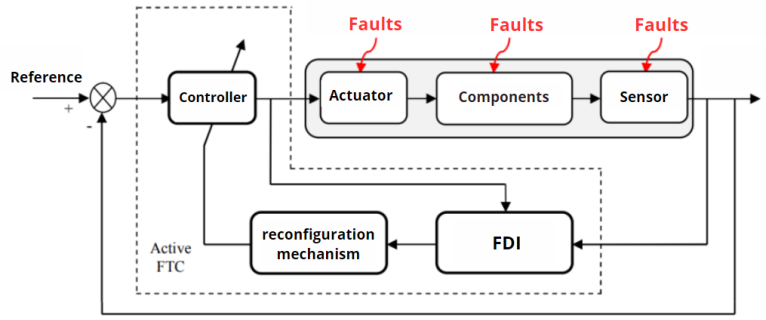


Figure 2.3: Active FTC

Active techniques are generally classified into three types based on their mechanism of action on the system and the type of fault that occurs: fault accommodation, control system reconfiguration, and control system restructuring.

- Fault Accommodation
- System Reconfiguration
- Restructuring

2.2.3 Stabilization by T-S observers

2.2.3.1 Non-linear observers

The application of nonlinear observers to Takagi-Sugeno (T-S) systems primarily aims at state estimation in complex systems where the dynamics can be modeled as weighted linear submodels with nonlinear functions [12]. Here are some key points to consider:

1. Diversity of approaches : Various methods have been developed to estimate the states of nonlinear systems, including the extended Kalman filter, high-gain observers.
2. Unknown input observers : These have been designed for bilinear systems and systems with Lipchitz-type nonlinearity.
3. Complexity of multi-model T-S systems : When the structure of the T-S model relies on Non-Measurable Premise Variables (NMPV), state estimation becomes more challenging.
4. Extensions and improvements : Recent studies have enhanced the robustness of T-S observers by proposing methods that decouple certain unknown inputs and minimize the effect of uncertainties on estimation errors.

2.2.3.2 T-S observers with MPV

The T-S observer with MPV has a T-S structure and uses the same state representation as a Luenberger-type observer. Its state representation is given by:

$$\begin{cases} \dot{\hat{x}}(t) = \sum_{i=1}^{N} \mu_i(z) \left(A_i \hat{x}(t) + B_i u(t) + L_i \Big(y(t) - \hat{y}(t) \Big) \right) \\ \hat{y}(t) = C\hat{x}(t) \end{cases} \tag{2.1}$$

where $\hat{x}(t)$ represents the estimated state vector. To determine the gains L_i of the T-S observers (2.1), a stability study of the system generating the state estimation error must be carried out.

In order to improve the temporal performance of the diagnosis of a faulty engine, a pole placement observer was proposed in [18]. A stability study using Lyapunov's theory and conditions formulated by Linear Matrix Inequalities (LMIs) was carried out.

The synthesis of the observer then involves calculating the gain matrix L. The state estimation error is given by:

$$e(t) = x(t) - \hat{x}(t) \tag{2.2}$$

By differentiating, we obtain:

$$\dot{e}(t) = \dot{x}(t) - \dot{\hat{x}}(t) \tag{2.3}$$

By substituting the expressions for $\dot{x}$ and $\dot{\hat{x}}$, we have:

$$\dot{e}(t) = A_i x(t) + B_i u(t) - A_i \hat{x}(t) - B_i u(t) - L_i\Big(y(t) - \hat{y}(t)\Big) \tag{2.4}$$

$$\dot{e}(t) = A_i(x(t) - \hat{x}(t)) - L_i C_j(x(t) - \hat{x}(t)) \tag{2.5}$$

$$\dot{e}(t) = (A_i - L_i C_j)e(t) \tag{2.6}$$

It is noted that all pairs (A_i, C_i) must be observable.

Theorem 3. *[7] The estimation error described by equation (2.6) is asymptotically stable if there exists a common positive definite matrix P such that the following conditions are satisfied:*

$$\begin{aligned} G_{ii}^T P + P G_{ii} < 0 \qquad & \text{for } i \in \{1, 2, \ldots, r\} \\ \left(\frac{G_{ij} + G_{ji}}{2}\right)^T P + P\left(\frac{G_{ij} + G_{ji}}{2}\right) \leq 0 \qquad & \text{for } i < j \leq r \end{aligned} \tag{2.7}$$

$$G_{ij} = A_i - W_i C_j$$

The observer gains are obtained from the following equation:

$$L_i = P^{-1} W_i$$

2.2.3.3 T-S observers with NMPV

T-S observers based on NMPV have been proposed in [14] and [21]. The state representation of a T-S observer with NMPV is given as follows:

$$\begin{cases} \dot{\hat{x}}(t) = \sum_{i=1}^{N} \mu_i(\hat{z}) \left(A_i \hat{x}(t) + B_i u(t) + L_i\Big(y(t) - \hat{y}(t)\Big)\right) \\ \hat{y}(t) = C\hat{x}(t) \end{cases} \tag{2.8}$$

When the premise variables are unknown, their factorization becomes impossible, and the dynamics of the state estimation error are written as follows:

$$\dot{e}(t) = \sum_{i=1}^{N} \mu_i(z) \left(A_i x(t) + B_i u(t)\right) - \sum_{i=1}^{N} \mu_i(\hat{z}) \left(A_i \hat{x}(t) + B_i u(t) + L_i e(t)\right) \tag{2.9}$$

In our case, we will work with measurable premise variables to simplify the implementation and facilitate result validation for the iterative observer. Choosing measurable premise variables allows us to avoid the complexity associated with estimating these variables in real-time, which can introduce additional uncertainties and degrade observer performance.

2.3 Fault-tolerant control actuators case with MPV

In this section, the premise variables are assumed to be measurable [22]. The fault-tolerant control strategy is studied for two types of observers: the first concerns the Proportional-Integral (PI) observer, and the second focuses on the Proportional Multi-Integral (PMI) observer.

Consider the fault-free system described by the following TS model:

$$\begin{cases} \dot{x}(t) = \sum_{i=1}^{r} \mu_i(x(t)) \Big(A_i x(t) + B_i u(t) \Big) \\ y(t) = \sum_{i=1}^{r} \mu_i(x(t)) \Big(C_i x(t) + D_i u(t) \Big) \end{cases} \tag{2.10}$$

Where:

- $x(t) \in \mathbb{R}^n$: State vector,
- $u(t) \in \mathbb{R}^m$: Input vector,
- $y(t) \in \mathbb{R}^p$: Output vector.

The matrices A_i, B_i, C_i, and D_i are constants with appropriate dimensions. The functions $\mu_i(x(t))$ represent activation functions dependent on the system's state. It is assumed that the state is fully measurable.

The system affected by actuator faults is described as follows:

$$\begin{cases} \dot{x}(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(A_i x(t) + B_i \left(u(t) + f(t) \right) \right) \\ y(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(C_i x(t) + D_i \left(u(t) + f(t) \right) \right) \end{cases} \tag{2.11}$$

where $f(t) \in \mathbb{R}^{n_f}$ is the fault vector.

The goal is to design an FTC law such that the state $x(t)$ of the faulty system (3.1) converges to the state of the fault-free system (3.1). The proposed FTC law is given as:

$$u(t) = -\sum_{i=1}^{r} \mu_i(x(t)) K_i \hat{x}(t) - \hat{f}(t) \tag{2.12}$$

The matrices K_i are determined to ensure the stability of the closed-loop system.

The control law 1.36 requires the estimation of the fault vector $f(t)$. This estimation is obtained using a PI observer that simultaneously estimates the state of the system and the fault vector.

2.3.1 PI observer

The structure of the PI observer for system (3.1) is given by [22]:

$$\begin{cases} \dot{\hat{x}}(t) = \sum_{i=1}^{r} \mu_i\Big(x(t)\Big) \Big(A_i \hat{x}(t) + B_i \Big(u(t) + \hat{f}(t) \Big) + L_{Pi} \Big(y(t) - \hat{y}(t) \Big) \Big) \\ \hat{y}(t) = \sum_{i=1}^{r} \mu_i\Big(x(t)\Big) \Big(C_i \hat{x}(t) + D_i \Big(u(t) + \hat{f}(t) \Big) \Big) \\ \hat{f}(t) = \sum_{i=1}^{r} \mu_i\Big(x(t)\Big) L_{Ii} \Big(y(t) - \hat{y}(t) \Big) \end{cases} \tag{2.13}$$

where L_{Pi} represents the gains for the proportional actions and L_{Ii} for the integral actions of the observer (2.13).

Assumption 1 : Throughout this section, the fault $f(t)$ is assumed to be constant. In other words, the derivative of $f(t)$ is zero:

$$\dot{f}(t) = 0 \tag{2.14}$$

The state estimation error $e_x(t)$ and fault estimation error $e_f(t)$ are defined as:

$$\begin{cases} e_x(t) = x(t) - \hat{x}(t) \\ e_f(t) = f(t) - \hat{f}(t) \end{cases} \tag{2.15}$$

The output error between the faulty system (3.1) and the observer (2.13) is given by:

$$y(t) - \hat{y}(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(C_i e_x(t) + D_i e_f(t)\right) \tag{2.16}$$

Using equations (3.1) and (2.13), and considering Assumption 1, the dynamics of the state estimation error and fault estimation error are given by:

$$\begin{cases} \dot{e}_x(t) = \sum_{i=1}^{r} \sum_{j=1}^{r} \mu_i(x(t))\mu_j(x(t)) \left((A_i - L_{Pi}C_j)e_x(t) + (B_i - L_{Pi}D_j)e_f(t)\right) \\ \dot{e}_f(t) = \sum_{i=1}^{r} \sum_{j=1}^{r} \mu_i(x(t))\mu_j(x(t)) \left(-L_{Ti}C_j e_x(t) - L_{Ti}D_j e_f(t)\right) \end{cases} \tag{2.17}$$

The closed-loop system is obtained by applying control law (2.12) to the model (3.1). It is expressed as follows:

$$\dot{x}(t) = \sum_{i=1}^{r} \sum_{j=1}^{r} \mu_i(x(t))\mu_j(x(t)) \left(G_{ij}x(t) + B_i K_j e_x(t) + B_i e_f(t)\right) \tag{2.18}$$

With:

$$G_{ij} = A_i - B_i K_j \tag{2.19}$$

From the dynamics (2.17), we introduce the following augmented error dynamics:

$$\dot{e}(t) = \sum_{i=1}^{r} \sum_{j=1}^{r} \mu_i(x(t))\mu_j(z(t)) F_{ij} \bar{e}(t) \tag{2.20}$$

where:

$$F_{ij} = \left(\bar{A}_i - \bar{L}_i \bar{C}_j\right) \tag{2.21}$$

and:

$$\bar{A}_i = \begin{bmatrix} A_i & B_i \\ 0 & 0 \end{bmatrix}, \quad \bar{L}_i = \begin{bmatrix} L_{pi} \\ L_{li} \end{bmatrix}, \quad \bar{C}_j = \begin{bmatrix} C_j & D_j \end{bmatrix}, \quad \bar{e}(t) = \begin{bmatrix} e_x(t) \\ e_f(t) \end{bmatrix} \tag{2.22}$$

The closed-loop system (2.18) is also rewritten in terms of the augmented estimation error $\bar{e}(t)$. Thus, the system (2.18) becomes:

$$\dot{x}(t) = \sum_{i=1}^{r}\sum_{j=1}^{r} \mu_i(x(t))\mu_j(x(t)) \left(G_{ij}x(t) + B_i\bar{K}_j\bar{e}(t)\right) \tag{2.23}$$

with:

$$\bar{K}_j = \begin{bmatrix} K_j & I \end{bmatrix} \tag{2.24}$$

The augmented system can be written by concatenating the augmented estimation error $\bar{e}(t)$ and the state $x(t)$, as follows:

$$\tilde{x}(t) = \begin{bmatrix} x(t) \\ \bar{e}(t) \end{bmatrix} \tag{2.25}$$

The dynamics of the augmented system are expressed as:

$$\dot{\tilde{x}}(t) = \sum_{i=1}^{r}\sum_{j=1}^{r} \mu_i(x(t))\mu_j(x(t))\tilde{G}_{ij}\tilde{x}(t) \tag{2.26}$$

where:

$$\tilde{G}_{ij} = \begin{bmatrix} G_{ij} & B_i\bar{K}_j \\ 0 & F_{ij} \end{bmatrix}$$

In this section, the main objective is to calculate the gains K_i and $\bar{L}_i$, ensuring the asymptotic stability of the augmented system (2.26) while guaranteeing fault compensation for the actuators. The stability analysis of system (2.26) can be ensured using the separation principle.

2.3.1.1 Separation of Observer and Controller Design:

According to the classification proposed in [23], [7] and many other articles, when the same measurable premise variable is used for both the T-S fuzzy model and the fuzzy observer, the controller and observer design procedures can be carried out independently. This configuration corresponds to the case where the scheduling variable used in the fuzzy rules, is directly accessible through sensors (e.g., input or output variables).

Under these conditions, the estimation error dynamics become independent of the system states, and the stability of the augmented system (plant + observer) can be established using separate LMI conditions for the observer and the controller. Consequently, the observer gains L_i and the controller gains K_i can be computed separately without coupling constraints, which significantly simplifies the design process.

This property of decoupling, known as the separation principle, does not necessarily hold when the premise variables differ in nature, in which case the joint design must consider their interdependence.

2.3.1.2 Observer's poles placement

In the design of observers, we must take into account the observer's dynamics relative to the system dynamics to ensure that the observer's dynamics are faster than those of the system [24]. To satisfy this condition, we will carefully place the observer poles.

For this purpose, we consider a region of the complex plane bounded by the vertical line with the abscissa $-\alpha$, where $\alpha > 0$. To impose the desired pole placement, we must ensure that the following LMI condition is satisfied:

$$(A_i - L_i C_j)^T P + P(A_i - L_i C_j) + 2\alpha P < 0 \tag{2.27}$$

2.3.1.3 Example 2.1 :

To verify the effectiveness of the proposed FTC approach, we consider the T-S model affected by actuator faults as follows:

$$\begin{cases} \dot{x}(t) = \sum_{i=1}^{2} \mu_i(x(t)) \left(A_i x(t) + B_i \left(u(t) + f(t)\right)\right) \\ y(t) = \sum_{i=1}^{2} \mu_i(x(t)) C_i x(t) \end{cases} \tag{2.28}$$

with:

$$A_1 = \begin{bmatrix} -2 & 1 & 1 \\ 1 & 3 & 0 \\ 2 & 1 & -8 \end{bmatrix}, \quad A_2 = \begin{bmatrix} -3 & 2 & -2 \\ 5 & 3 & 0 \\ 1 & 2 & -4 \end{bmatrix}, \quad B_1 = B_2 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad C_1 = C_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

By placing the observer poles in the left half-plane and setting $\alpha = 2$ to ensure faster convergence of the observation error, the state vector is initialized as [1 2 3] and by performing the calculation using the separation principle, we obtain the gains as follows: By injecting a constant fault (unit step):

$$f_1 = \begin{cases} 1, & 5 < t < 10 \\ 0, & \text{otherwise} \end{cases} \quad f_2 = \begin{cases} -1, & 15 < t < 20 \\ 0, & \text{otherwise} \end{cases}$$

The activation functions are defined as:

$$\begin{cases} \mu_1(x) = \frac{1-\tanh(x_1)}{2} \\ \mu_2(x) = 1 - \mu_1(x) \end{cases}$$

The gains are as follow :

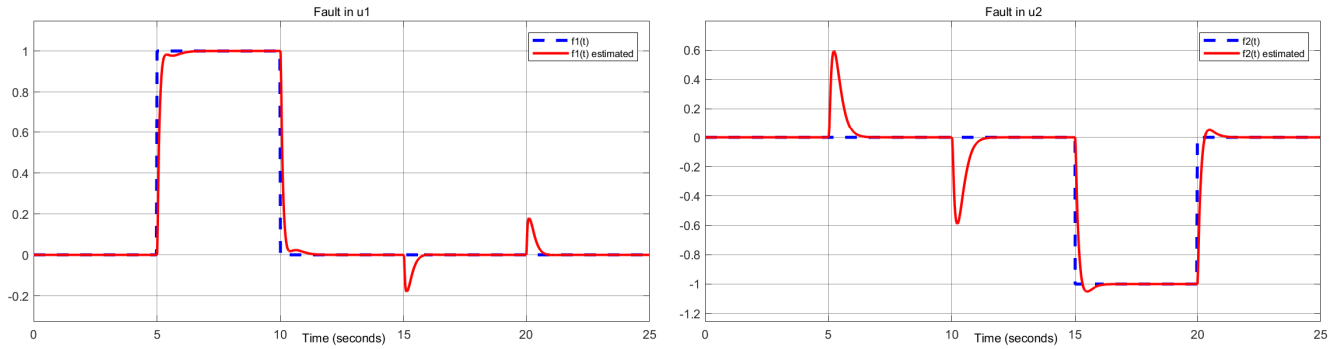
$$K_1 = \begin{bmatrix} 3.8772 & 2.7234 & 5.7092 \\ 12.0129 & -6.7592 & -7.4140 \end{bmatrix} \quad K_2 = \begin{bmatrix} 15.5072 & 1.1059 & 42.7296 \\ 32.4814 & -47.1381 & -2.8722 \end{bmatrix}$$

$$\bar{L}_1 = \begin{bmatrix} -43.3569 & -0.7811 \\ 81.7958 & -45.7333 \\ -23.8117 & 78.7569 \\ 574.4190 & -178.4007 \\ -15.0153 & 390.2877 \end{bmatrix} \quad \bar{L}_2 = 10^3 \times \begin{bmatrix} -0.0720 & 0.1709 \\ -0.0308 & -0.1157 \\ 0.3036 & -0.3357 \\ 0.3185 & -1.4543 \\ 1.6045 & -1.9478 \end{bmatrix}$$

Result analysis :

The dynamics of the observation error in figure (2.5) can be adjusted either for state estimation or fault estimation by appropriately placing the poles in specific regions. This approach is particularly useful for high-frequency faults; however, it involves trade-offs, especially regarding overshoot.

It is observed that fault estimation is influenced by other input disturbances as shown in figure (2.4) . Additionally, the estimation error rapidly converges to zero, demonstrating the effectiveness of the PI observer in estimating constant faults.



(a) f_1 *and* $\hat{f}_1$ (b) f_2 *and* $\hat{f}_2$

Figure 2.4: Faults estimation for PI observer

By initializing the estimated state vector with values close to the actual ones, we obtain the following estimation error.

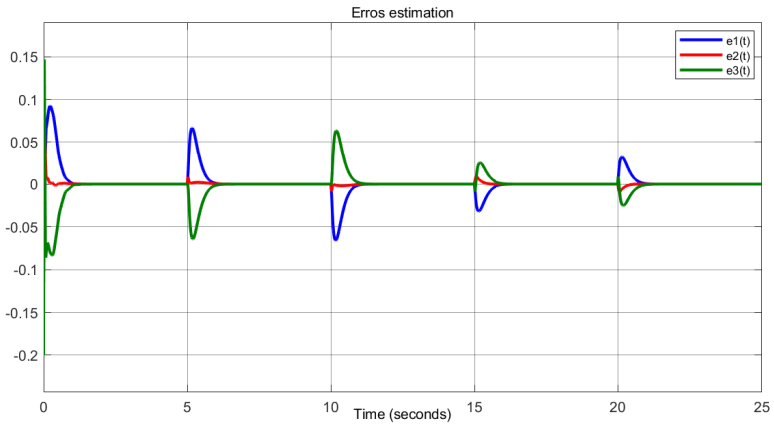


Figure 2.5: Estimation error for PI observer

While the PI observer is effective for constant faults, its performance may need to be adapted for time-varying faults using more advanced observer designs (e.g., adaptive or sliding mode observers).

2.3.2 PMI observer

The structure of the PMI observer for estimating states and faults affecting the system (3.1) is given by [22]:

$$\begin{cases} \dot{\hat{x}}(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(A_i \hat{x}(t) + B_i \left(u(t) + \hat{f}(t) \right) + L_{Pi} \left(y(t) - \hat{y}(t) \right) \right) \\ \hat{y}(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(C_i \hat{x}(t) + D_i \left(u(t) + \hat{f}(t) \right) \right) \\ \dot{\hat{f}}_1(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(\hat{f}_1(t) + L_{1i} \left(y(t) - \hat{y}(t) \right) \right) \\ \dot{\hat{f}}_j(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(\hat{f}_{j+1}(t) + L_{ji} \left(y(t) - \hat{y}(t) \right) \right), \quad j = 1, \ldots, q-1 \end{cases} \tag{2.29}$$

The proportional and integral gains of the observer are represented as L_{Pi} for the proportional terms, and L_{Ii} and L_{Ii}^{j} for the integral terms of the observer 2.29.

Hypothesis 2: Throughout this section, the fault $f(t)$ is assumed to be time-varying, and its q-th derivative is zero:

$$f^{(q)}(t) = 0 \tag{2.30}$$

Considering $f^{(q)}(t) = f_q(t)$, the successive derivatives of $f(t)$ can be expressed in state-space form as follows:

$$\begin{bmatrix} \dot{f}(t) \\ \dot{f}_1(t) \\ \vdots \\ \dot{f}_{q-1}(t) \end{bmatrix} = \begin{bmatrix} f_1(t) \\ f_2(t) \\ \vdots \\ f_q(t) \end{bmatrix} \tag{2.31}$$

Figure 2.6 illustrates the structure of the observer. It shows that the estimation of unknown inputs involves estimating its first $q-1$ derivatives through $q-1$ integral actions [21], hence the name Multi-Integral.

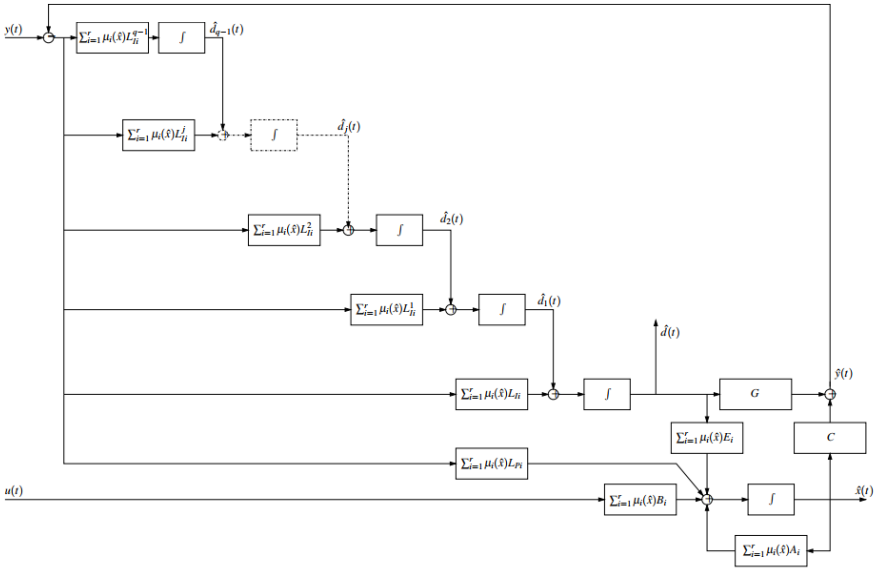


Figure 2.6: Principle of the PMI observer

From equation (2.31), considering equation (2.30), the system (3.1) can be represented in the following augmented form:

$$\begin{cases} \dot{\bar{x}}(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(\bar{A}_i \bar{x}(t) + \bar{B}_i u(t) \right) \\ y(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(\bar{C}_i \bar{x}(t) + D_i u(t) \right) \end{cases} \tag{2.32}$$

where:

$$\bar{x}(t) = \begin{bmatrix} x(t) \\ f(t) \\ f_1(t) \\ \vdots \\ f_{q-1}(t) \end{bmatrix}, \quad \bar{A}_i = \begin{bmatrix} A_i & B_i & 0 & \cdots & 0 \\ 0 & 0 & I_{n_f} & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & I_{n_f} \end{bmatrix}, \quad \bar{B}_i = \begin{bmatrix} B_i \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \bar{C}_i = \begin{bmatrix} C_i & D_i & 0 & \cdots & 0 \end{bmatrix}$$

The PMI observer (equation 2.29) can also be expressed in the augmented form as:

$$\begin{cases} \dot{\hat{\bar{x}}}(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(\bar{A}_i \hat{\bar{x}}(t) + \bar{B}_i u(t) + \bar{L}_i \left(y(t) - \hat{y}(t) \right) \right) \\ \hat{y}(t) = \sum_{i=1}^{r} \mu_i(x(t)) \left(\bar{C}_i \hat{\bar{x}}(t) + D_i u(t) \right) \end{cases} \tag{2.33}$$

With :

$$\hat{\bar{X}}(t) = \begin{bmatrix} \hat{x}(t) \\ \hat{f}(t) \\ \hat{f}_1(t) \\ \vdots \\ \hat{f}_{q-1}(t) \end{bmatrix} \quad , \quad \bar{L}_i = \begin{bmatrix} L_{Pi} \\ L_{1i} \\ L_{1i}^1 \\ \vdots \\ L_{1i}^{q-2} \\ L_{1i}^{q-1} \end{bmatrix}$$

The augmented estimation error is defined as:

$$\bar{e}(t) = \bar{x}(t) - \hat{\bar{x}}(t) \tag{2.34}$$

The output error between the system (2.32) and the observer (2.33) is given by:

$$y(t) - \hat{y}(t) = \sum_{i=1}^{r} \mu_i(x(t)) \bar{C}_i \bar{e}(t) \tag{2.35}$$

Using equations (2.32), (2.33), and (2.34), the dynamics of the augmented estimation error can be expressed as:

$$\dot{\bar{e}}(t) = \sum_{i=1}^{r} \sum_{j=1}^{r} \mu_i(x(t)) \mu_j(x(t)) F_{ij} \bar{e}(t) \tag{2.36}$$

where:

$$F_{ij} = \left(\bar{A}_i - \bar{L}_i \bar{C}_j\right)$$

The closed-loop system is obtained by applying control law (2.12) to the model (3.1), which is written as:

$$\dot{x}(t) = \sum_{i=1}^{r}\sum_{j=1}^{r} \mu_i(x(t))\mu_j(x(t)) \left(G_{ij}x(t) + B_i K_j \bar{e}(t)\right) \tag{2.37}$$

From equation (2.36) and equation (2.37), the concatenation of the augmented estimation error $\bar{e}(t)$ and the state $x(t)$ allows us to write the following augmented system:

$$\dot{\bar{x}}(t) = \sum_{i=1}^{r}\sum_{j=1}^{r} \mu_i(x(t))\mu_j(x(t))\bar{G}_{ij}\bar{x}(t) \tag{2.38}$$

where:

$$\bar{x}(t) = \begin{bmatrix} x(t) \\ \bar{e}(t) \end{bmatrix}, \quad \bar{G}_{ij} = \begin{bmatrix} G_{ij} & B_i\bar{K}_j \\ 0 & F_j \end{bmatrix}$$

The observer used in this section is the PMI observer, which is designed to simultaneously estimate states and faults. This observer is adapted for estimating polynomial signals with a null q-th derivative. The fault-tolerant control design involves calculating the gains K_i and $\bar{L}_i$ to ensure the asymptotic stability of the augmented system (2.38) while guaranteeing fault compensation for the actuators.

Remark : In the case where the disturbance has an infinite-order derivative, such as an exponential or sinusoidal function, the observer's order q is determined experimentally by testing at which order q the disturbance is compensated or its effect no longer appears in the system (i.e., the disturbance is well observed). As indicated in the examples discussed in [22].

2.3.3 Example 2.2

Let us consider same Example (2.1), which we have analyzed in the case of a PI observer, by placing the observer poles in the left half-plane and setting $\alpha = 1.5$, and by injecting the following fault, :

$$f_1 = \begin{cases} t, & 2 < t < 6 \\ 0, & \text{otherwise} \end{cases}$$

$$f_2 = \begin{cases} -t, & 12 < t < 16 \\ 0, & \text{otherwise} \end{cases}$$

We get :

$$\bar{L}_1 = 1.0 \times 10^4 \times \begin{bmatrix} 0.0370 & -0.1166 \\ 0.0097 & -0.0548 \\ 0.0117 & 0.0771 \\ 0.0784 & -0.3336 \\ 0.1249 & 0.3782 \\ 1.0555 & -3.3982 \\ 1.0393 & -2.1414 \end{bmatrix} \; ; \; \bar{L}_2 = 1.0 \times 10^5 \times \begin{bmatrix} -0.0088 & 0.0924 \\ 0.0057 & 0.0409 \\ -0.0432 & -0.0453 \\ 0.0139 & 0.2520 \\ -0.3004 & -0.1896 \\ -0.2263 & 2.6867 \\ -0.6474 & 1.8632 \end{bmatrix}$$

In this case, where there are two disturbances,

$$\hat{\bar{X}}(t) = \begin{bmatrix} \hat{x}(t) \\ \hat{f}1(t) \\ \hat{f}2(t) \\ \dot{\hat{f}}1(t) \\ \dot{\hat{f}}2(t) \end{bmatrix} \quad , \quad \bar{L}_i = \begin{bmatrix} L_{Pi} \\ L1_{1i} \\ L2_{1i} \\ L1_{1i}^1 \\ L2_{1i}^1 \end{bmatrix}$$

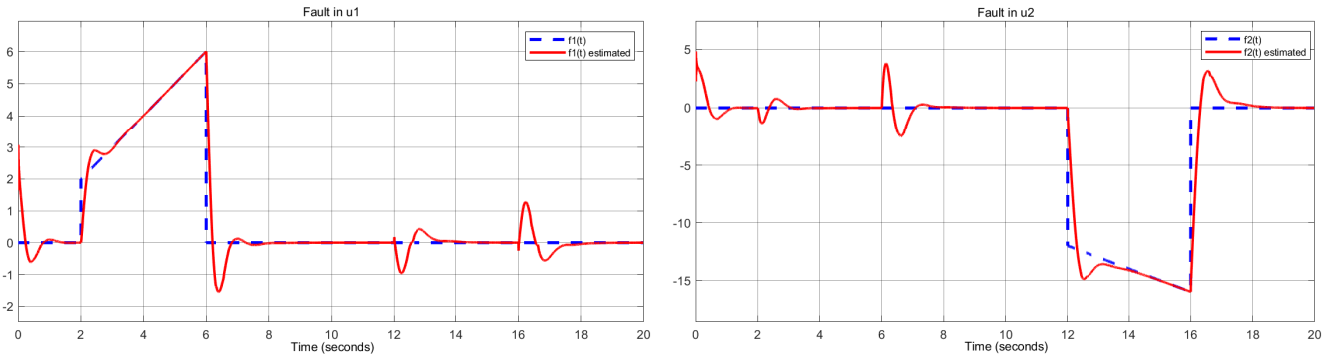
where $\hat{f}1$ and $\hat{f}2$ represent the estimated errors, and $\dot{\hat{f}}1$ and $\dot{\hat{f}}2$ denote their successive derivatives.
L_1 represents the estimation gains for the first disturbance and its derivatives. L_2 represents the estimation gains for the second disturbance and its derivatives.

Result analysis

It can be observed that, in the case of the PMI observer, the estimation error is slightly larger compared to the PI observer, but it converges directly to zero.

The PMI observer is a generalization of the PI observer, where the successive integration of the estimation error helps in further reducing it as we can see in figure (2.8) . By increasing the order and complexity of the fault model, the observer can still achieve good results.

The same remark as in the PI observer we can see in the figure (2.7 that the fault can influence on the other input fault estimation.



(a) f_1 *and* $\hat{f}_1$ (b) f_2 *and* $\hat{f}_2$

Figure 2.7: Faults estimation for PMI observer

FTC Control Approach with PMI Observer it's an approach which effectively compensates for such faults, leading to significantly improved system performance.

By initializing the estimated state vector with values close to the actual ones, we obtain the following estimation error.

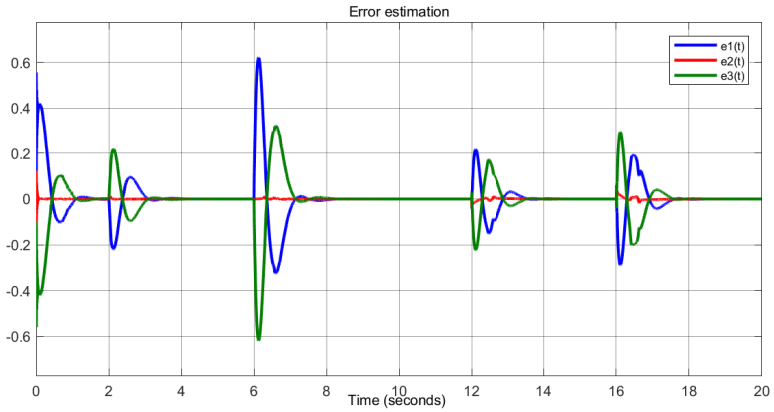


Figure 2.8: State estimation error for PMI observer

Conclusion

In this chapter, we investigated fault-tolerant control strategies for nonlinear systems represented using Takagi-Sugeno models. We began by reviewing general fault-tolerant control concepts, distinguishing between passive and active approaches, and highlighting the relevance of observer-based techniques in modern control applications.

Particular emphasis was placed on T-S observers under measurable premise variables MPV, covering both classical Luenberger-type observers and advanced structures such as Proportional-Integral PI and PMI observers. The mathematical formulation of these observers was presented, including the use of LMI for ensuring stability and convergence of estimation errors. Through these designs, we addressed both constant and time-varying actuator faults.

Overall, the results confirm the effectiveness of the designed fault-tolerant controllers and their observers in maintaining system stability and ensuring robust tracking performance. These strategies offer practical applicability in critical control systems requiring real-time fault accommodation while maintaining acceptable computational cost.

Chapter 3

Iterative Observer Design : Single and Multi-Agent systems

Introduction

The reliability and safety of modern dynamic systems are increasingly threatened by unexpected faults, especially in complex and interconnected environments. Accurate fault estimation is therefore a key component of fault-tolerant control strategies. While classical observers such as Luenberger and Kalman filters have long been used for fault detection and state estimation, their single-step nature and sensitivity to disturbances and modeling errors often limit their performance in practice.

To overcome these challenges, this chapter investigates an advanced iterative fault estimation method, known as the *k-step fault estimation*, which successively refines the fault estimate at each step using previous estimation errors. The approach is particularly well-suited for systems modeled using the Takagi–Sugeno fuzzy formalism, which provides a powerful framework for representing nonlinear dynamics through a convex combination of linear models.

We begin by applying the k-step fault estimation technique to a single-agent T-S system, establishing the mathematical formulation, observer design, and stability analysis using an H_∞ performance framework. The method is then extended to a multi-agent setting, where agents are connected through a communication topology. In this context, we explore distributed iterative observers that incorporate both local and consensus-based estimation errors, and an output-feedback fault-tolerant control law.

Simulation results demonstrate the effectiveness of the proposed method in reconstructing actuator faults and maintaining consensus across the agent network. This chapter lays the theoretical and practical groundwork for robust and scalable fault-tolerant architectures that will be further explored in the next chapters.

3.1 Iterative Fault Estimation of a Single-Agent T-S System

3.1.1 Iterative and classical estimation

Iterative estimation is an advanced method that significantly improves the accuracy of fault and state estimation in dynamic systems by refining the estimation process over multiple iterations [25, 26]. This approach reuses previous estimation errors to reduce uncertainties and enhance robustness, especially when applied to nonlinear systems modeled through the T-S framework. Unlike single-step methods, which perform a one-time correction, iterative observers continually adjust their outputs, enabling better handling of time-varying faults and external disturbances.

Classical observers, such as the Luenberger observer and Kalman filter, although widely used, face several limitations [27]. They often assume accurate modeling and specific noise characteristics, which makes them sensitive to real-world uncertainties. Moreover, they require complex gain tuning and struggle to track rapidly changing faults, while robust methods like sliding mode observers suffer from chattering effects. In contrast, iterative observers offer greater adaptability to nonlinear and uncertain environments, making them a more suitable choice for T-S systems that demand resilience against modeling errors and external perturbations [25].

3.1.2 H_∞ Performance Index

The objective of H_∞ control is to minimize the maximum gain between a disturbance $d(t)$ and an error $e(t)$ defined as [28]:

$$\sup_{d\neq 0} \frac{\|e\|_2}{\|d\|_2} < \lambda$$

where:

- $\|e\|_2$ is the L_2 norm of the error (measuring its total energy),
- $\|d\|_2$ is the L_2 norm of the disturbance,
- λ is a tolerance threshold that defines the maximum level of disturbance amplification.

$$\int_0^t e^T(s)e(s)ds < \lambda^2 \int_0^t d^T(s)d(s)ds.$$

This condition guarantees that the system does not overreact to disturbances and that the estimation error remains controlled under external perturbations. By imposing this constraint, the system maintains a robust estimation performance, ensuring that the disturbances do not excessively influence the estimated fault.

3.1.3 Stability Analysis of Iterative Fault Estimation

By selecting an appropriate Lyapunov function defined as:

$$V = e^T P e$$

where e is the estimation error vector that includes both the state estimation error $e_x = x - \hat{x}$ and the fault estimation error $e_f = f - \hat{f}$ at a given iteration, and where $P = P^T > 0$ is a symmetric positive definite matrix, we can assess the stability of the iterative observer [29]. If the H_∞ performance condition is satisfied, then the quadratic stability of the iterative observer in the sense of Lyapunov is ensured.

This framework offers flexibility in choosing the observer dynamics and the control design technique. Whether the observer is implemented via static or dynamic gains, and regardless of the control law [25] (e.g., output feedback, or PDC control), the stability remains guaranteed as long as the Lyapunov condition is fulfilled. Additionally, this formulation allows the incorporation of robustness criteria such as attenuation of disturbances, which is crucial for fault estimation in uncertain or nonlinear systems.

3.1.4 Single Agent iterative observer design

In this section, we present the design of an iterative observer intended to estimate both the system states and additive actuator faults with improved accuracy, it refines the fault estimation over successive steps by incorporating previous estimation errors into the current correction process.

3.1.4.1 One-Step Fault Estimation Approach

Let define the system dynamic model in the situation of disturbance and actuator fault of the following form :

$$\begin{cases} \dot{x}(t) &= A(\xi)x(t) + B(\xi)u(t) + F(\xi)f(t) + G(\xi)d(t) \\ y(t) &= C(\xi)x(t) \end{cases} \tag{3.1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector of the system, $f(t) \in \mathbb{R}^r$ stands for the additive mismatched actuator fault, $u(t) \in \mathbb{R}^m$ stands for the input, $d(t) \in \mathbb{R}^d$ stands for the bounded disturbance, and $y(t) \in \mathbb{R}^p$ represents the output vector. $A(\xi)$, $B(\xi)$, $F(\xi)$, $G(\xi)$, and $C(\xi)$ are described as follows :

$$A(\xi) = \sum_{i=1}^{r} \mu_i(\xi) A_i, \quad B(\xi) = \sum_{i=1}^{r} \mu_i(\xi) B_i,$$
$$C(\xi) = \sum_{i=1}^{r} \mu_i(\xi) C_i, \quad F(\xi) = \sum_{i=1}^{r} \mu_i(\xi) F_i,$$
$$G(\xi) = \sum_{i=1}^{r} \mu_i(\xi) G_i$$

The activation functions $\mu_i(\xi)$ satisfy:

$$\sum_{i=1}^{r} \mu_i(\xi) = 1, \quad \mu_i(\xi) \geq 0, \quad \forall i. \tag{3.2}$$

Such that r is the number of rules of the T-S system, and A,B,C,G and F represent the identified constant real parameter matrices of the system with appropriate dimensions.

Assumption 1. *The system (3.1) is observable and stabilizable.*

Assumption 2. *rank*$(B(\xi), F(\xi)) =$ *rank*$(B(\xi))$.

Assumption (2) indicates that there exists a matrix $B^*(\xi) \in \mathbb{R}^{n\times m}$ which makes the following equation hold:

$$(I - B(\xi)B^*(\xi))F(\xi) = 0 \tag{3.3}$$

Assumption 3. $f, d \in L_2[0, \infty)$

A function $f(t)$ belongs to $L_2[0, \infty)$ if its energy is finite [30], which is expressed as:

$$\int_0^\infty |f(t)|^2 dt < \infty$$

This condition ensures that $f(t)$ does not grow indefinitely and has a finite energy, it guarantees that the fault $f(t)$ remains within a limited energy range, making it possible to estimate and compensate for it.

The state and the unknown mismatched additive actuator fault of the system (3.1) are estimated by the k-step FE algorithm.
The one-step FE observer is constructed as [29] :

$$\begin{cases} \dot{\hat{x}}_1 & = A(\xi)\hat{x}_1 + B(\xi)u + F(\xi)\hat{f}_1 + L_s(\xi)(y - \hat{y}_1) \\ \hat{y}_1 & = C(\xi)\hat{x}_1 \\ \dot{\hat{f}}_1 & = L_f(\xi)(y - \hat{y}_1) \end{cases} \tag{3.4}$$

where $\hat{f}_1 \in \mathbb{R}^r$, $\hat{y}_1 \in \mathbb{R}^p$, $\hat{x}_1 \in \mathbb{R}^n$, are the one-step estimations of f, y, and x, respectively. L_f and L_s stand for the FE observer gain matrices.

The estimate error is constructed as $e_{x_1} = x - \hat{x}_1$, $e_{f_1} = f - \hat{f}_1$, $e_{y_1} = y - \hat{y}_1$, respectively, and the augmented vector is denoted as:

$$e_1^T = [e_{x_1}^T, e_{f_1}^T].$$

For simplicity, the time index t is omitted. Then, combining (3.1) and (3.4), the error dynamic system of the first augmented estimation can be constructed as follows:

$$\begin{cases} \dot{e}_1 & = (\bar{A}(\xi) - L(\xi)\bar{C}(\xi))e_1 + \bar{G}(\xi)d_1, \\ e_{y_1} & = \bar{C}(\xi)e_1. \end{cases} \tag{3.5}$$

where

$$\bar{A}(\xi) = \begin{pmatrix} A(\xi) & F(\xi) \\ 0 & 0 \end{pmatrix}, \quad L(\xi) = \begin{pmatrix} L_s(\xi) \\ L_f(\xi) \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{r} \mu_i(\xi) L_{si} \\ \sum_{i=1}^{r} \mu_i(\xi) L_{fi} \end{pmatrix}$$

$$\bar{C}(\xi) = \begin{pmatrix} C(\xi) & 0 \end{pmatrix}, \quad \bar{G}(\xi) = \begin{pmatrix} G(\xi) & 0 \\ 0 & I \end{pmatrix},$$

$$d_1 = \begin{pmatrix} d \\ \dot{f} \end{pmatrix}.$$

3.1.4.2 Limitations of one-step estimation

One-step fault estimation methods are generally based on ideal assumptions, such as bounded disturbances and constant system parameters. However, in practical situations, these estimators are highly sensitive to model uncertainties and external disturbances. Since the estimation is performed only once [31], there is no mechanism to refine the results, which leads to accumulated estimation errors especially in the presence of rapidly varying faults. The inability to correct these errors after the initial step makes the estimator less effective in dynamic environments.

Moreover, one-step approaches often rely on fixed observer gains, limiting their adaptability to time-varying systems [25]. Without a feedback or update mechanism, they fail to track fast-changing faults accurately and tend to offer lower robustness against noise and perturbations. This results in poor estimation performance and limited reliability in applications where high accuracy is essential.

3.1.4.3 Two step Fault estimation

Then the two-step FE observer can be constructed below [29]:

$$\begin{cases} \dot{\hat{x}}_2 &= A(\xi)\hat{x}_2 + B(\xi)u + F(\xi)\hat{f}_2 + L_s(\xi)(y - \hat{y}_2) \\ \hat{y}_2 &= C(\xi)\hat{x}_2 \\ \dot{\hat{f}}_2 &= L_f(\xi)(y - \hat{y}_2) + \dot{\hat{f}}_1 \end{cases} \tag{3.6}$$

where $\hat{x}_2 \in \mathbb{R}^n$, $\hat{y}_2 \in \mathbb{R}^p$, $\hat{f}_2 \in \mathbb{R}^r$ are the two-step estimations of x, y, and f, respectively. L_s and L_f are the same as those in equation (3.4).

Similarly, denote the estimation errors as:

$$e_{x_2} = x - \hat{x}_2, \quad e_{f_2} = f - \hat{f}_2, \quad e_{y_2} = y - \hat{y}_2$$

and define the augmented error vector as:

$$e_2^T = [e_{x_2}^T, e_{f_2}^T].$$

Then the second augmented error dynamic system is similar to (3.5):

$$\begin{cases} \dot{e}_2 &= (\bar{A}(\xi) - L_f(\xi)\bar{C}(\xi))e_2 + \bar{G}(\xi)d_2, \\ e_{y_2} &= \bar{C}(\xi)e_2. \end{cases} \tag{3.7}$$

where $\bar{A}(\xi)$, L_f, $\bar{C}(\xi)$, and $\bar{G}(\xi)$ are the same as those in equation (3.5).

It is worth noting that:

$$d_2 = \begin{pmatrix} d \\ \dot{f} - \dot{\hat{f}}_1 \end{pmatrix}.$$

3.1.4.4 The k-step Fault estimation approach

To enhance the accuracy of FE, the $(k-1)$-step FE information is applied to construct the k-step FE observer [29]:

$$\begin{cases} \dot{\hat{x}}_k &= A(\xi)\hat{x}_k + B(\xi)u + F(\xi)\dot{\hat{f}}_k + L_s(\xi)(y - \hat{y}_k) \\ \hat{y}_k &= C(\xi)\hat{x}_k \\ \dot{\hat{f}}_k &= L_f(\xi)(y - \hat{y}_k) + \dot{\hat{f}}_{k-1} \end{cases} \tag{3.8}$$

where $\hat{x}_k \in \mathbb{R}^n$, $\hat{y}_k \in \mathbb{R}^p$, $\hat{f}_k \in \mathbb{R}^r$ are the k-step estimations of x, y, and f, respectively. L_s and L_f are the same as those in equation (3.4).

Denote the estimation errors as:

$$e_{x_k} = x - \hat{x}_k, \quad e_{f_k} = f - \hat{f}_k, \quad e_{y_k} = y - \hat{y}_k,$$

and define the augmented error vector as:

$$e_k^T = [e_{x_k}^T, e_{f_k}^T].$$

The kth error dynamic equation can be obtained:

$$\begin{cases} \dot{e}_k &= (\bar{A}(\xi) - L_f(\xi)\bar{C}(\xi))e_k + \bar{G}(\xi)d_k, \\ e_{y_k} &= \bar{C}(\xi)e_k. \end{cases} \tag{3.9}$$

where $\bar{A}(\xi)$, L_f, $\bar{C}(\xi)$, and $\bar{G}(\xi)$ are the same as those in equation (3.5).

It is worth noting that:

$$d_k^T = \left(d^T, \quad \left(\dot{f} - \dot{\hat{f}}_{k-1}\right)^T\right).$$

Theorem 4. *Consider the system (3.1) with Assumptions 1 – 2. If there exist matrices P, Q, satisfying the following inequality, for the given scalar $\lambda > 0$, the error dynamic system (3.9) is stable at given H_∞ performance index:*

$$\int_0^t e_k^T(s)e_k(s)ds < \lambda^2 \int_0^t d_k^T(s)d_k(s)ds. \tag{3.10}$$

The condition to guarantee this stability is:

$$\begin{aligned} \Sigma_{ii} &< 0 \quad \textit{for } i \in \{1, 2, \ldots, r\} \\ \Sigma_{ij} + \Sigma_{ji} &< 0 \quad \textit{for } i < j \leq r \end{aligned} \tag{3.11}$$

$$\Sigma_{ij} = \begin{pmatrix} \bar{A}_i^T P + P\bar{A}_i - Q_i\bar{C}_j - (Q_i\bar{C}_j)^T + I & P\bar{G}_i \\ * & -\lambda^2 I \end{pmatrix}$$

where $P = P^T > 0$. The gain matrices of the observer (3.8) can be derived from:

$$\bar{L}_i = P^{-1}Q_i$$

Proof :
The Lyapunov function can be considered as:

$$V_k = e_k^T P e_k \tag{3.12}$$

Along the estimation error dynamic system (3.9), the time derivative of V_k is derived below:

$$\begin{aligned} \dot{V}_k = e_k^T \left[P(\bar{A}_i - L_{fi}\bar{C}_j) + (\bar{A}_i - L_{fi}\bar{C}_j)^T P\right] e_k \\ + 2e_k^T P\bar{G}_i d_k \end{aligned} \tag{3.13}$$

Denote the auxiliary function as

$$J = \dot{V}_k + e_k^T e_k - \lambda^2 d_k^T d_k \tag{3.14}$$

Substituting (3.13) into (3.14), it can be obtained that

$$\begin{aligned} J = e_k^T \left[P(\bar{A}_i - L_{fi}\bar{C}_j) + (\bar{A}_i - L_{fi}\bar{C}_j)^T P + I\right] e_k \\ + 2e_k^T P\bar{G}_i d_k - \lambda^2 d_k^T d_k \end{aligned} \tag{3.15}$$

Denote $e^T = (e_k^T, d_k^T)$. If inequality (4) holds, it can be obtained that $e^T \Sigma e < 0$ with $\bar{L} = P^{-1}Q$, namely

$$J = \dot{V}_k + e_k^T e_k - \lambda^2 d_k^T d_k < 0 \tag{3.16}$$

Furthermore, the integral of J is shown as follows:

$$\int_0^t \left[\dot{V}_k(s) + e_k^T(s)e_k(s) - \lambda^2 d_k^T(s)d_k(s)\right] ds < 0 \tag{3.17}$$

It can be formulated that

$$\int_0^t \left[\dot{V}_k(s) + e_k^T(s)e_k(s) - \lambda^2 d_k^T(s)d_k(s)\right] ds = V(t) + \int_0^t \left[e_k^T(s)e_k(s) - \lambda^2 d_k^T(s)d_k(s)\right] ds < 0 \tag{3.18}$$

It can be immediately obtained that

$$\int_0^t \left[e_k^T(s)e_k(s) - \lambda^2 d_k^T(s)d_k(s)\right] ds < 0. \tag{3.19}$$

Therefore, the actuator fault and the system state can be evaluated by the developed FE observer (3.8).

To make the above algorithm clearer, the algorithm of the fault estimation method is given in the following.

Algorithm 1. *k-step fault estimation algorithm.*

1. *Obtain* $\hat{f}_1(t)$ *by solving (3.4).*
2. *Let* $k = 2$ *and calculate* $\hat{f}_k(t)$ *from (3.8).*
   Calculate
   $$\epsilon = \|\hat{f}_k(t) - \hat{f}_{k-1}(t)\|$$
   If ϵ *is not less than a given small enough constant, then* $k = k + 1$*, go to step 2, else*
   $$\hat{f}(t) = \hat{f}_{k+1}(t)$$
   can be taken as the estimation of $f(t)$*.*

3.1.4.5 Example 2.1

Let the nonlinear system be represented by a Takagi-Sugeno fuzzy multi-model, defined as follows [29]:

$$\begin{cases} \dot{x}(t) = \sum_{i=1}^{2} \mu_i(z(t)) \left(A_i x(t) + B_i u(t) + F_i f(t) + G_i d(t)\right) \\ y(t) = \sum_{i=1}^{2} \mu_i(z(t))\, C_i\, x(t) \end{cases}$$

Such that :

$$A_1 = \begin{bmatrix} -0.5 & -0.5 \\ 0 & -0.3 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 0.2 & 0 \\ 0 & -0.3 \end{bmatrix}, \quad C_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$F_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad G_1 = \begin{bmatrix} 0.2 \\ 0.1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} -0.6 & -0.4 \\ -0.4 & -0.25 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0.1 & 0 \\ 0 & -0.4 \end{bmatrix}, \quad C_2 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$F_2 = \begin{bmatrix} 0.8 \\ 1.5 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 0.15 \\ 0.2 \end{bmatrix}$$

The weighting functions $\mu_i(\xi)$ are defines as :

$$\mu_1(x) = \frac{1}{1 + e^{-3(x_1 - 2)}}$$

$$\mu_2(x) = 1 - \mu_1(x)$$

By applying theorem (4) to the system and we take $\lambda = 0.08$ to fix the h_∞ performance index , we derive the observer gains:

$$L_{k1} = \begin{bmatrix} -20.3282 & -51.0913 \\ -19.3742 & -47.4633 \end{bmatrix}, \quad L_{f1} = \begin{bmatrix} -210.6993 & -385.7618 \end{bmatrix}$$

$$L_{k2} = \begin{bmatrix} -25.6183 & -50.2128 \\ -24.3006 & -46.7038 \end{bmatrix}, \quad L_{f2} = \begin{bmatrix} -257.0665 & -399.4831 \end{bmatrix}$$

To stabilize the system, we implement a simple PDC feedback control based on theorem (2) the matrix gain is as follow :

$$K_1 = \begin{bmatrix} -0.4125 & -6.1651 \\ -2.2955 & 0.0034 \end{bmatrix} \quad ; \quad K_2 = \begin{bmatrix} -2.7575 & -8.5692 \\ -0.0786 & -0.3312 \end{bmatrix}$$

By injecting a fault given by:

$$f(t) = \begin{cases} 0, & t \leq 5 \\ 2\left(1 - e^{5(5-t)}\right), & t > 5 \end{cases}$$

We set a tolerance error for fault estimation as: $\epsilon = 0.2$ -this value was chosen to avoid going too far in the number of iterations, while still allowing the estimation to converge within a reasonable number of steps. With this threshold, convergence was reached after 6 iterations, which is sufficient as illustrated in the figures- by setting the initial conditions at $x_0^T = [1 \ 2]^T$ with $d(t) = 0$, we will determine the iteration at which the estimation reaches this tolerance. Then, we will compare the fault estimation at this iteration with the one-step estimation and analyze the results.

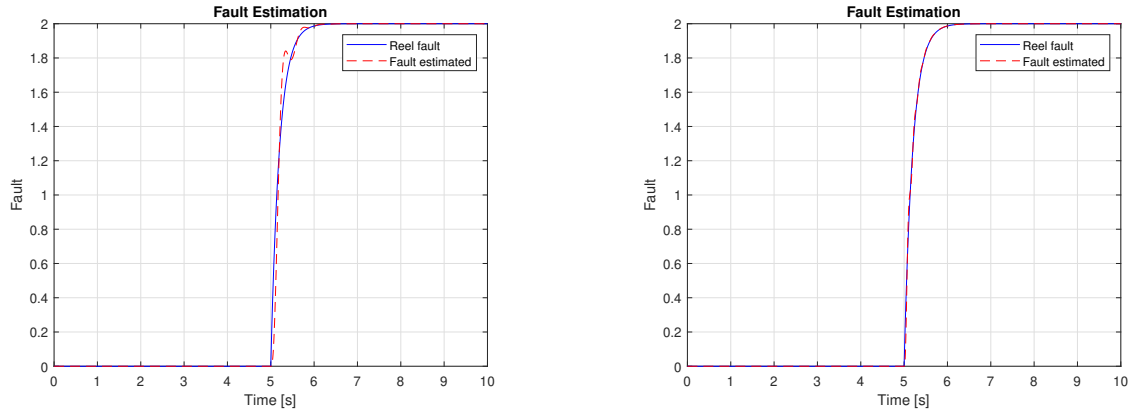
In this example, the matrix B is invertible, which allows us to proceed with the computations directly. However, in other cases, it is important to verify the invertibility condition of B before applying any operation that involves its inverse. If B is not invertible, alternative approaches such as using a **pseudo-inverse** or redesigning the observer may be required.

Simulation Results :

Iterative estimation is based on the first iteration, known as one-step fault estimation, which initially produces a small error between the actual fault and the estimated fault, as shown in Figure (3.1a). It is observed that when the estimation error reaches 0.2, convergence is achieved around the 6th iteration, as illustrated in Figure (3.1b). Given a nominal model of the system, it is possible to define an estimation tolerance, which allows for the identification of any type of fault, whether variable or constant.

The error curve, well illustrated in Figure (3.2), clearly demonstrates that at each iteration, the error is recalculated. This iterative process determines whether to continue refining the estimation or terminate the algorithm. The iterative estimation approach provides control inputs that directly depend on the fault dynamics this highlights the improvement in precision with an increased number of iterations, while still respecting Assumption 3.

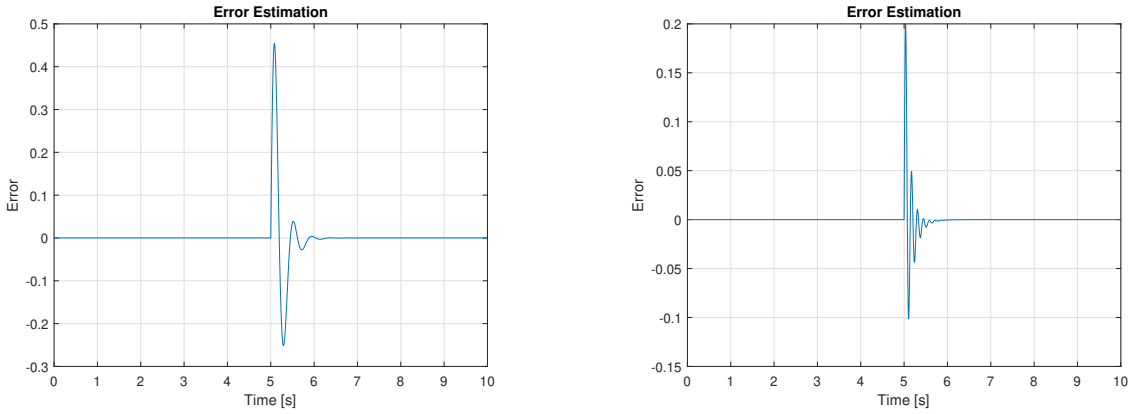
Furthermore, Figure (3.3) demonstrates that with a sufficiently high number of iterations, the state estimation is significantly improved, ensuring that the effect of the fault is properly compensated. This confirms the effectiveness of the estimation process. However, special attention must be given to the initial conditions of the estimated states at each iteration. They must be set to the same values as the actual system states to maintain estimation consistency. Therefore, a prior knowledge of the system's initial state is essential to ensure accurate and reliable estimation results.



(a) One step fault estimation

(b) 6 - step fault estimation

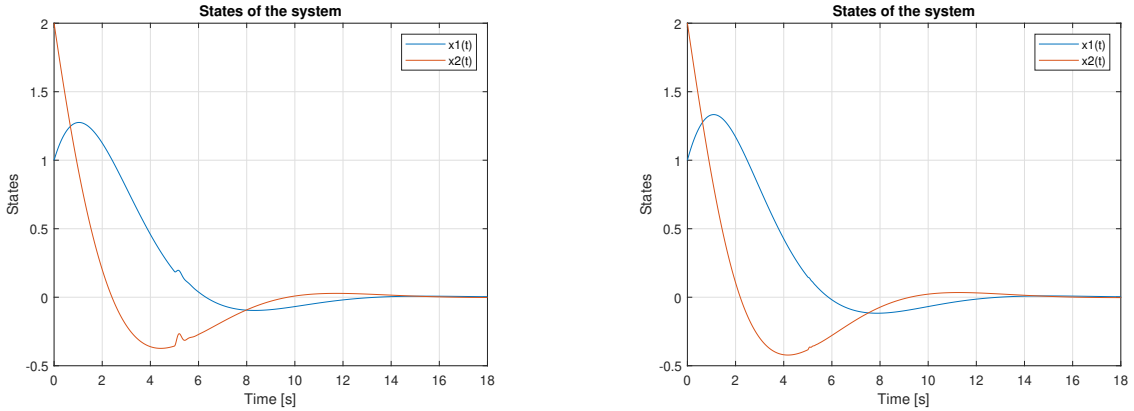
Figure 3.1: Fault Estimation



(a) One step fault estimation

(b) 6 - step fault estimation

Figure 3.2: Fault Estimation Error



(a) One step fault estimation

(b) 6 - step fault estimation

Figure 3.3: States evolution

The single-agent k-step fault estimation approach effectively improves accuracy by iteratively refining the estimation using previous error information. It ensures robustness against disturbances and modeling uncertainties through an H_∞-based stability analysis. While this method performs well in isolated systems, real-world applications often involve multiple interacting agents. Therefore, the following section extends this approach to multi-agent systems, incorporating communication topology and distributed estimation dynamics.

3.2 Iterative Fault Estimation of Multi-Agent T-S Systems

The increasing complexity and scale of modern systems demand robust fault diagnosis and control strategies. Multi-agent systems offer a decentralized framework well-suited for such environments, but a fault in a single agent can destabilize the entire network due to inter-agent interactions.

While numerous works have addressed fault detection and estimation in linear MAS using distributed observers and proportional-integral schemes [32, 33, 34], the nonlinear case remains less explored. T-S fuzzy models have proven effective for representing nonlinear dynamics, and recent efforts have extended observer design to estimate both states and faults in such systems [35, 36].

A notable approach is the k-step fault estimation method [37], which improves estimation accuracy by iteratively refining fault estimates. However, this method has primarily been applied to single-agent systems in the previous section. This work extends the k-step strategy to nonlinear MAS by combining centralized and distributed estimation errors, enabling accurate actuator fault reconstruction and supporting the design of a robust fault-tolerant controller.

3.2.1 Multi-Agent System Modeling

Consider a network composed of N nonlinear agents interacting over a directed communication topology. The dynamics of each agent $i \in \{1, 2, \ldots, N\}$ are described by the following Takagi–Sugeno fuzzy model:

$$\begin{cases} \dot{x}_i(t) = A(\xi)x_i(t) + B(\xi)u_i(t) + E(\xi)f_i(t) + D(\xi)d_i(t), \\ y_i(t) = C(\xi)x_i(t) \end{cases} \tag{3.20}$$

where $x_i(t) \in \mathbb{R}^n$ is the system state of the agent i, $y_i(t) \in \mathbb{R}^p$ is the measured output, and $u_i(t) \in \mathbb{R}^m$ is the control input. The signal $d_i(t) \in \mathbb{R}^q$ denotes the external disturbance, assumed to satisfy $d_i(t) \in L_2[0, \infty)$. The term $f_i(t) \in \mathbb{R}^r$ represents the actuator fault affecting agent i, and is also assumed to belong to $L_2[0, \infty)$. The matrices $A(\xi)$, $B(\xi)$, $C(\xi)$, $D(\xi)$, and $E(\xi)$ are fuzzy interpolations of linear matrices with appropriate dimensions, depending on the scheduling variable ξ as described in the single agent case 3.1.

3.2.2 Theoretical concepts on Multi-Agent Systems

A **multi-agent system MAS** is a collection of interacting agents that cooperate (or sometimes compete) to achieve individual or global objectives [38]. Each agent is typically an autonomous subsystem capable of sensing, decision-making, and acting within an environment.

- **Agent:** An agent refers to an independent control system (e.g., a robot, vehicle, or sensor node) capable of performing tasks and exchanging information with others.
- **Communication Graph:** The interaction topology between agents is usually modeled by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of agents (nodes), and $\mathcal{E}$ the set of communication links (edges).
- **Consensus:** A fundamental objective in MAS is consensus, which means all agents agree on certain quantities (e.g., position, velocity, or state estimates) using only local information and interactions.
- **Distributed Control:** Unlike centralized control, distributed control implies that each agent computes its control input based on its own information and that received from neighbors, leading to scalable and robust coordination.
- **Applications:** MASs are widely used in robotics (e.g., swarm robotics), sensor networks, autonomous vehicles, and smart grids, where coordination and robustness are critical.

The key advantage of MAS lies in their flexibility, fault-tolerance, and scalability. However, challenges remain in areas such as stability under switching topologies, fault detection, and coordination in the presence of disturbances or limited communication.

3.2.3 Multi Agent iterative observer design

To estimate the states of the multi-agent system defined in (3.20), we introduce two types of output estimation errors: the distributed output estimation error $\zeta_{1i}(t)$ and the centralized output estimation error $\zeta_{2i}(t)$.

The distributed output estimation error captures the disagreement between agent i and its neighbors, and is defined as:

$$\zeta_{1i}(t) = \sum_{j \in \mathcal{N}_i} a_{ij} \left((\hat{y}_i(t) - y_i(t)) - (\hat{y}_j(t) - y_j(t)) \right)$$

The centralized output estimation error compares the estimated and actual outputs of agent i:

$$\zeta_{2i}(t) = \hat{y}_i(t) - y_i(t)$$

Here, $y_i(t)$ and $\hat{y}_i(t)$ denote the actual and estimated outputs of agent i, respectively. These errors will be used in the design of distributed iterative fault estimation observers.

3.2.3.1 One-Step Fault Estimation Approach

To obtain an initial estimate of the actuator fault for each agent i, we define a 1-step fault estimation observer as follows [26]:

$$\begin{aligned} \dot{\hat{x}}_{i1}(t) &= A(\xi)\hat{x}_{i1}(t) + B(\xi)u_i(t) + E(\xi)\hat{f}_{i1}(t) - \rho_1 H(\xi)\zeta_{1i1}(t) - \rho_2 G(\xi)\zeta_{2i1}(t) \\ \hat{y}_{i1}(t) &= C(\xi)\hat{x}_{i1}(t) \\ \dot{\hat{f}}_{i1}(t) &= -\rho_1 F_1(\xi)\zeta_{1i1}(t) - \rho_2 F_2(\xi)\zeta_{2i1}(t) \end{aligned} \tag{3.21}$$

where $\hat{x}_{i1}(t) \in \mathbb{R}^n$ is the state estimate, $\hat{y}_{i1}(t) \in \mathbb{R}^p$ is the estimated output, and $\hat{f}_{i1}(t) \in \mathbb{R}^r$ is the fault estimate at the first step. The matrices $H(\xi), G(\xi) \in \mathbb{R}^{n \times p}$ are the observer gains to be designed, while $F_1(\xi), F_2(\xi) \in \mathbb{R}^{r \times p}$ are fault estimation gain matrices.

The terms $\zeta_{1i1}(t)$ and $\zeta_{2i1}(t)$ represent the distributed and centralized output estimation errors, respectively, as defined earlier. The scalars $\rho_1, \rho_2 \geq 0$ are weighting coefficients that satisfy:

$$\rho_1 + \rho_2 = 1$$

The scalars ρ_1 and ρ_2 are often used to weight the relative importance of the two errors mentioned earlier:

- ρ_1 weights the *consensus error*, i.e., the disagreement between agents.
- ρ_2 weights the *local estimation error*, i.e., the difference between the estimated and the actual output of each agent.

These parameters allow the designer to balance the trade-off between inter-agent coordination and individual estimation accuracy.

To analyze the observer dynamics, we define the estimation errors as:

$$e_{xi1}(t) = \hat{x}_{i1}(t) - x_i(t), \quad e_{yi1}(t) = \hat{y}_{i1}(t) - y_i(t), \quad e_{fi1}(t) = \hat{f}_{i1}(t) - f_i(t)$$

Based on these definitions and the system model (3.20), the augmented estimation error dynamics can be derived and used for stability analysis and iterative estimation improvement.

To derive the estimation error dynamics of the 1-step observer, we define the augmented error vector:

$$e_{i1}(t) = \begin{bmatrix} e_{xi1}(t) \\ e_{fi1}(t) \end{bmatrix} = \begin{bmatrix} \hat{x}_{i1}(t) - x_i(t) \\ \hat{f}_{i1}(t) - f_i(t) \end{bmatrix}$$

Using the system and observer dynamics, we obtain the following differential equation for the augmented error dynamics:

$$\begin{aligned} \dot{e}_{i1}(t) &= \bar{A}(\xi)e_{i1}(t) - \bar{D}(\xi)\bar{\omega}_{i1}(t) - \rho_1\bar{H}(\xi)\sum_{j\in\mathcal{N}_i} a_{ij}\left(\bar{C}(\xi)e_{i1}(t) - \bar{C}(\xi)e_{j1}(t)\right) - \rho_2\bar{G}(\xi)\bar{C}(\xi)e_{i1}(t) \\ e_{yi1}(t) &= \bar{C}(\xi)e_{i1}(t) \end{aligned} \tag{3.22}$$

where the matrices are defined as:

$$\bar{A}(\xi) = \begin{bmatrix} A(\xi) & E(\xi) \\ 0 & 0 \end{bmatrix} \quad , \quad \bar{D}(\xi) = \begin{bmatrix} D(\xi) & 0 \\ 0 & I \end{bmatrix} \quad , \quad \bar{C}(\xi) = \begin{bmatrix} C(\xi) & 0 \end{bmatrix}$$

$$\bar{H}(\xi) = \begin{bmatrix} H(\xi) \\ F_1(\xi) \end{bmatrix} = \begin{pmatrix} \sum_{i=1}^{r} \mu_i(\xi)H_i \\ \sum_{i=1}^{r} \mu_i(\xi)F_{1i} \end{pmatrix}, \quad \bar{G} = \begin{bmatrix} G(\xi) \\ F_2(\xi) \end{bmatrix} = \begin{pmatrix} \sum_{i=1}^{s} \mu_i(\xi)G_i \\ \sum_{i=1}^{s} \mu_i(\xi)F_{2i} \end{pmatrix}$$

and the disturbance and fault vector is:

$$\bar{\omega}_{i1}(t) = \begin{bmatrix} d_i(t) \\ \dot{f}_i(t) \end{bmatrix}$$

To compactly express the global estimation error dynamics across all agents, we define:

$$e_1(t) = \begin{bmatrix} e_{11}^T(t) & \dots & e_{1N}^T(t) \end{bmatrix}^T, \quad \bar{\omega}_1(t) = \begin{bmatrix} \bar{\omega}_{11}^T(t) & \dots & \bar{\omega}_{1N}^T(t) \end{bmatrix}^T$$

Then, the global error dynamics can be expressed using Kronecker products as:

$$\begin{aligned} \dot{e}_1(t) &= \left(I_N \otimes \bar{A}(\xi) - \mathcal{L} \otimes \rho_1\bar{H}(\xi)\bar{C}(\xi) - I_N \otimes \rho_2\bar{G}(\xi)\bar{C}(\xi)\right)e_1(t) - \left(I_N \otimes \bar{D}(\xi)\right)\bar{\omega}_1(t) \\ e_{y1}(t) &= (I_N \otimes \bar{C}(\xi))e_1(t) \end{aligned} \tag{3.23}$$

This formulation shows that the fault estimate $\hat{f}(t)$ is directly affected by disturbances and modeling uncertainties. To enhance the robustness and accuracy of the estimation, we propose in the next section an extension based on the k-step fault estimation approach with $k > 2$.

3.2.3.2 Two-Step Fault Estimation Approach

To improve fault estimation accuracy and suppress the influence of the estimated fault from the 1-step observer, a 2-step fault estimation observer is constructed using the result of the first step. The observer equations for agent i at the second step are given by [26]:

$$\begin{aligned}\dot{\hat{x}}_{i2}(t) &= A(\xi)\hat{x}_{i2}(t) + B(\xi)u_i(t) + E(\xi)\hat{f}_{i2}(t) - \rho_1 H(\xi)\zeta_{1i2}(t) - \rho_2 G(\xi)\zeta_{2i2}(t)\\ \hat{y}_{i2}(t) &= C(\xi)\hat{x}_{i2}(t)\\ \dot{\hat{f}}_{i2}(t) &= -\rho_1 F_1(\xi)\zeta_{1i2}(t) - \rho_2 F_2(\xi)\zeta_{2i2}(t) + \hat{f}_{i1}(t)\end{aligned} \tag{3.24}$$

Let us define the estimation errors as:

$$e_{xi2}(t) = \hat{x}_{i2}(t) - x_i(t), \quad e_{yi2}(t) = \hat{y}_{i2}(t) - y_i(t), \quad e_{fi2}(t) = \hat{f}_{i2}(t) - f_i(t)$$

and the augmented error vector as:

$$e_{i2}(t) = \begin{bmatrix} e_{xi2}(t) \\ e_{fi2}(t) \end{bmatrix}, \quad \bar{\omega}_{i2}(t) = \begin{bmatrix} d_i(t) \\ \dot{f}_i(t) - \dot{\hat{f}}_{i1}(t) \end{bmatrix}$$

The corresponding error dynamics of the 2-step observer are then given by:

$$\begin{aligned}\dot{e}_{i2}(t) &= \bar{A}(\xi)e_{i2}(t) - \bar{D}(\xi)\bar{\omega}_{i2}(t) - \rho_1 \bar{H}(\xi)\sum_{j\in\mathcal{N}_i} a_{ij}\left(\bar{C}(\xi)e_{i2}(t) - \bar{C}(\xi)e_{j2}(t)\right) - \rho_2 \bar{G}(\xi)\bar{C}(\xi)e_{i2}(t)\\ e_{yi2}(t) &= \bar{C}(\xi)e_{i2}(t)\end{aligned} \tag{3.25}$$

where the matrices are defined as in the first step estimation. By aggregating the error dynamics across all agents, the compact global error system can be expressed as:

$$\begin{aligned}\dot{e}_2(t) &= \left(I_N \otimes \bar{A}(\xi) - \mathcal{L} \otimes \rho_1 \bar{H}(\xi)\bar{C}(\xi) - I_N \otimes \rho_2 \bar{G}(\xi)\bar{C}(\xi)\right) e_2(t) - \left(I_N \otimes \bar{D}(\xi)\right)\bar{\omega}_2(t)\\ e_{y2}(t) &= (I_N \otimes \bar{C}(\xi))e_2(t)\end{aligned} \tag{3.26}$$

This 2-step structure improves estimation accuracy by taking into account the residual between the true fault and the previous estimate. The process can be repeated to design a general k-step fault estimation observer.

3.2.3.3 k-Step Fault Estimation Approach

To further improve the fault estimation accuracy, the k-step fault estimation observer is recursively constructed by incorporating the result from the previous $(k-1)$-th step. The observer dynamics for agent i at step k are defined as [26]:

$$\begin{aligned}\dot{\hat{x}}_{ik}(t) &= A(\xi)\hat{x}_{ik}(t) + B(\xi)u_i(t) + E(\xi)\hat{f}_{ik}(t) - \rho_1 H(\xi)\zeta_{1ik}(t) - \rho_2 G(\xi)\zeta_{2ik}(t)\\ \hat{y}_{ik}(t) &= C(\xi)\hat{x}_{ik}(t)\\ \dot{\hat{f}}_{ik}(t) &= -\rho_1 F_1(\xi)\zeta_{1ik}(t) - \rho_2 F_2(\xi)\zeta_{2ik}(t) + \dot{\hat{f}}_{i(k-1)}(t)\end{aligned} \tag{3.27}$$

The associated estimation errors are defined as:

$$e_{xik}(t) = \hat{x}_{ik}(t) - x_i(t), \quad e_{yik}(t) = \hat{y}_{ik}(t) - y_i(t), \quad e_{fik}(t) = \hat{f}_{ik}(t) - f_i(t)$$

with the augmented error vector:

$$e_{ik}(t) = \begin{bmatrix} e_{xik}(t) \\ e_{fik}(t) \end{bmatrix}, \quad \bar{\omega}_{ik}(t) = \begin{bmatrix} d_i(t) \\ \dot{f}_i(t) - \dot{\hat{f}}_{i(k-1)}(t) \end{bmatrix}$$

Then, the error dynamics of the k-th observer are given by:

$$\begin{aligned}\dot{e}_{ik}(t) &= \bar{A}(\xi)e_{ik}(t) - \bar{D}(\xi)\bar{\omega}_{ik}(t) - \rho_1 \bar{H}(\xi)\sum_{j\in\mathcal{N}_i} a_{ij}\left(\bar{C}(\xi)e_{ik}(t) - \bar{C}(\xi)e_{jk}(t)\right) - \rho_2 \bar{G}(\xi)\bar{C}(\xi)e_{ik}(t)\\ e_{yik}(t) &= \bar{C}(\xi)e_{ik}(t)\end{aligned} \tag{3.28}$$

The corresponding global compact form is expressed as:

$$\begin{aligned}\dot{e}_k(t) &= \left(I_N \otimes \bar{A}(\xi) - \mathcal{L} \otimes \rho_1 \bar{H}(\xi)\bar{C}(\xi) - I_N \otimes \rho_2 \bar{G}(\xi)\bar{C}(\xi)\right)e_k(t) - \left(I_N \otimes \bar{D}(\xi)\right)\bar{\omega}_k(t)\\ e_{yk}(t) &= (I_N \otimes \bar{C}(\xi))e_k(t)\end{aligned} \tag{3.29}$$

3.2.4 Stability Analysis of the k-Step Observer

To ensure the robustness and convergence of the k-step fault estimation observer, it is essential to analyze the stability of the augmented estimation error system. This analysis is conducted under an H_∞ performance framework, which guarantees that the estimation error remains bounded and attenuates the influence of disturbances and model uncertainties.

We define a Lyapunov function candidate for the global estimation error system (3.29), and derive conditions under which its derivative is strictly negative. By formulating these conditions as an LMI, we provide sufficient criteria for the stability and H_∞ performance of the observer. The result is summarized in the following theorem.

Theorem 5. *Consider nonlinear multi-agent systems with given scalars $\gamma > 0$, for a given H_∞ performance level γ_1, if there exist a positive definite matrix Q and matrices Y_1, Y_2 of appropriate dimensions such that the following LMI holds:*

$$\begin{aligned} \chi_{ii} &< 0 \quad \text{for } i \in \{1, 2, \dots, r\} \\ \chi_{ij} + \chi_{ji} &< 0 \quad \text{for } i < j \leq r \end{aligned} \tag{3.30}$$

where

$$\chi_{ij} = \begin{bmatrix} \eta_{ij} & -(I_N \otimes Q\bar{D}_i) \\ * & -\gamma_1^2 I \end{bmatrix}$$

$$\eta_{ij} = I_N \otimes (Q\bar{A}_i + \bar{A}_i^T Q) - \mathcal{L} \otimes \rho_1 Y_{1i}\bar{C}_j - \mathcal{L}^T \otimes \rho_1 \bar{C}_j^T Y_{1i}^T - I_N \otimes \rho_2(\bar{C}_j^T Y_{2i}^T + Y_{2i}\bar{C}_j + I$$

then the global estimation error system is stable under H_∞ performance.
The observer gain matrices are computed as:

$$\bar{H}_i = Q^{-1}Y_{1i}, \quad \bar{G}_i = Q^{-1}Y_{2i}$$

Proof :
Consider the following Lyapunov function :

$$V_k(t) = e_k^T(t)(I_N \otimes Q)e_k(t)$$

Taking the time derivative of $V_k(t)$ along the trajectories of the global error system (3.29) (and neglecting the nonlinear terms), we obtain:

$$\begin{aligned} \dot{V}_k(t) = e_k^T(t)\Big(& I_N \otimes (Q\bar{A}_i + \bar{A}_i^T Q) - \mathcal{L} \otimes \rho_1 Q\bar{H}_i\bar{C}_j - \mathcal{L}^T \otimes \rho_1 \bar{C}_j^T \bar{H}_i^T Q \\ & - I_N \otimes \rho_2(\bar{G}_i^T \bar{C}_j^T Q + Q\bar{G}_i\bar{C}_j)\Big)e_k(t) - 2e_k^T(t)(I_N \otimes Q\bar{D}_i)\bar{\omega}_k(t) \end{aligned} \tag{3.31}$$

This expression is used to derive the LMI condition ensuring that $\dot{V}_k(t) < 0$, which guarantees the asymptotic stability of the estimation error under disturbances [39] .

Note that in the derivation of the Lyapunov function derivative $\dot{V}_k(t)$, the equality becomes an inequality due to the treatment of cross-product terms involving the disturbance vector $\bar{\omega}_k(t)$. Specifically, since the disturbance is assumed to belong to $L_2[0, \infty)$, and is therefore unknown but energy-bounded, it is not possible to precisely evaluate the cross term:

$$-2e_k^T(t)(I_N \otimes Q\bar{D}_i)\bar{\omega}_k(t)$$

Instead, this term is upper-bounded using the Cauchy–Schwarz inequality or quadratic completion techniques, which leads to a conservative estimation of the Lyapunov derivative. Consequently, the time derivative of the Lyapunov function is no longer expressed as an equality

but as an inequality:

$$\begin{aligned}\dot{V}_k(t) \leq e_k^T(t)\Big(I_N \otimes (Q\bar{A}_i + \bar{A}_i^T Q) - \mathcal{L} \otimes \rho_1 Q\bar{H}_i\bar{C}_j - \mathcal{L}^T \otimes \rho_1 \bar{C}_j^T \bar{H}_i^T Q \\ - I_N \otimes \rho_2(\bar{G}_i^T \bar{C}_j^T Q + Q\bar{G}_i\bar{C}_j)\Big) e_k(t) - 2e_k^T(t)(I_N \otimes Q\bar{D}_i)\bar{\omega}_k(t)\end{aligned} \tag{3.32}$$

To evaluate the H_∞ performance of the k-step observer, we define the following cost function:

$$J = \int_0^\infty \left(e_k^T(t)e_k(t) - \gamma_1^2 \bar{\omega}_k^T(t)\bar{\omega}_k(t) \right) dt \tag{3.33}$$

According to the global error system and Lyapunov derivative, and assuming zero initial conditions, we derive the following inequality:

$$J \leq \int_0^\infty \left(\dot{V}_k(t) + e_k^T(t)e_k(t) - \gamma_1^2 \bar{\omega}_k^T(t)\bar{\omega}_k(t) \right) dt \tag{3.34}$$

The integrand can be rewritten compactly as a quadratic form involving a stacked vector $\nu(t)$:

$$\nu^T(t) = \begin{bmatrix} e_k^T(t) & \bar{\omega}_k^T(t) \end{bmatrix}, \quad \chi_{ij} = \begin{bmatrix} \eta_{ij} & -I_N \otimes Q\bar{D}_i \\ * & -\gamma_1^2 I \end{bmatrix}$$

with:

$$\eta = I_N \otimes (Q\bar{A}_i + \bar{A}_i^T Q) - \mathcal{L} \otimes \rho_1 Q\bar{H}_i\bar{C}_j - \mathcal{L}^T \otimes \rho_1 \bar{C}_j^T \bar{H}_i^T Q - I_N \otimes \rho_2(\bar{G}_i^T \bar{C}_j^T Q + Q\bar{G}_i\bar{C}_j) + I$$

So, η_{ij} can be rewritten as :

$$\eta_{ij} = I_N \otimes (Q\bar{A}_i + \bar{A}_i^T Q) - \mathcal{L} \otimes \rho_1 Y_{1i}\bar{C}_j - \mathcal{L}^T \otimes \rho_1 \bar{C}_j^T Y_{1i}^T - I_N \otimes \rho_2(\bar{C}_j^T Y_{2i}^T + Y_{2i}\bar{C}_j) + I$$

Such that :

$$\bar{H}_i = Q^{-1}Y_{1i}, \quad \bar{G}_i = Q^{-1}Y_{2i}$$

$$\begin{bmatrix} \eta_{ij} & -I_N \otimes Q\bar{D}_i \\ * & -\gamma^2 I \end{bmatrix} < 0 \tag{3.35}$$

It can be seen from the formula, (3.35) is equivalent to condition (3.30), which implies (3.34) is satisfied.

3.2.5 Fault Tolerant Output Feedback Controller Design

3.2.5.1 Control Law Design

For the obtained k-step fault estimate $\hat{f}_{ik}(t)$, a fault-tolerant controller with output feedback is proposed to ensure the consistency of the multi-agent system. The control law for each agent i is designed as:

$$u_i(t) = K_s \sum_{j \in \mathcal{N}_i} a_{ij}(\hat{x}_i(t) - \hat{x}_j(t)) - B^* E \hat{f}_{ik}(t) \tag{3.36}$$

where $K_s \in \mathbb{R}^{m \times n}$ is the feedback gain matrix, and B^{-1} is a generalized inverse of the input matrix B or the pseudo-inverse in the case were matrix B is vector or not squar matrix . The control input ensures that the influence of the estimated fault is compensated while enforcing synchronization among neighboring agents.

Introducing Synchronization Error Variables :

In multi-agent systems, achieving a consistent collective behavior is as crucial as maintaining individual stability. This coordination is typically evaluated in terms of synchronization, which means that all agent states converge to a common trajectory or remain bounded relative to each other.

To capture this coordination behavior explicitly, we introduce the synchronization error variable:

$$\delta_i(t) = x_i(t) - \frac{1}{N} \sum_{j=1}^{N} x_j(t)$$

This variable measures the deviation of agent i's state from the average state of the network. Expressing the system dynamics in terms of $\delta_i(t)$ allows us to study and regulate inter-agent consensus errors and design controllers that enforce synchronization while compensating for actuator faults.

By combining the system model (3.20) with the control law (3.36), in this section we will express the model index by the index q such that $q \in \{1, 2, \ldots, r\}$ s is the number of models, the error dynamics in terms of $\delta_i(t)$ can be written as:

$$\begin{aligned} \dot{\delta}_i(t) = {} & A_q \delta_i(t) + D_q d_i(t) - E_q e_{fi}(t) + B_q K_{sq} \sum_{j \in \mathcal{N}_i} a_{ij}(\hat{x}_i(t) - \hat{x}_j(t)) \\ & - \frac{1}{N} \sum_{j=1}^{N} \sum_{l \in \mathcal{N}_j} B_k K_{sq} a_{jl}(\hat{x}_j(t) - \hat{x}_l(t)) \end{aligned} \tag{3.37}$$

where

$$\begin{aligned} \delta_i(t) &= x_i(t) - \frac{1}{N} \sum_{j=1}^{N} x_j(t) = x_i(t) - \tilde{x}(t), \\ \bar{d}_i(t) &= d_i(t) - \frac{1}{N} \sum_{j=1}^{N} d_j(t), \\ \bar{e}_{fi}(t) &= e_{fi}(t) - \frac{1}{N} \sum_{j=1}^{N} e_{fj}(t) \end{aligned}$$

Using the symmetry property of the undirected topology graph, where $a_{ij} = a_{ji}$, it is easy to show that:

$$\frac{1}{N} \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_i} B_k K_{sq} a_{ij}(\hat{x}_i(t) - \hat{x}_j(t)) = 0 \tag{3.38}$$

From the definition $e_i(t) = \hat{x}_i(t) - x_i(t)$, one can derive:

$$\sum_{j\in\mathcal{N}_i} a_{ij}(\hat{x}_i(t) - \hat{x}_j(t)) = \sum_{j\in\mathcal{N}_i} a_{ij}\left(\delta_i(t) - \delta_j(t) + e_i(t) - e_j(t)\right) \tag{3.39}$$

Combining equations (3.37), (3.2.5.1), (3.38), and (3.39), the global synchronization error dynamics become:

$$\dot{\delta}(t) = \left(I_N \otimes A_q + \mathcal{L} \otimes B_q K_{sq}\right)\delta(t) - \left(I_N \otimes E_q\right)\bar{e}_f(t) + \left(\mathcal{L} \otimes B_q K_{sq}\right)e(t) + \left(I_N \otimes D_q\right)\bar{d}(t) \tag{3.40}$$

3.2.5.2 Stability and Robust Performance Conditions

To ensure robust consensus and fault-tolerant behavior of the multi-agent system under actuator faults, it is necessary to establish sufficient conditions that guarantee synchronization and disturbance attenuation.

The following result provides a LMI condition under which the H_∞ fault-tolerant consensus is ensured for the system described by the synchronization error dynamics in (3.40), and the control law in (3.36).

Theorem 6. *[26] The problem of robust H_∞ fault-tolerant consensus for systems* (3.20) *with actuator faults can be solved under the fault-tolerant consensus protocol* (3.36)*, the Laplacian matrix $\mathcal{L}$, and for a scalar $\gamma_2 > 0$ and a positive definite matrix P, if there exist matrices A_q, B_q, D_q, E_q, and K_{sq} of compatible dimensions such that the following LMI holds.*

$$\begin{bmatrix} \Psi_{11} & \Psi_{12} & \Psi_{13} & \Psi_{14} \\ * & \Psi_{22} & 0 & 0 \\ * & * & \Psi_{33} & 0 \\ * & * & * & \Psi_{44} \end{bmatrix} < 0 \tag{3.41}$$

where:

$$\Psi_{11} = He(PA_q + \alpha_j H_{sq}) + I, \quad \Psi_{12} = \alpha_j H_{sq}, \quad \Psi_{13} = -PE_q,$$
$$\Psi_{14} = PD_q, \quad \Psi_{22} = -\gamma_2^2 I, \quad \Psi_{33} = -\gamma_2^2 I, \quad \Psi_{44} = -\gamma_2^2 I$$

with

$$H_s = PB_q K_{sq} \quad He(M) = M + M^T \quad q \in \{1, 2, \ldots, r\}$$

Proof :
Due to (3.40), use a Lyapunov function candidate to :

$$V(t) = \delta^T(t)(I_N \otimes P)\delta(t) \tag{3.42}$$

Taking its time derivative along the trajectory of the synchronization error system (3.40), we obtain:

$$\begin{aligned}\dot{V}(t) = \delta^T(t)\mathrm{He}(I_N \otimes PA_q + \mathcal{L} \otimes PB_qK_{sq})\delta(t) \\ - 2\delta^T(t)(I_N \otimes PE_q)\bar{e}_f(t) + 2\delta^T(t)(\mathcal{L} \otimes PB_qK_{sq})e(t) \\ + 2\delta^T(t)(I_N \otimes PD_q)\bar{d}(t)\end{aligned} \tag{3.43}$$

Since the Lyapunov derivative expression (3.43) includes uncertain or unmeasurable terms (e.g., $\bar{e}_f(t)$ and $\bar{d}(t)$), we consider a conservative upper bound on $\dot{V}(t)$ by relaxing the exact dynamics and focusing on the quadratic structure. Therefore, we write:

$$\begin{aligned}\dot{V}(t) \leq \delta^T(t)\left(\mathrm{He}(I_N \otimes PA_q + \mathcal{L} \otimes PB_qK_{sq})\right)\delta(t) \\ - 2\delta^T(t)(I_N \otimes PE_q)\bar{e}_f(t) + 2\delta^T(t)(\mathcal{L} \otimes PB_qK_{sq})e(t) + 2\delta^T(t)(I_N \otimes PD_q)\bar{d}(t)\end{aligned} \tag{3.44}$$

To simplify the analysis and decouple the modes of the multi-agent system, we perform a coordinate transformation:

$$\epsilon(t) = (\Pi^T \otimes I_n)\delta(t) \tag{3.45}$$

Here, Π is a matrix that diagonalizes the Laplacian matrix $\mathcal{L}$, i.e., $\Pi^{-1}\mathcal{L}\Pi = \Lambda$, where Λ is diagonal. This transformation decomposes the global synchronization error into modal components associated with the eigenvalues of $\mathcal{L}$.

The diagonalization simplifies the analysis by isolating the consensus mode (associated with eigenvalue 0) from the non-consensus (disagreement) modes. Since $\sum_{i=1}^{N}\delta_i(t) = 0$, the consensus mode vanishes (i.e., $\epsilon_1(t) = 0$), and only the disagreement modes contribute to the Lyapunov evolution. This allows us to write:

$$\dot{V}(t) \leq \sum_{j=2}^{N}\epsilon_j^T(t)\left(\mathrm{He}(PA_q + \alpha_j PB_kK_{sq})\right)\epsilon_j(t) \tag{3.46}$$

Inequality (3.46) can be expressed as:

$$\dot{V}(t) \leq \sum_{j=2}^{N}\epsilon_j^T(t)\chi_{jk}\epsilon_j(t) \tag{3.47}$$

where each matrix χ_{jk} is defined by:

$$\chi_j = \mathrm{He}(PA_k + \alpha_j PB_qK_{sq})$$

It is obvious that if $\chi_{jq} < 0$, then $\dot{V}(t) < 0$ for any $\epsilon_j(t) \neq 0$, which implies that the consensus errors are guaranteed to decay over time.

Now, let us consider the robust performance in the presence of nonzero disturbances. For this purpose, we introduce the following coordinate transformations:

$$\vartheta(t) = (\Pi^T \otimes I_n)e(t), \quad \kappa(t) = (\Pi^T \otimes I_r)\bar{e}_f(t), \quad \sigma(t) = (\Pi^T \otimes I_q)\bar{d}(t)$$

where Π is the matrix that diagonalizes the Laplacian matrix $\mathcal{L}$. These transformed variables represent the decoupled components of the estimation error, fault estimation error, and disturbance input, respectively, in the modal coordinates.

Note: Based on the structure of the Laplacian matrix and the assumption $\sum_{i=1}^{N} \delta_i(t) = 0$, it follows that $\epsilon_1(t) = 0$, and consequently, $\kappa_1(t) = 0$, $\sigma_1(t) = 0$, the inequality (3.44) can be rewritten as:

$$\begin{aligned} \dot{V}(t) \leq \sum_{j=2}^{N} \epsilon_j^T(t)\Big(\mathrm{He}(PA_q + \alpha_j PB_q K_{sq})\Big)\epsilon_j(t) - 2\sum_{j=2}^{N} \epsilon_j^T(t)(PE_q)\kappa_j(t) \\ +2\sum_{j=2}^{N} \epsilon_j^T(t)(\alpha_j PB_q K_{sq})\vartheta_j(t) + 2\sum_{j=2}^{N} \epsilon_j^T(t)(PD_q)\sigma_j(t) \end{aligned} \tag{3.48}$$

Let the transformed disturbance vector be defined as:

$$\bar{\omega}(t) = \begin{bmatrix} \vartheta^T(t) & \kappa^T(t) & \sigma^T(t) \end{bmatrix}^T$$

Then, the following H_∞ performance index is considered:

$$J_r = \int_0^\infty \left(\epsilon^T(t)\epsilon(t) - \gamma_2^2 \bar{\omega}^T(t)\bar{\omega}(t)\right) dt \tag{3.49}$$

Let us define the combined signal vector:

$$\rho_j(t) = \begin{bmatrix} \epsilon_j^T(t) & \vartheta_j^T(t) & \kappa_j^T(t) & \sigma_j^T(t) \end{bmatrix}^T$$

From inequality (3.48) and assuming zero initial conditions, the performance index J_r satisfies:

$$J_r \leq \int_0^\infty \left(\sum_{j=2}^{N} \rho_j^T(t)\Xi_{jq}\rho_j(t) - \gamma_2^2 \vartheta_1^T(t)\vartheta_1(t)\right) dt - V(T) \tag{3.50}$$

Ξ_{jq} is described in the theorem (6), if $\Xi_{jq} < 0$ (for $j = 2, 3, \ldots, N$, $q \in \{1, 2, \ldots, s\}$), then the performance index function $J_r < 0$ is satisfied.

3.2.5.3 Example 2.2

Consider multi-agent system comprising four agents, and each agent can be modelled as follows:

$$\begin{cases} \dot{x}_j(t) = \sum_{i=1}^{2} \mu_i(z(t)) \left(A_i x_j(t) + B_i u_j(t) + E_i f_j(t) + D_i d_j(t)\right) \\ y_j(t) = \sum_{i=1}^{2} \mu_i(z(t))\, C_i\, x_j(t) \end{cases} \qquad j = (1, ..., 4)$$

$$A_1 = \begin{bmatrix} -2 & 1 \\ -1 & -1 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 0.4 \\ 0.5 \end{bmatrix}, \quad E_1 = B_1, \quad D_1 = \begin{bmatrix} 0.01 \\ 0.02 \end{bmatrix}, \quad C_1 = \begin{bmatrix} 1 & -0.01 \\ 0.1 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} -1 & 0.5 \\ -0.9 & -0.5 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0.5 \\ 0.2 \end{bmatrix}, \quad E_2 = B_2, \quad D_2 = \begin{bmatrix} 0.02 \\ 0.05 \end{bmatrix}, \quad C_2 = \begin{bmatrix} 0.6 & 0 \\ 0.2 & 1.5 \end{bmatrix}$$

The network can be expressed in the following topology in Fig. 3.4.

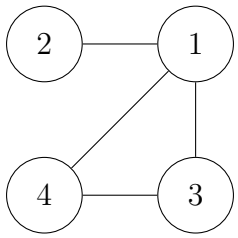


Figure 3.4: Communication topology

From Fig. 3.4, the Laplacian matrix $\mathcal{L}$ can be given as follows:

$$\mathcal{L} = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

By applying theorem (5) to the system and fixing the H_∞ performance index $\gamma_1 = 0.5$, we derive the observer gains:

$$H_1 = 10^{-11} \begin{bmatrix} -0.0900 & 0.0280 \\ -0.1360 & 0.0210 \end{bmatrix}, \quad G_1 = \begin{bmatrix} 62.8000 & 25.5000 \\ 83.0000 & 31.8000 \end{bmatrix},$$

$$F_{11} = 10^{-10} \begin{bmatrix} -0.1756 & 0.0454 \end{bmatrix}, \quad F_{21} = 10^{3} \begin{bmatrix} 1.4050 & 0.5377 \end{bmatrix},$$

$$H_2 = 10^{-11} \begin{bmatrix} 0.1650 & -0.0330 \\ 0.2770 & -0.0430 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 123.4000 & 8.0000 \\ 152.4000 & 10.8000 \end{bmatrix},$$

$$F_{12} = 10^{-10} \begin{bmatrix} 0.3395 & -0.0697 \end{bmatrix}, \quad F_{22} = 10^{3} \begin{bmatrix} 2.7022 & 0.1482 \end{bmatrix},$$

To stabilize the system, we implement the output feedback control based on theorem (6) with $\gamma_2 = 0.3$ the matrix gain is as follow :

$$K_{s1} = \begin{bmatrix} -0.0638 & -0.0936 \end{bmatrix}, \quad K_{s2} = \begin{bmatrix} -0.1524 & -0.0536 \end{bmatrix}$$

The actuator fault signals are defined as follows in every agent :

$$f_1(t) = \begin{cases} 0, & t < 5 \\ 20(t-5)e^{-2(t-5)}, & t \geq 5 \end{cases}, \qquad f_2(t) = \begin{cases} 0, & t < 5 \\ 2(t-5), & 5 \leq t \leq 7 \\ 4, & t > 7 \end{cases}$$

$$f_3(t) = \begin{cases} 0, & t < 5 \\ 10\left(1 - e^{-2(t-5)}\right), & t \geq 5 \end{cases}, \qquad f_4(t) = 10 \cdot e^{(-0.25 \cdot (t-5))}$$

The external disturbance is defined by:

$$d_j(t) = 0.08 \cdot e^{-0.01t} \cdot \sin(2\pi t), \quad j = (1, ..., 4)$$

The tolerance error is fixed as: $\epsilon = 0.2$ and by setting the initial conditions at The initial states of the four agents are given by: $x_{1,0} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $x_{2,0} = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$, $x_{3,0} = \begin{bmatrix} 2 \\ -0.5 \end{bmatrix}$, and $x_{4,0} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$, we will compare the fault estimation at this iteration with the one-step estimation and analyze the results.

Simulation Results :

In the first iteration, the shape and dynamics of the injected faults are generally captured by the estimator. However, the estimation lacks precision, as illustrated in Figure 3.5. The maximum of the estimation errors are respectively : 1.20, 0.197, 1.472 , 2.92

This limitation is reflected in the corresponding estimation error shown in Figure 3.6, where the maximum error reaches approximately 2.92 for Agent 4. Such a large deviation can have a serious impact on closed-loop performance and even threaten system stability. This influence can also be observed in the state evolution Figure 3.7, where state trajectories show significant deviation and oscillation, particularly for agents affected by large faults.

These observations highlight the importance of improving the accuracy of fault estimation before engaging in fault-tolerant control. To address this, we employ a multi-step iterative estimation strategy, where the process is repeated until a given error tolerance is satisfied.

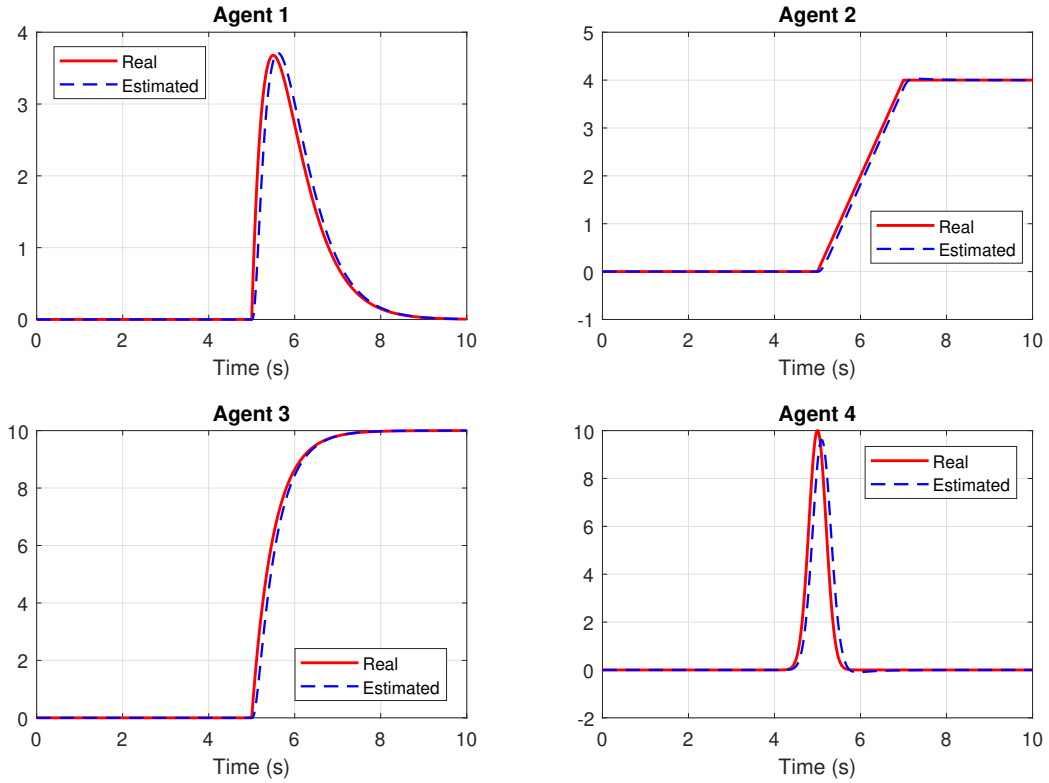


Figure 3.5: Fault Estimation - One step fault estimation

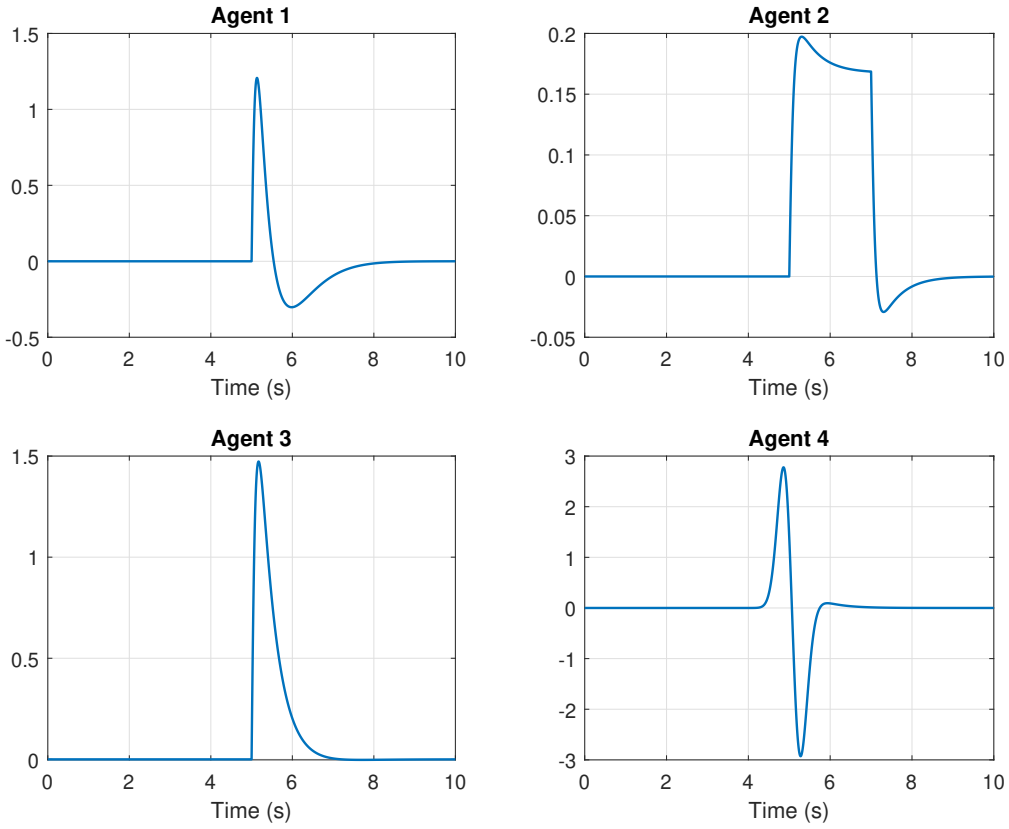


Figure 3.6: Estimation error - One step fault estimation

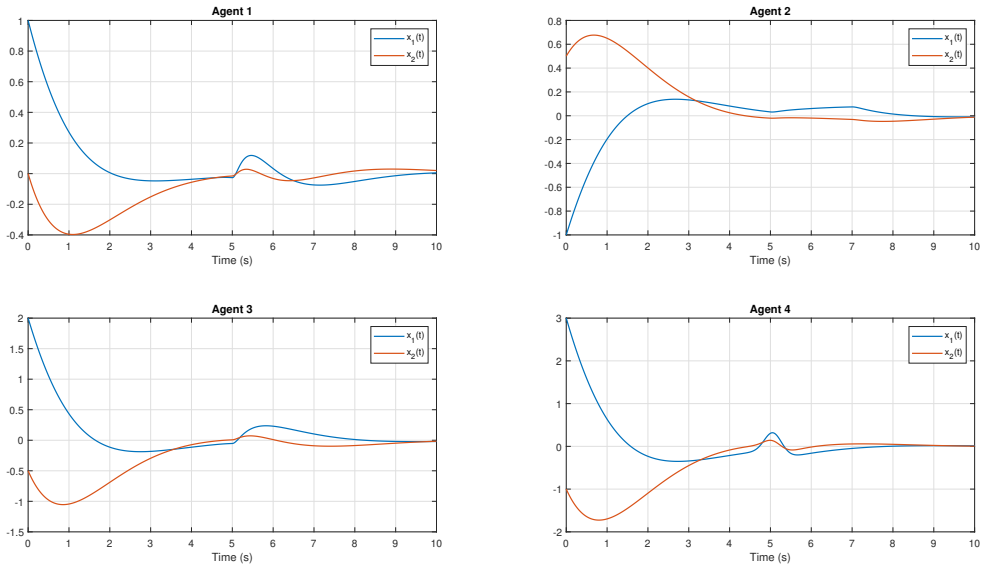


Figure 3.7: State evolution - One step fault estimation

In our case, by setting the tolerance threshold to 0.5, convergence is achieved at the 7^{th} iteration. Although smaller thresholds could be considered to further improve estimation, they come at the cost of increased computational time. It is important to note that prior knowledge of the nominal model allows for an informed choice of the tolerance value that balances precision and efficiency. The maximum of the fault estimation errors are respectively : 0.41 , 0.04, 0.432

0.477.

At the 7^{th} iteration, the estimated faults closely match the actual faults, as shown in Figure 3.8. In parallel, the estimation error decreases significantly and remains below the set threshold, with a peak of around 0.47 Figure 3.9. As a result, the reconstructed faults are more reliable and allow for better fault compensation. This improvement is reflected in the evolution of the agents' states (Figure 3.10), which now converge more smoothly and exhibit behavior consistent with stability requirements.

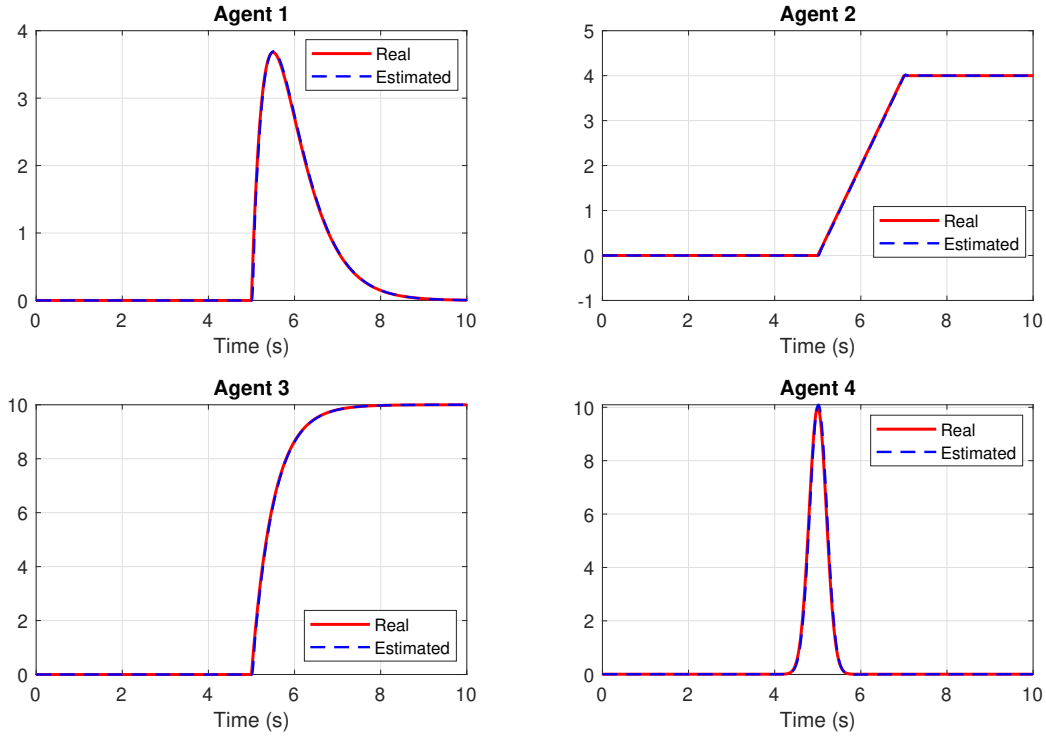


Figure 3.8: Fault Estimation - 7 step fault estimation

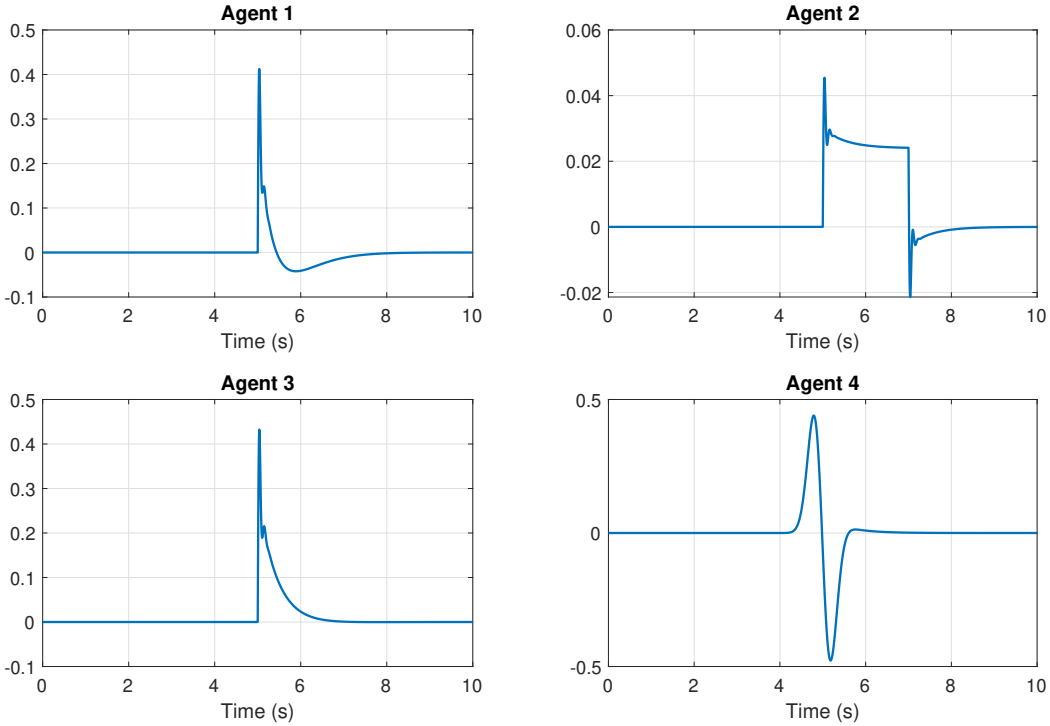


Figure 3.9: Estimation error - 7 step fault estimation

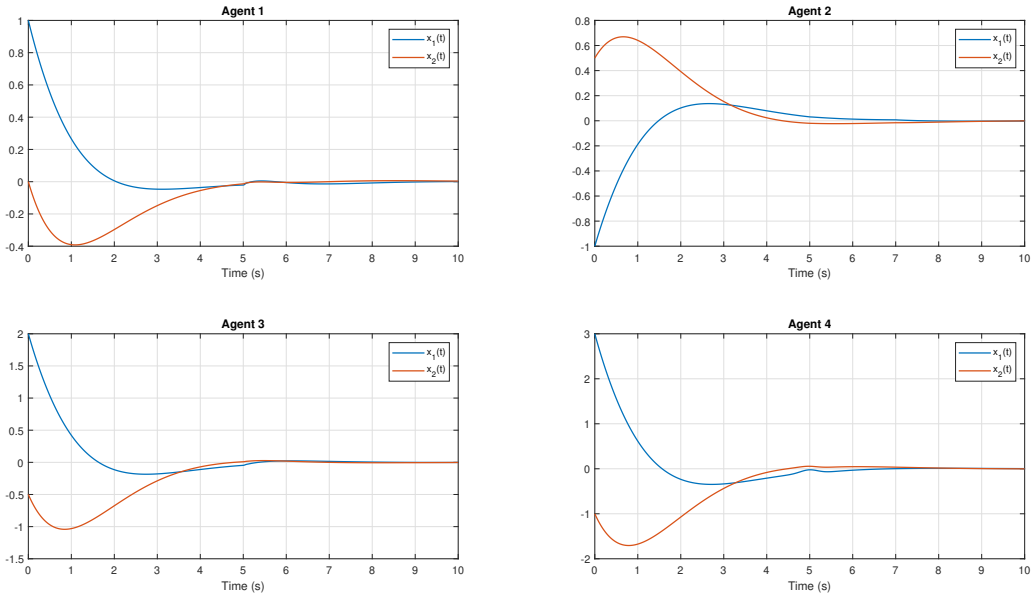


Figure 3.10: State evolution - 7 step fault estimation

It is important to emphasize that the goal of this chapter is to demonstrate the iterative fault estimation process, and not to assess the performance of the fault-tolerant controller itself. As such, closed-loop performance and tracking are not the focus here. In the following chapter, we will investigate advanced control strategies capable of operating in real-world fault scenarios, including adaptive and robust FTC schemes.

Special case :

In this special case, we evaluate the robustness of the proposed distributed estimation fault by introducing a fault in only one agent and verifying whether it influences the fault estimation results of the other agents.

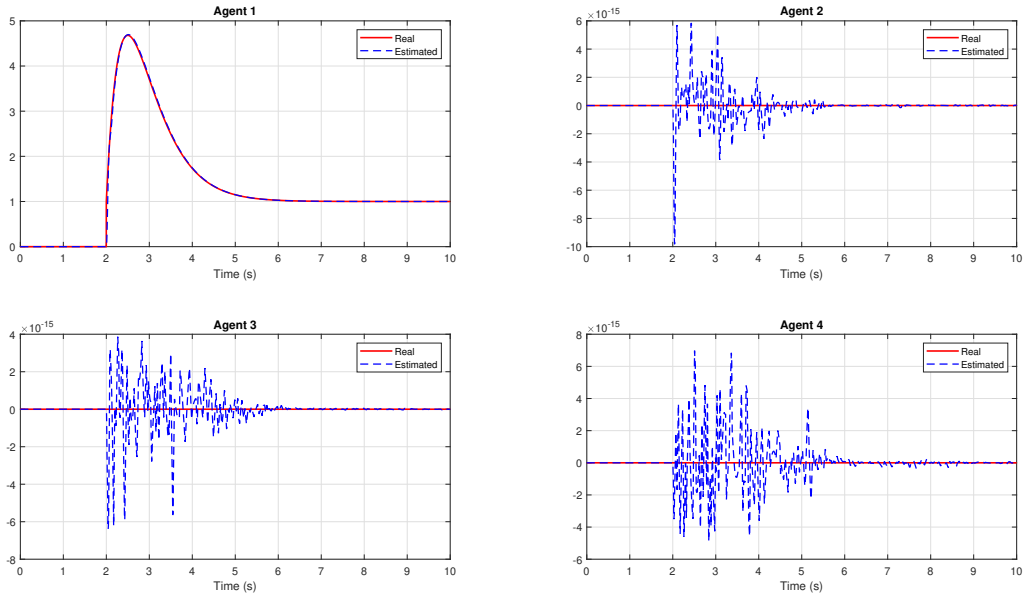


Figure 3.11: 7-Step Fault estimation in one agent

As shown in Figure 3.11, the actual fault injected in a single agent is accurately estimated, while the estimators of the remaining agents produce negligible outputs close to 10^{-15} and for error around zero in the fault estimation. This behavior is mainly due to numerical round-off errors inherent to double-precision computations and minor inter-agent coupling effects through the communication topology. The algorithm successfully rejects such insignificant signals, confirming its ability to isolate the faulty agent without generating false positives in healthy agents.

Remarque :

- The estimation process at higher iterations could be improved by recording the dynamics of certain fault signals. This would allow the estimator to start from non-zero initial iterations, thus avoiding unnecessary time loss during the early stages. However, this approach requires the use of non-volatile memory to store historical data, which introduces additional hardware considerations. Therefore, this method should be explored in future work to enhance estimation performance and intelligence.

- In our current work, we have assumed a fixed communication topology among agents during the fault estimation process, which simplifies the design and enables us to validate the proposed iterative observer-based estimation strategy under controlled conditions. In future studies, we plan to address more realistic and complex scenarios by allowing the

communication topology to vary over time, depending on the application context—such as agent mobility, network disruptions, or environmental changes. In such cases, observer gains must be adapted in real-time according to the current interaction graph, introducing new challenges like the development of distributed adaptive observers, online gain computation, and ensuring convergence under switching topologies. These scenarios will require advanced stability analysis tools such as common Lyapunov functions or dwell-time methods and will impose greater demands on both computational and communication resources.

Conclusion

In this chapter, we have developed a robust and scalable framework for iterative fault estimation in both single-agent and multi-agent nonlinear systems modeled by T-S fuzzy representations. We first demonstrated how the k-step observer improves estimation accuracy by compensating for residual errors in successive iterations. A Lyapunov-based stability analysis was presented, guaranteeing convergence and robustness through an H_∞ performance index.

The methodology was then extended to multi-agent systems with interconnected dynamics and actuator faults. By combining centralized and distributed estimation errors, we used an observer capable of exploiting inter-agent information to enhance fault reconstruction. A distributed fault-tolerant controller was also introduced, which ensures synchronization and compensation despite actuator failures and external disturbances.

Simulation results validated the approach, highlighting the importance of accurate fault estimation in achieving stable and resilient multi-agent coordination. Although this work focused primarily on the estimation procedure, the next chapter will shift attention to the design of advanced control strategies, including adaptive and robust fault-tolerant control, for real-world deployment under uncertainty and more complex fault scenarios.

Chapter 4

Fault Tolerant Control for Differential Drive Robots

Introduction

The transition from theoretical fault estimation frameworks to practical applications is crucial for validating their efficiency and robustness. In this chapter, we investigate the implementation of our previously developed fault estimation approach based on k-step observers on a real-time control system. Specifically, we apply it to the kinematic model of a differential drive mobile robot, both in single agent and multi agent configurations.

Mobile robots are widely used in real-world applications such as logistics, service robotics, and autonomous exploration. However, ensuring reliable performance in the presence of actuator faults is a significant challenge. The integration of fault-tolerant control (FTC) mechanisms, particularly those capable of estimating and compensating for unknown faults in real time, is therefore essential for safe operations.

Our objective is to demonstrate how the fault estimation method enhances the robustness of a differential-drive robot by enabling continuous control despite actuator faults. This chapter serves as a bridge between theory and implementation, setting the foundation for future real-time validation in robotic middleware environments.

4.1 Mobile robot modelling

Before proceeding to the kinematic modeling of the differential-drive mobile robot, it is important to justify the rationale behind this choice. The kinematic model is widely favored in control design due to its simplicity, lower computational cost, and ease of implementation. More importantly, in the case of differential drive robots, actuator faults such as wheel motor degradation or command signal corruption often manifest directly at the input level of the kinematic model. This is because the kinematic behavior results from the low-level actuation dynamics, and thus any faults affecting the actuators or the inputs of the dynamic model ultimately influence the kinematic control inputs, i.e., the linear and angular velocities (v, ω)

This characteristic is particularly advantageous in fault-tolerant control schemes. By working with the kinematic model, one can detect and estimate actuator faults indirectly by observing deviations in the kinematic behavior, without requiring full knowledge of the underlying

dynamics. While the type and structure of the fault may be transformed, i.e., its effect may appear in a different form, the presence of the fault itself remains observable. This backward propagation from observed kinematic deviations to fault estimation enables the application of high-level observer-based FTC methods with reduced model complexity.

4.1.1 Kinematic model

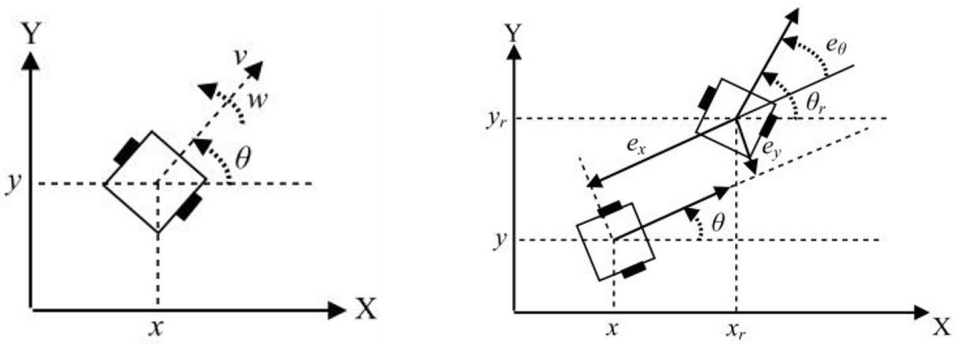
The differential mobile robot is modeled using the standard nonholonomic kinematic model under the assumption of no lateral slip. The system's state is defined by the position (x, y) and orientation θ, while the control inputs are the linear and angular velocities v and ω, respectively. The kinematic equations are given by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{4.1}$$

4.1.2 Kinematic error model

To formulate a trajectory tracking problem, we define a reference trajectory generated by a virtual robot with states (x_r, y_r) and control inputs (v_r, ω_r). The posture error in the robot's local frame is computed using a rotation matrix transformation as proposed by [40]:

$$\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \tag{4.2}$$



(a) Mobile robot model (b) Posture error

Considering that the imaginary mobile robot has a kinematic model like in equation (4.1), by deriving the equation (4.2) we obtain the following kinematic model:

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_r \\ w_r \end{bmatrix} + \begin{bmatrix} -1 & e_y \\ 0 & -e_x \\ 0 & -1 \end{bmatrix} u \tag{4.3}$$

where v_r, w_r are the linear and angular reference velocities, respectively, and u is the control input.

4.1.3 Linearization

In order to facilitate the linearization of the kinematic error model, a feedforward control action u_F is introduced. This term is defined as a nonlinear transformation of the reference inputs, and it allows us to explicitly cancel or compensate certain nonlinear components present in the error dynamics.

The control input is defined as the sum of the feedforward and feedback control inputs as follows [41]:

$$u = u_F + u_B = \begin{bmatrix} v_r \cos e_\theta \\ w_r \end{bmatrix} + \begin{bmatrix} v \\ w \end{bmatrix} \tag{4.4}$$

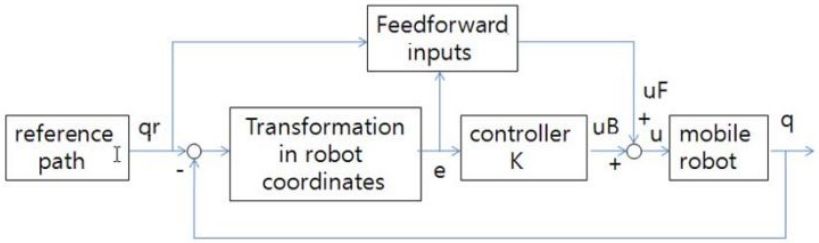


Figure 4.2: Structure of mobile robot control system

As a result, the remaining terms represented by u_B, the feedback control action are easier to handle in a linearized framework.

Substituting equation (4.4) into equation (4.3), the resulting model is given by :

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta \end{bmatrix} = \begin{bmatrix} -v + e_y(w_r + w) \\ \sin e_\theta \cdot v_r - e_x(w_r + w) \\ -w \end{bmatrix} \tag{4.5}$$

By linearizing the error model in equation (4.5) around the reference configuration ($e_x = e_y = e_\theta = 0$, $u_{B1} = u_{B2} = 0$), we obtain the following linear kinematic error model:

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta \end{bmatrix} = \begin{bmatrix} 0 & w_r & 0 \\ -w_r & 0 & v_r \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{4.6}$$

Remark 1: v_r and w_r can be given as in [42] or should be calculated by:

$$v_r = \pm\sqrt{\dot{x}_r^2 + \dot{y}_r^2} \quad \text{and} \quad w_r = \frac{\dot{y}_r \ddot{x}_r - \dot{x}_r \ddot{y}_r}{\dot{x}_r^2 + \dot{y}_r^2}$$

4.2 Control Strategy

In this section, we present the control strategy developed for the differential-drive mobile robot to ensure both trajectory tracking and obstacle avoidance. Given the nature of the robot's kinematics, the control objective is twofold: the robot must accurately follow a predefined reference trajectory while simultaneously avoiding static obstacles encountered along its path.

To achieve this, we design two separate control laws: one dedicated to trajectory tracking and another specifically for obstacle avoidance. Since the linearized kinematic error model is dependent on the reference angular velocity, we adopt a Takagi–Sugeno fuzzy modeling approach. This allows us to represent the nonlinear error dynamics as a collection of local linear sub-models, each associated with specific values of the reference velocities.

Accordingly, the control inputs are generated using a fuzzy parallel distributed compensation scheme, where a state feedback controller is designed for each sub-model. To ensure a smooth and intelligent balance between tracking and avoidance behaviors, a fuzzy fusion mechanism is introduced. This fusion gain determines how the outputs of the two controllers are combined in real time, based on the proximity and orientation of obstacles.

4.3 Trajectory Tracking Control

The first component of the control architecture is responsible for making the mobile robot follow a given reference trajectory. To achieve this, we design a state-feedback controller based on the tracking error dynamics. The methodology follows a structure similar to gain scheduling, where the system is represented as a collection of linearized models around different operating conditions.

In our case, since the error model is derived by linearizing the kinematic model around the reference trajectory (x_r, y_r, θ_r), the resulting dynamics are centered around the origin of the error space $(e_x, e_y, e_\theta) = (0, 0, 0)$. This justifies the use of a linear error feedback controller, as the objective is to drive the tracking error to zero.

To account for the nonlinearities induced by varying reference velocities, we adopt a T-S fuzzy modeling framework. The global error dynamics are represented as a weighted combination of r local linear sub-models. For each sub-model, a linear feedback gain K_i is computed to stabilize the corresponding dynamics.

The overall tracking control law is then constructed as [41]:

$$u(t) = -\sum_{i=1}^{r} \mu_i(\xi(t)) K_i R(\theta)^T e(t) \tag{4.7}$$

where:

- $e(t) \in \mathbb{R}^3$ is the state vector of tracking errors,
- $K_i \in \mathbb{R}^{2\times 3}$ is the feedback gain matrix associated with the i-th sub-model,
- $R(\theta) \in \mathbb{SO}_3$ is the rotation matrix from the global frame to the robot frame,
- $\mu_i(\xi(t)) \in [0, 1]$ are the normalized activation functions that depend on the scheduling variables $\xi(t)$, such as the reference velocities $v_r(t)$ and $\omega_r(t)$.

The rotation matrix $R(\theta)$ is introduced because control inputs must be applied in the robot's local frame. Since we operate in the global reference frame, it is necessary to transform the control inputs into the robot's frame to ensure they are correctly interpreted and executed.

This fuzzy control strategy generates the linear velocity v_t and angular velocity ω_t required for trajectory tracking. By ensuring a smooth interpolation between the different local controllers, it guarantees a continuous and stable response, and drives the tracking errors asymptotically to zero, provided that the feedback gains are appropriately chosen.

4.4 Obstacle Avoidance Control

In addition to trajectory tracking, the robot must be able to avoid obstacles that may appear along its path. Assuming accurate knowledge of the robot and obstacle positions, an elementary avoidance maneuver is triggered whenever the robot detects a nearby obstacle.

The idea is to momentarily shift the reference to a virtual error that pushes the robot away from the obstacle, and then smoothly bring it back toward the original trajectory. This is achieved by generating a smooth repulsive reference error defined as:

$$e_r(t) = \frac{F}{{D_{ob}}^2} \begin{bmatrix} x_{ob} - x \\ y_{ob} - y \\ \theta - \arctan(\frac{y_{ob}-y}{x_{ob}-x}) \end{bmatrix} \tag{4.8}$$

where D_{ob} is the distance to the obstacle, F is the fusion gain, and (x_{ob}, y_{ob}) is the position of the obstacle.

The resulting avoidance control input is given by:

$$u(t) = -\sum_{i=1}^{r} \mu_i(\xi(t)) K_i R(\theta)^T (e(t) - e_r(t)) \tag{4.9}$$

This control law generates the linear velocity v_{ob} and angular velocity ω_{ob} that allow the robot to perform obstacle avoidance maneuvers. The robot responds as if it were compensating for the virtual error e_r, resulting in a temporary deviation from the reference trajectory to bypass the obstacle. The fusion coefficient F plays a key role in modulating this behavior. As the robot distances itself from the obstacle, e_r gradually diminishes, and the control input smoothly transitions back to nominal tracking.

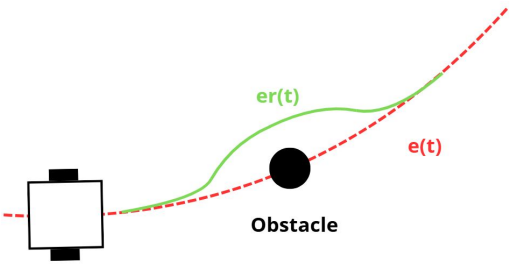
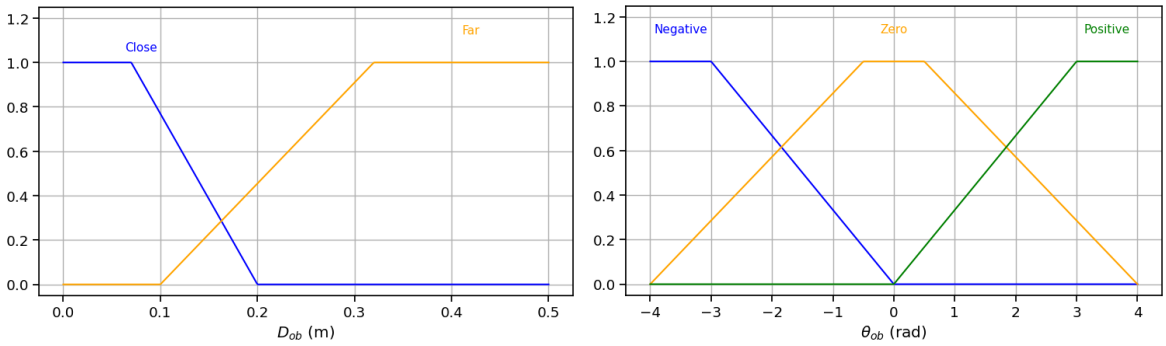


Figure 4.3: Obstacle avoidance

4.5 Fusion controller

In the presence of obstacles on the desired path, the mobile robot must avoid them while it is still tracking the virtual mobile robot, therefore, the mobile robot will be controlled by the tracking velocities (v_T, ω_T) and the obstacle avoidance velocities $(v_{\text{ob}}, \omega_{\text{ob}})$ simultaneously. This is accomplished using a Mamdani fuzzy controller, whose inputs are the current distance (D_{ob}) and the angle (θ_{ob}) between the mobile robot and each obstacle encountered on the desired trajectory, and its output is the fusion gain (F). D_{ob} has two fuzzy sets: close and far, which are spread over the range $[0, 0.5\,\text{m}]$, θ_{ob} range is $[-4, 4]\,\text{rad}$ divided into three fuzzy sets: negative, zero and positive, and F domain is $[0, 1]$ divided into three fuzzy sets: little, more and lots. The forms of the activation functions are presented in Figures 4.4a, 4.4b and 4.5 . The fuzzy rules used to define the values of the fusion gain are summarized in Table 4.1.



(a) Membership function for D_{ob}

(b) Membership function for θ_{ob}

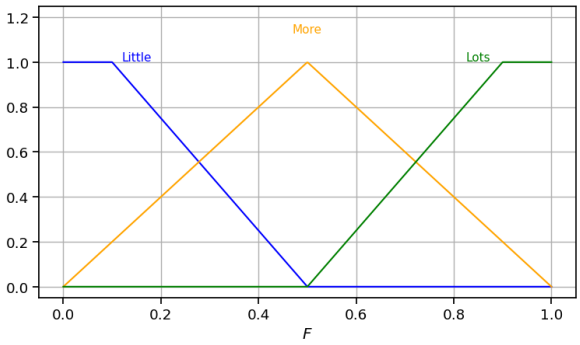


Figure 4.5: Membership function for F

To ensure a smooth and stable coordination between trajectory tracking and obstacle avoidance, the fusion factor F is designed as a fuzzy variable rather than a binary switch. This approach avoids completely disregarding one control objective in favor of the other. For instance, when the robot encounters a nearby obstacle, the avoidance command should dominate; however, maintaining a small contribution from the tracking command helps preserve awareness of the global reference trajectory, thus preventing divergence or disorientation.

The control inputs of the mobile robot are obtained as follows:

$$\begin{cases} v = (1 - F)v_T + Fv_{\text{ob}} \\ \omega = (1 - F)\omega_T + F\omega_{\text{ob}} \end{cases} \tag{20}$$

Rule number	**Linguistic inputs**		**Linguistic outputs**
	D_{ob}	θ_{ob}	F
1	Close	Negative	More
2	Close	Zero	Lots
3	Close	Positive	More
4	Far	Negative	Little
5	Far	Zero	Little
6	Far	Positive	Little

Table 4.1: The fuzzy rules to determine the fusion gain values

4.6 Control Implementation

To evaluate the performance of the proposed control strategy, a Matlab simulation has been conducted and the trajectory tracking was performed.

The linear and angular velocities of the virtual mobile robot have been used as premise variables, and they were expressed by these equations as in [41]:

$$v_r(t) = 1 + 5\exp(-2t), \quad \omega_r(t) = 5\sin(0.01t) \quad [\text{rad/s}]$$

The initial states of the mobile robot and the virtual mobile robot were: $q(0) = [-0.5, 2, 0]^T$ and $q_r(0) = [0, 0, 0]^T$. The range of $v_r(t)$ is $[1, 6]$ m/s and the range of $\omega_r(t)$ is $[-10, 10]$ rad/s (we let the possibility to get the max equals 10 rad/s in the angular velocity). According to the lower and upper bounds of both reference velocities, the following subsystems were found:

$$A_1 = \begin{bmatrix} 0 & -10 & 0 \\ 10 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & 10 & 0 \\ -10 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 0 & -10 & 0 \\ 10 & 0 & 6 \\ 0 & 0 & 0 \end{bmatrix}, \quad A_4 = \begin{bmatrix} 0 & 10 & 0 \\ -10 & 0 & 6 \\ 0 & 0 & 0 \end{bmatrix}$$

To obtain more than four subsystems, other values of $v_r(t)$ and $\omega_r(t)$ should be taken in their ranges.

From equation (4.6), the matrix B_i will always have the same values, which are:

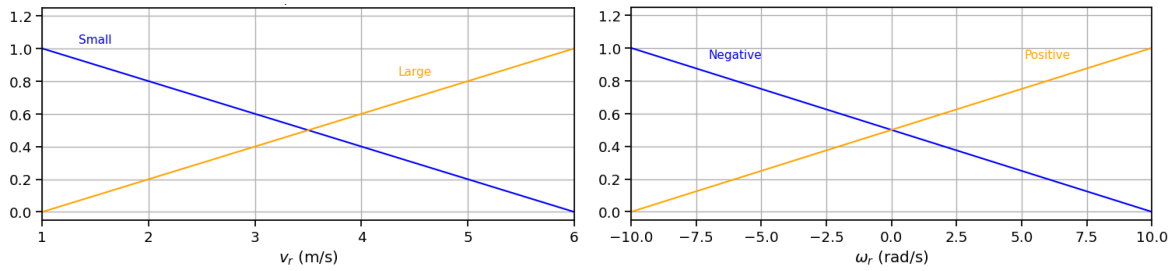
$$B_1 = B_2 = B_3 = B_4 = \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix}$$

The T-S fuzzy rules are given as follows:

- Rule 1: If v_r is small and ω_r is negative, then $\dot{e}(t) = A_1 e(t) + B_1 u(t)$
- Rule 2: If v_r is small and ω_r is positive, then $\dot{e}(t) = A_2 e(t) + B_2 u(t)$

- Rule 3: If v_r is large and ω_r is negative, then $\dot{e}(t) = A_3 e(t) + B_3 u(t)$
- Rule 4: If v_r is large and ω_r is positive, then $\dot{e}(t) = A_4 e(t) + B_4 u(t)$

The activation functions used in the T-S fuzzy model and controller are presented in figure 4.6a and 4.6b.



(a) Membership function for v_r (b) Membership function for ω_r

By imposing a closed-loop dynamics within a region defined by $\alpha = 10$, the following state feedback gains were obtained:

$$K_1 = \begin{bmatrix} -20.311658 & -1685.1992 & -98.509052 \\ -4.538567 & -515.5166 & -41.46704 \end{bmatrix}; \quad K_2 = \begin{bmatrix} -20.311658 & 1720.1016 & 101.16299 \\ 1.117774 & -515.5166 & -41.46704 \end{bmatrix}$$

$$K_3 = \begin{bmatrix} -20.311658 & -1704.174 & -99.951873 \\ -2.6788407 & -5553.2243 & -336.54994 \end{bmatrix}; \quad K_4 = \begin{bmatrix} -20.311658 & 1724.9306 & 101.53018 \\ 0.64448126 & -5553.2243 & -336.54994 \end{bmatrix}$$

The tracking error converges to zero figure 4.7 within a short period of time, primarily due to the high values of the state feedback gains used in the control design.

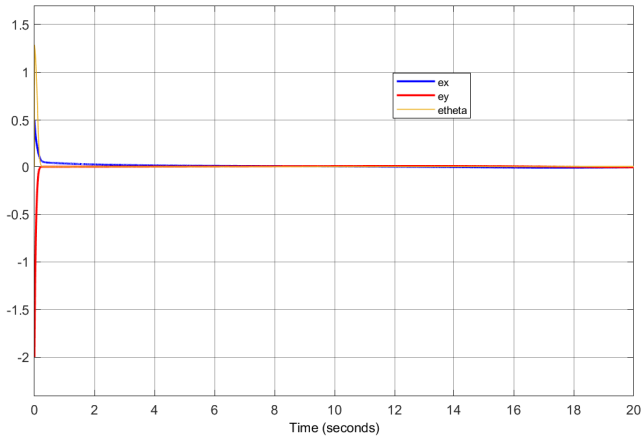
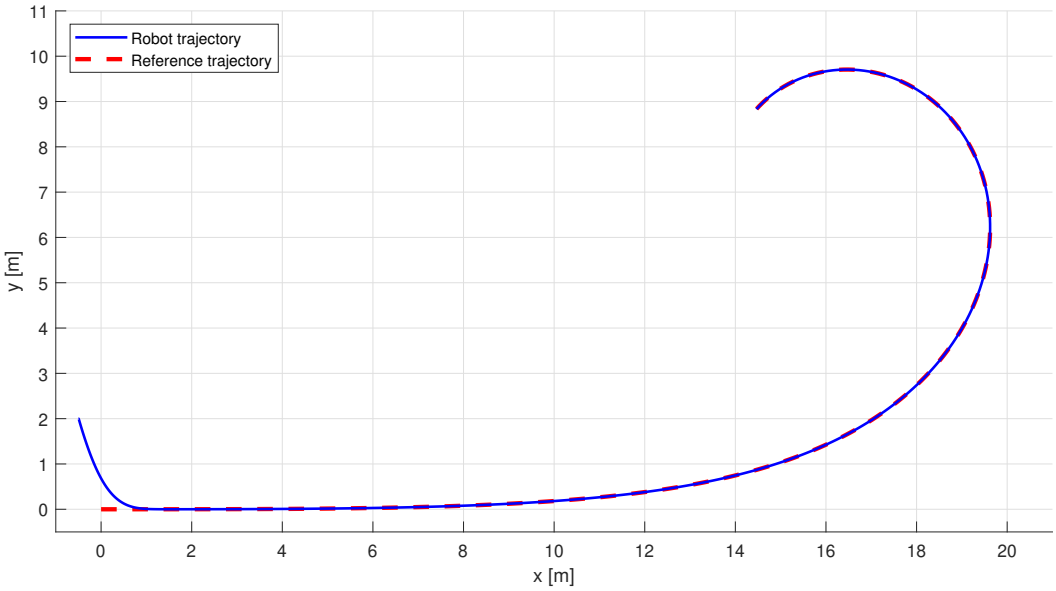
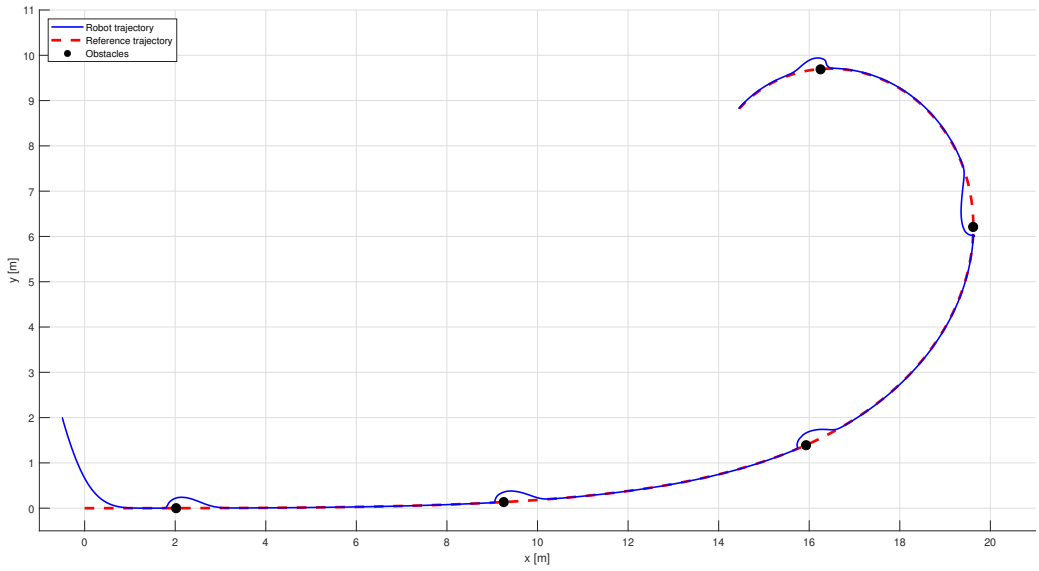


Figure 4.7: Tracking errors



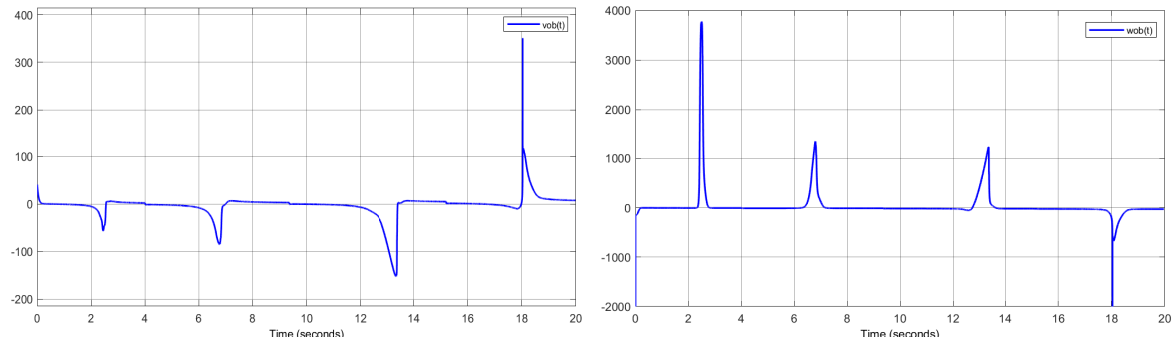
(a) Without obstacles



(b) With obstacles

Figure 4.8: Trajectory tracking

As shown in Figure 4.8a and Figure 4.8b, the trajectory tracking performance is achieved in both scenarios without obstacles and in the presence of multiple obstacles. However, it is important to note that this level of performance is obtained through relatively high gain values, which enforce aggressive corrective actions to minimize the tracking error.



(a) Linear velocity to avoid obstacles (b) Angular velocity to avoid obstacles

Figure 4.9: Velocities for obstacle avoidance

This effect is also reflected in the velocity profiles. As illustrated in Figure 4.9, the control inputs exhibit very high values, which could potentially stress or damage the actuators in a real system. To address this issue, the next chapter will be dedicated to low-level control strategies aimed at constraining the velocities within admissible physical limits, thereby ensuring both system safety and actuator longevity.

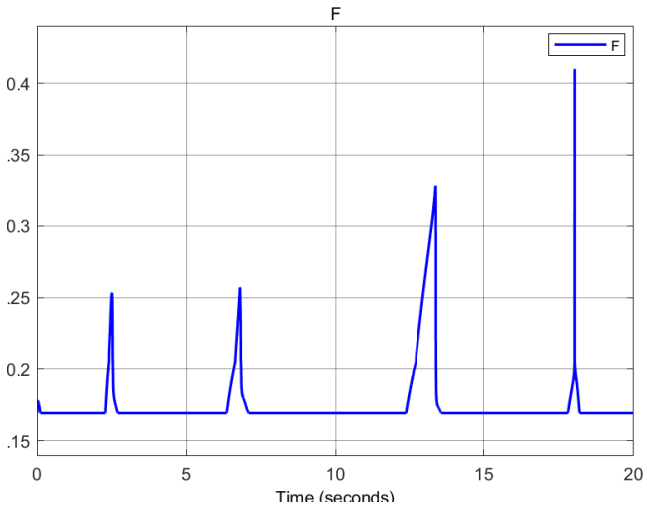


Figure 4.10: Fusion Coefficient

Moreover, Figure 4.10 provides a clear visualization of the moments when the robot encounters obstacles. During these intervals, the fusion coefficient F increases significantly compared to the periods of pure trajectory tracking. This behavior reflects the dynamic switching of control priority, as the robot detects a nearby obstacle, the controller shifts emphasis toward avoidance, increasing the weight of the corresponding control input.

4.7 Extension to Multi-Agent Systems

To extend the proposed control framework, we developed a multi-agent system composed of four mobile robots, one leader and three followers arranged in specific geometric formations relative to the leader. This configuration was chosen for its practical applicability, as it is commonly encountered in real-world scenarios such as coordinated exploration, convoy systems, and formation control. Moreover, the design deliberately minimizes inter-agent communication, thereby reducing system complexity.

4.7.1 Formation Control for the Multi-Agent System

In this configuration, the leader robot navigates according to a predefined reference trajectory. The follower robots use the leader's trajectory as a guide to maintain their respective formations while avoiding collisions. All four robots are equipped with obstacle avoidance capabilities. Importantly, when the leader encounters and avoids an obstacle, the followers do not mimic its actual deviated path; instead, they continue to track the leader's original reference trajectory. This decouples the obstacle response of the followers from the exact motion of the leader, preserving formation coherence and simplifying the control logic.

It is important to note that the control strategy applied to each robot remains identical to the single-agent case. Each robot simply receives its own reference trajectory and follows it independently. No structural modification of the control algorithm is required.

4.7.2 Reference Trajectories for Follower Robots

To maintain the desired formation, the follower robots are assigned reference trajectories derived from the current position and orientation of the leader robot. The relative positioning is based on a geometric configuration where three followers are placed around the leader at predefined offsets.

This leader-follower strategy is a formation control technique, where the spatial arrangement of the agents is maintained through predefined relative positions with respect to the leader.

Let (x_l, y_l, θ_l) represent the current position of the leader robot. The reference positions for the follower robots are computed as:

$$\mathbf{R}(\theta_l) = \begin{bmatrix} \cos\theta_l & -\sin\theta_l \\ \sin\theta_l & \cos\theta_l \end{bmatrix}$$

$$\mathbf{p}_1 = \begin{bmatrix} x_l \\ y_l \end{bmatrix} + \mathbf{R}(\theta_l) \begin{bmatrix} -\frac{d}{\sqrt{2}} \\ \frac{d}{\sqrt{2}} \end{bmatrix}; \quad \mathbf{p}_2 = \begin{bmatrix} x_l \\ y_l \end{bmatrix} + \mathbf{R}(\theta_l) \begin{bmatrix} d \\ 0 \end{bmatrix}; \quad \mathbf{p}_3 = \begin{bmatrix} x_l \\ y_l \end{bmatrix} + \mathbf{R}(\theta_l) \begin{bmatrix} -\frac{d}{\sqrt{2}} \\ -\frac{d}{\sqrt{2}} \end{bmatrix} \tag{4.10}$$

Where: - $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ are the reference positions of followers 1, 2, and 3 respectively, - d is the desired inter-robot distance.

Each follower is assigned the same reference orientation as the leader:

$$\theta_{\text{ref},i} = \theta_l, \quad i \in \{1, 2, 3\}$$

This method allows the follower robots to maintain a triangular formation around the leader 4.11, regardless of the leader's orientation or trajectory. While the current configuration forms a triangle, it is not restrictive other formation shapes can be selected depending on the requirements of the specific task or application. The approach offers flexibility in defining relative positions to suit different mission scenarios.

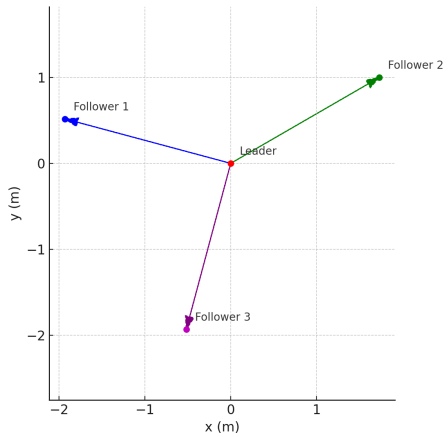


Figure 4.11: followers position relatice to leader

4.7.2.1 Control Implementation

It is important to note that the premise variables used in the fuzzy system are the reference linear and angular velocities. In a multi-agent system with centralized control, it would not be appropriate to assign individual reference velocities to each robot. Instead, it is assumed that all agents share the same reference velocities, as they are intended to exhibit coordinated behavior and follow a common motion pattern.

The effectiveness of the control strategy is clearly demonstrated in Figure 4.12, where all robots successfully follow their respective trajectories. However, it is important to ensure that a minimum inter-robot distance is maintained during trajectory generation in order to prevent potential collisions.

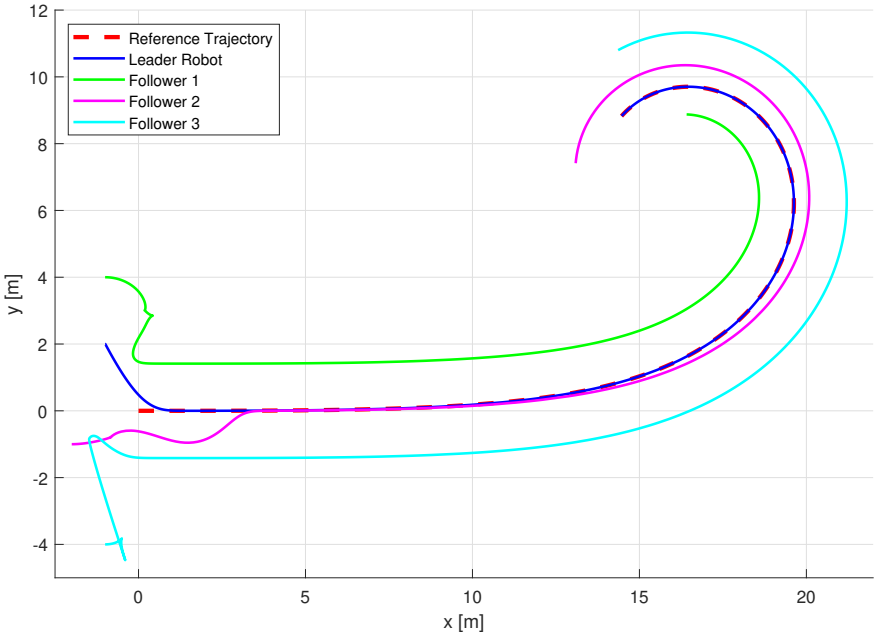


Figure 4.12: Trajectory Tracking and Formation of Multi-Agent System

Figure 4.13 clearly shows that the chosen formation shape is successfully maintained throughout the motion of the entire multi-agent system in the absence of obstacles.

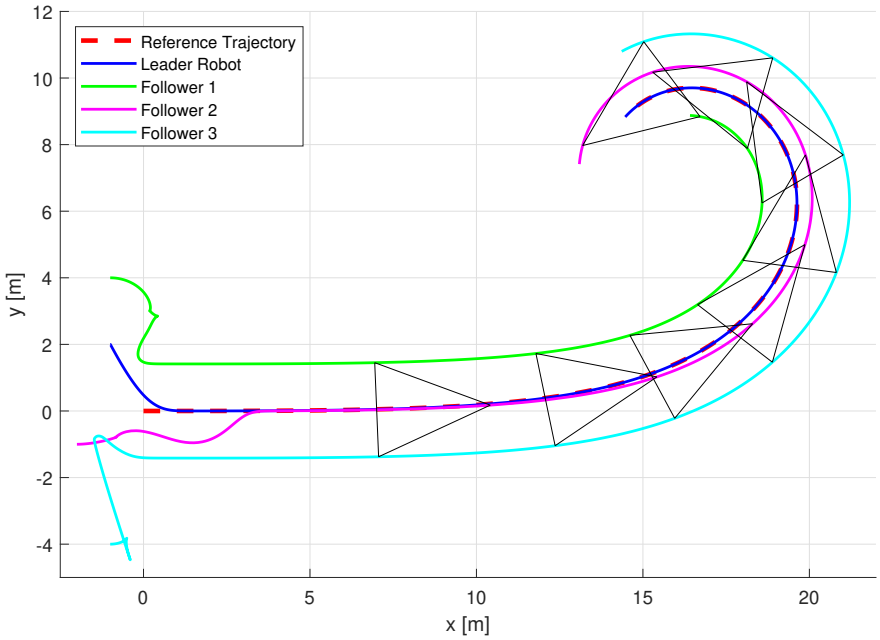


Figure 4.13: Formation control

As shown in Figure 4.14, the robots perform their tasks effectively by following their respective reference trajectories while avoiding obstacles. During the transient phase, the follower robots may exhibit seemingly irregular or dispersed trajectories. This behavior is a result of the fuzzy control strategy, which, despite its apparent variability, remains robust. Importantly, these trajectory deviations do not lead to collisions. In the rare case where robots converge toward the same point, they treat one another as dynamic obstacles. In a real world scenario, one of the robots would actively perform an avoidance maneuver to maintain safety.

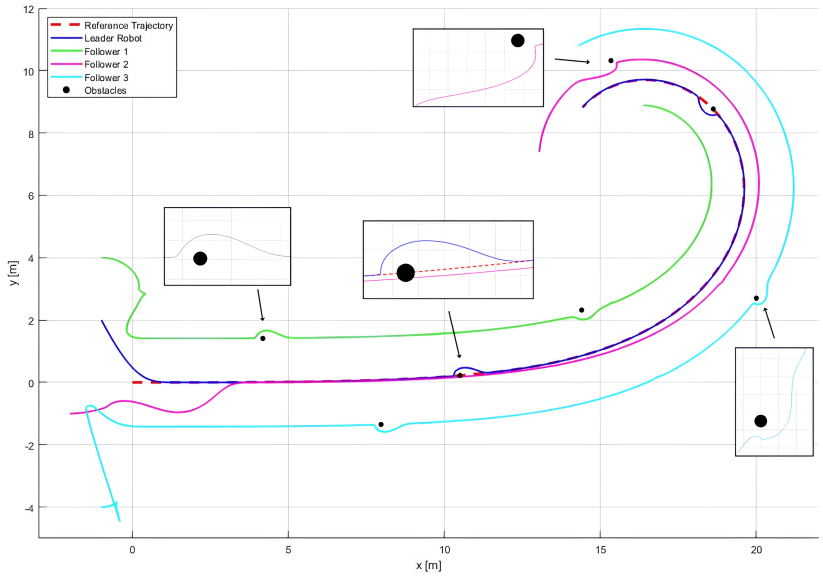


Figure 4.14: Multi-Agent Navigation with Local Obstacle Avoidance and Reference Tracking

Obstacles in the environment can be either static or dynamic. This explains why the leader robot was able to avoid one of the obstacles, while the robot in front of it did not, simply because there was no obstacle present at that moment in its path. However, operating in a dynamic environment requires high performance sensors and a fast computation rate to ensure timely and accurate reactions.

4.8 Fault Estimation for a Single-Agent System

In this section, we address the problem of actuator fault estimation in mobile robot systems. We begin by focusing on the single-agent case, where a dedicated observer is designed to estimate the fault affecting the control inputs. Once validated for the single agent, we extend to the multi-agent system, where each robot is equipped with a local fault estimation mechanism, enabling decentralized diagnosis while preserving the overall coordination.

4.8.1 Application to the Kinematic Model

The theoretical framework for fault estimation in linear systems was established in chapter2.3.3 . In this section, we apply the same approach to estimate an actuator fault acting on the input of a nonlinear system specifically, the kinematic model of a differential drive robot 4.1. To enable this, we assume the presence of a fault at the input of the nonlinear model and derive the corresponding error dynamics. After linearization, it is shown that the fault affecting the original system also appears in the error dynamics like in 4.6. Although there is no strict bijective relationship between the two formulations, estimating the fault within the error model is effectively equivalent to identifying the original fault in the robot's input. This link is clearly illustrated by the transition from Equation 4.11 to Equation 4.12, which validates the applicability of the linear estimation method to the nonlinear case.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v + f_v \\ w + f_\omega \end{bmatrix} \tag{4.11}$$

$$\downarrow\downarrow$$

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta \end{bmatrix} = \begin{bmatrix} 0 & w_r & 0 \\ -w_r & 0 & v_r \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} f_v \\ f_\omega \end{bmatrix} \tag{4.12}$$

With f_v and f_ω are the faults affecting the linear and angular velocity inputs of the system, respectively.

4.8.2 Control law

As planned, the system states and the fault affecting each input will be estimated. The control law will then be generated based on the estimated signals, while preserving the same gains and overall control structure introduced in the previous section as follows :

$$\bar{u}(t) = u(\hat{x}, \hat{y}, \hat{\theta}) - [f_v, f_\theta]^T \tag{4.13}$$

Where $u(t)$ is the control law developed in 4.7 or 4.9, the subtraction of the estimated fault is justified, as it serves as a compensation technique and is an approach commonly used in fault-tolerant control.

4.8.3 Fault Observer Implementation

To evaluate the effectiveness of the iterative fault estimation method, various types of faults will be injected at the input of the kinematic model. This results offers better estimation performance will then be analyzed and discussed.

The injected faults are defined as follows:

$$f_v(t) = \begin{cases} 2(t-5), & \text{if } 5 \leq t \leq 10 \\ 10, & \text{if } t > 10 \\ 0, & \text{otherwise} \end{cases} \quad \text{(linear velocity fault)}$$

$$f_\omega(t) = \begin{cases} 10\, e^{-10(t-7)^2}, & \text{if } 6 \leq t \leq 8 \\ 0, & \text{otherwise} \end{cases} \quad \text{(angular velocity fault)}$$

In practice, the iterative observer offers a more optimal alternative to classical observers. Instead of enhancing the estimation performance by increasing observer gains, which can be costly and difficult to implement in real systems, the iterative approach allows for improved accuracy through repeated estimations using relatively low gain values. This results in better estimation dynamics within a few iterations, without requiring aggressive tuning.

For a performance level defined by an $\mathcal{H}_\infty$ norm bound $\lambda = 0.8$, and pole placement constrained to a region with real parts less than -5, the following observer gains were obtained:

$$L_{k1} = \begin{bmatrix} 74.8667 & 44.3416 & 3.8492 \\ -24.2749 & 10.3336 & -2.0225 \\ -3.8492 & 4.7921 & 74.8667 \end{bmatrix}, \quad L_{f1} = \begin{bmatrix} -811.6830 & -570.3105 & -40.3973 \\ 40.3973 & -50.2931 & -811.6830 \end{bmatrix}$$

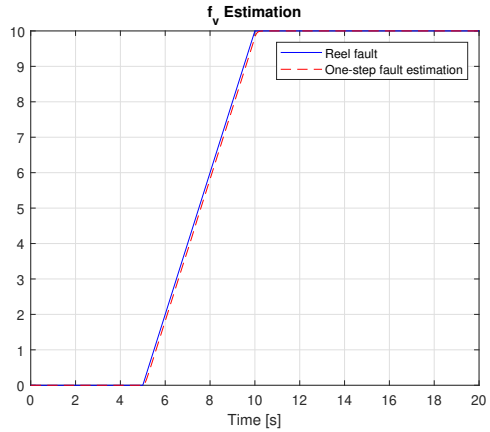
$$L_{k2} = \begin{bmatrix} 74.8667 & -44.2269 & 4.8972 \\ 24.2026 & 10.3336 & -5.2581 \\ -4.8972 & 9.9220 & 74.8667 \end{bmatrix}, \quad L_{f2} = \begin{bmatrix} -811.6830 & 569.1071 & -51.3957 \\ 51.3957 & -104.1300 & -811.6830 \end{bmatrix}$$

$$L_{k3} = \begin{bmatrix} 74.8667 & 45.7031 & 15.8424 \\ -25.1336 & 10.3336 & 6.7218 \\ -15.8424 & -1.1444 & 74.8667 \end{bmatrix}, \quad L_{f3} = \begin{bmatrix} -811.6830 & -584.5994 & -166.2647 \\ 166.2647 & 12.0108 & -811.6830 \end{bmatrix}$$

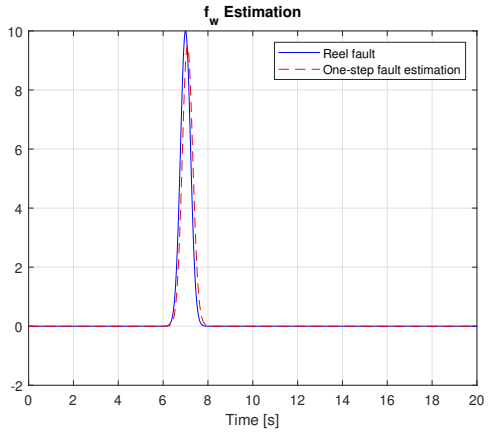
$$L_{k4} = \begin{bmatrix} 74.8667 & -43.8524 & 0.0647 \\ 23.9663 & 10.3336 & 3.8321 \\ -0.0647 & 3.4372 & 74.8667 \end{bmatrix}, \quad L_{f4} = \begin{bmatrix} -811.6830 & 565.1766 & -0.6790 \\ 0.6790 & -36.0732 & -811.6830 \end{bmatrix}$$

As shown in Figures 4.15a and 4.15b, the one-step fault estimation leads to a rough approximation of the fault signals with the following errors : 0.195 for the linear velocity and 2.3 for

the angular velocity. This inaccuracy is more evident in the state estimation results in Figures 4.16a and 4.16c, where large deviations and discontinuities, especially in θ, appear. Such abrupt angular errors can lead to unstable or erratic rotational motion, which may damage the actuators in real world systems.

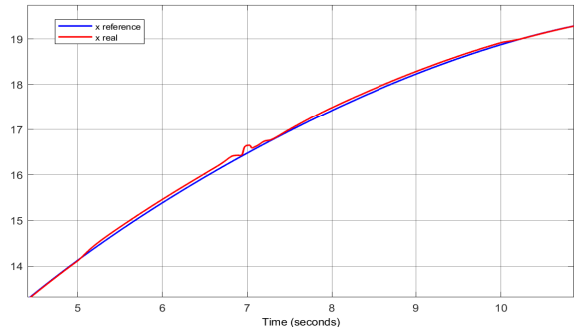


(a) Linear velocity fault estimation

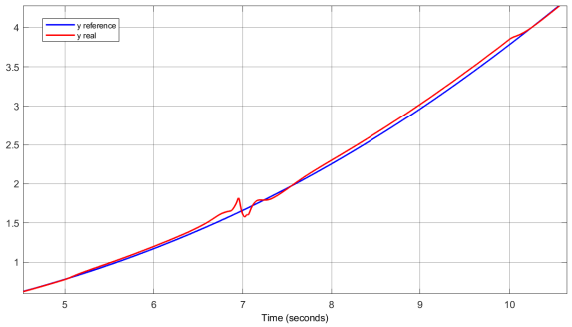


(b) angular velocity fault estimation

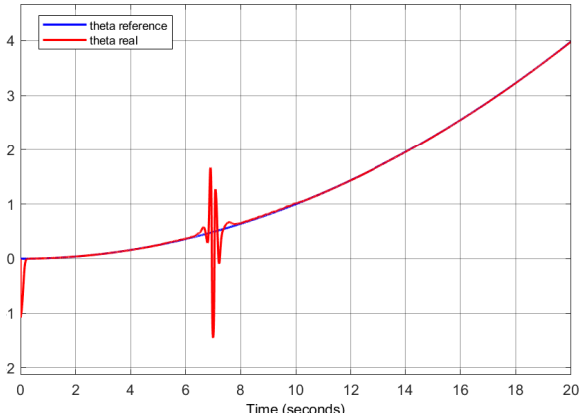
Figure 4.15: One-step fault estimation



(a) Tracking of the state x



(b) Tracking of the state y



(c) Tracking of the state θ

Figure 4.16: Signals evolution under one-step fault conditions and compensation

The experiment injects a sharp, short duration fault to evaluate system sensitivity (Figure 4.17). The system deviates from its reference trajectory but reconverges once the fault disappears. This highlights both the fault impact and the partial ability of the estimator to recover the desired behavior even after temporary disruption.

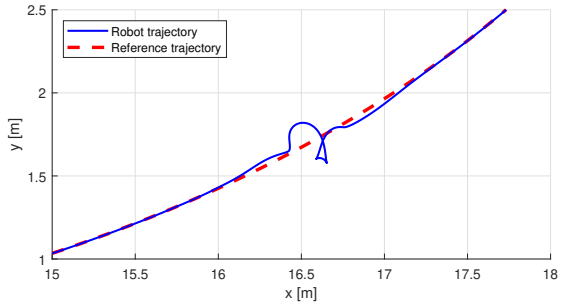
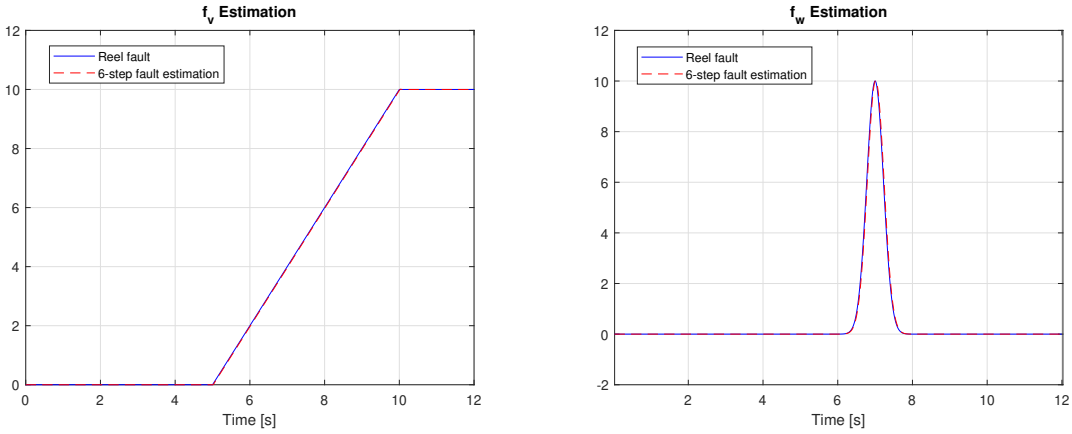


Figure 4.17: Trajectory tracking under one-step fault estimation

In contrast, Figures 4.18a and 4.18b show that after six estimation iterations, the fault is captured accurately with the following errors : 0.038 for the linear velocity fault estimated and 0.4 for the angular one. This leads to improved compensation, enhancing tracking performance and overall system reliability.



(a) Linear velocity fault estimation (b) angular velocity fault estimation

Figure 4.18: 6-step fault estimation

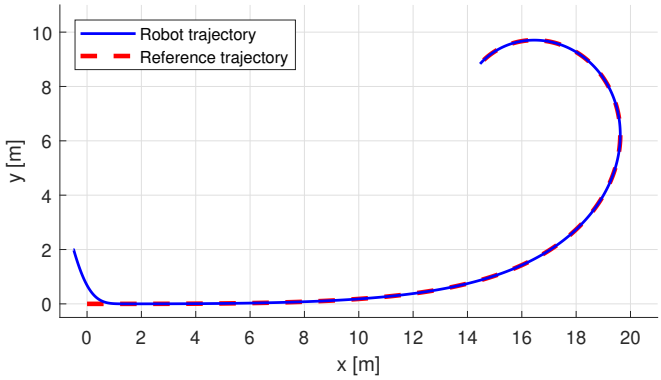
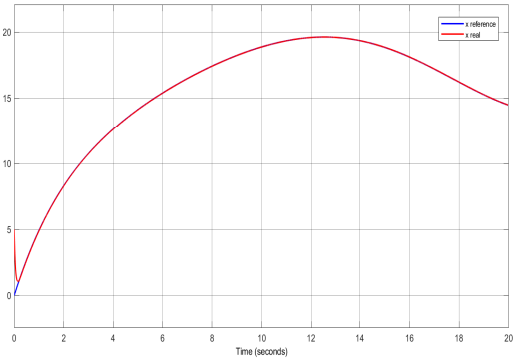
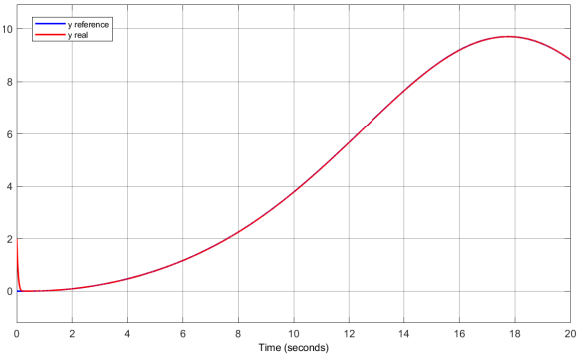


Figure 4.19: Trajectory tracking under 6-step fault estimation

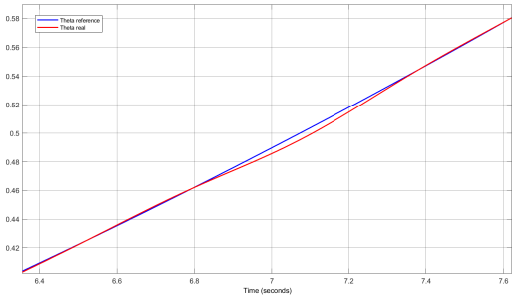
Improving fault estimation accuracy contributes directly to better trajectory tracking performance. This is confirmed by the trajectory shown in Figure 4.19. Although minor discrepancies can still be observed in the state variables x, y, and θ in Figure 4.20, the results remain acceptable for most mobile robotics applications. In such contexts, a trade-off is often made between estimation precision and computational load, and the achieved performance is sufficiently accurate without overburdening the onboard processor with unnecessary computations.



(a) Tracking of the state x



(b) Tracking of the state y



(c) Tracking of the state θ

Figure 4.20: Signals evolution under 6-step fault conditions and compensation

4.9 Distributed Fault Estimation for Multi-Agent Systems

To validate the iterative estimation method in a more realistic setting, we extend our study to the multi-agent case. Although the control strategy for the multi-agent system is not the core focus, it serves as a practical platform to assess the distributed fault estimation approach.

4.9.1 Interaction Structure

The configuration shown in Figure 4.21 illustrates the communication topology between four mobile robots. This structure facilitates direct interaction between the leader robot (node 1) and the follower robots (nodes 2, 3, and 4), enabling efficient coordination and information exchange.

The corresponding Laplacian matrix associated with this topology is given by:

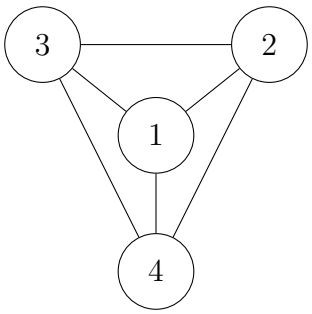


Figure 4.21: Communication topology

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

This symmetric and fully connected structure ensures strong interaction between agents, while keeping the estimation process localized and efficient.

4.9.2 Fault Observer Implementation

The faults injected into each agent are described below, and their time-dependent expressions are provided in the appendix.

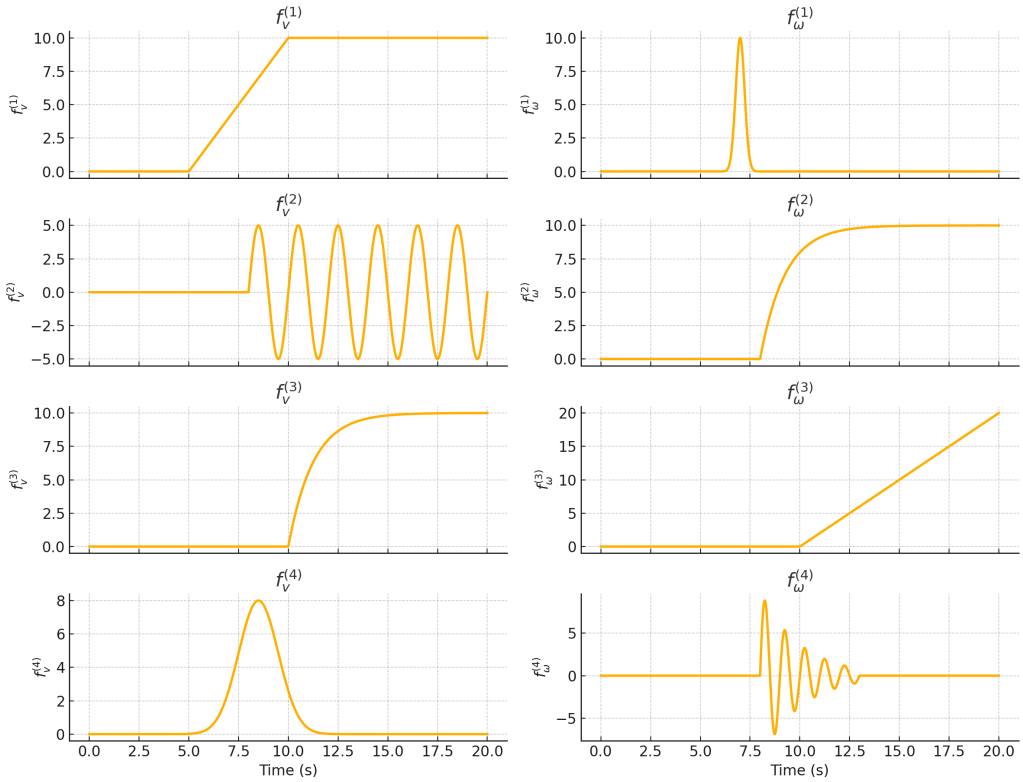


Figure 4.22: individual fault signals injected per agent

In multi-agent implementation, the estimation gains vary significantly across agents and subsystems. Some gains are extremely high, while others are very small, which imposes strong requirements on the computational capacity and stability of the onboard controllers.

$$H_1 = 10^{-11} \cdot \begin{bmatrix} 0.0087 & -0.0237 & 0.5249 \\ 0.0025 & 0.0180 & -0.0035 \\ 0.5174 & -0.1033 & -0.0540 \end{bmatrix} \quad G_1 = \begin{bmatrix} 82.2821 & 37.3527 & -52.4100 \\ -10.9790 & 0.7587 & -0.0991 \\ 52.4100 & 3.8861 & 82.2821 \end{bmatrix}$$

$$F_{11} = 10^{-10} \cdot \begin{bmatrix} -0.0171 & 0.0353 & -0.5209 \\ -0.5115 & 0.1344 & 0.0721 \end{bmatrix} \quad F_{21} = 10^{3} \cdot \begin{bmatrix} -1.0732 & -0.5692 & 0.5202 \\ -0.5202 & -0.0386 & -1.0732 \end{bmatrix}$$

$$H_2 = 10^{-11} \cdot \begin{bmatrix} 0.0193 & -0.0159 & -0.0541 \\ 0.0016 & -0.0143 & -0.0046 \\ -0.0406 & -0.1169 & -0.0281 \end{bmatrix} \quad G_2 = \begin{bmatrix} 82.2821 & -38.3113 & -50.0538 \\ 11.4968 & 0.7587 & -1.4914 \\ 50.0538 & 6.4638 & 82.2821 \end{bmatrix}$$

$$F_{12} = 10^{-10} \cdot \begin{bmatrix} -0.0152 & 0.0215 & 0.0541 \\ 0.0372 & 0.1524 & 0.0277 \end{bmatrix} \quad F_{22} = 10^{3} \cdot \begin{bmatrix} -1.0732 & 0.5788 & 0.4968 \\ -0.4968 & -0.0642 & -1.0732 \end{bmatrix}$$

$$H_3 = 10^{-11} \cdot \begin{bmatrix} -0.0257 & 0.0053 & -0.0240 \\ -0.0014 & 0.0255 & -0.0005 \\ -0.0344 & -0.2148 & -0.2206 \end{bmatrix} \quad G_3 = \begin{bmatrix} 82.2821 & 37.6191 & 70.7288 \\ -11.1229 & 0.7587 & 0.9793 \\ -70.7288 & 20.4030 & 82.2821 \end{bmatrix}$$

$$F_{13} = 10^{-10} \cdot \begin{bmatrix} 0.0271 & -0.0041 & 0.0236 \\ 0.0358 & 0.2825 & 0.2260 \end{bmatrix} \quad F_{23} = 10^{3} \cdot \begin{bmatrix} -1.0732 & -0.5719 & -0.7020 \\ 0.7020 & -0.2025 & -1.0732 \end{bmatrix}$$

$$H_4 = 10^{-11} \cdot \begin{bmatrix} 0.0675 & -0.0168 & 0.1464 \\ -0.0042 & -0.0214 & 0.0018 \\ 0.1595 & -0.1527 & -0.2045 \end{bmatrix} \quad G_4 = \begin{bmatrix} 82.2821 & -39.2647 & -16.5412 \\ 12.0118 & 0.7587 & 7.4649 \\ 16.5412 & 8.3959 & 82.2821 \end{bmatrix}$$

$$F_{14} = 10^{-10} \cdot \begin{bmatrix} -0.0715 & 0.0232 & -0.1428 \\ -0.1597 & 0.2010 & 0.1989 \end{bmatrix} \quad F_{24} = 10^{3} \cdot \begin{bmatrix} -1.0732 & 0.5882 & 0.1642 \\ -0.1642 & -0.0833 & -1.0732 \end{bmatrix}$$

As shown in Figures 4.23 and 4.24, applying one-step fault estimation in multi-agent systems produces limited accuracy. While it provides a rough fault estimate useful for general awareness or triggering fault alarms, it is not suitable for high-precision control applications.

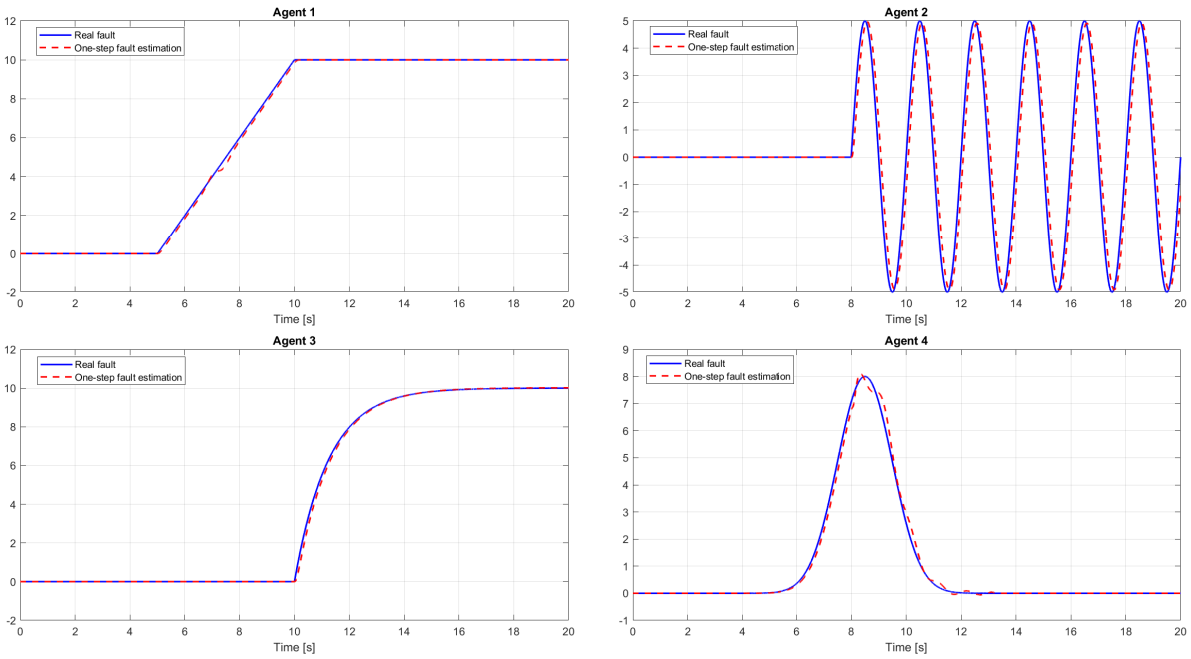


Figure 4.23: One-step fault estimation of linear velocity input

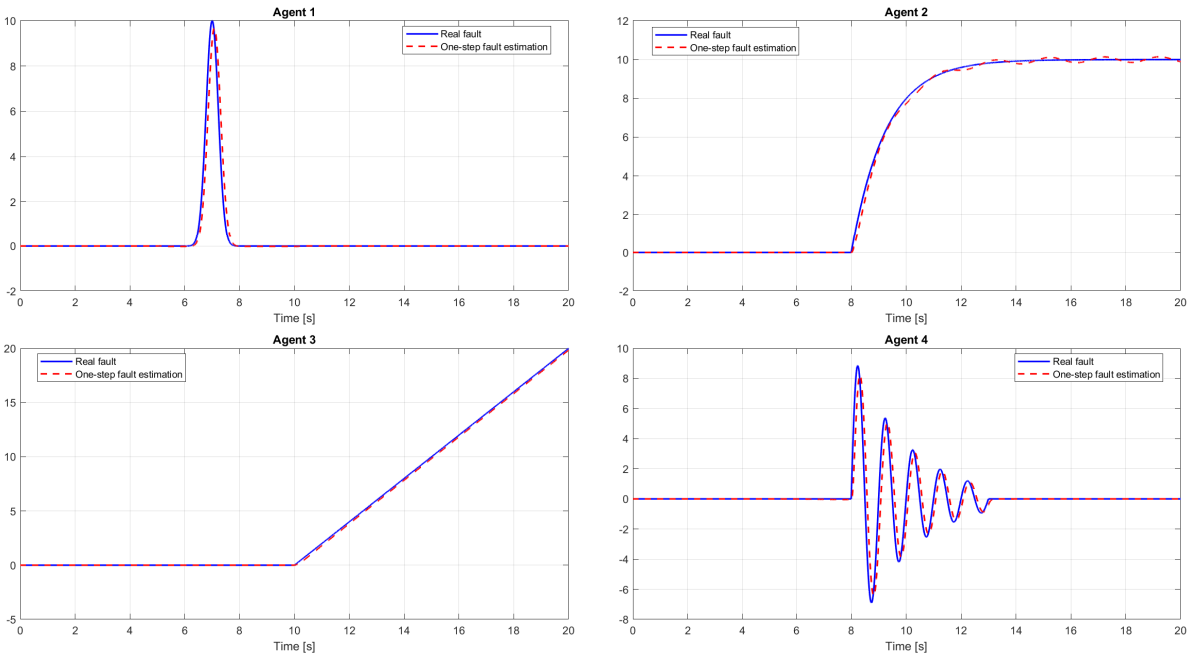


Figure 4.24: One-step fault estimation of angular velocity input

The results in Figures 4.25 and 4.26 demonstrate that after five iterations, the estimation satisfies a predefined convergence criterion based on fault change tolerances:

$$\texttt{tolerance1-8} = \{0.1, 0.5, 0.5, 0.2, 0.2, 0.1, 0.3, 1.5\}$$

These thresholds ensure that the difference between fault estimates at iteration k and $k-1$ is within acceptable bounds. A fixed $\mathcal{H}_\infty$ performance index $\gamma = 0.8$ was used for all observers.

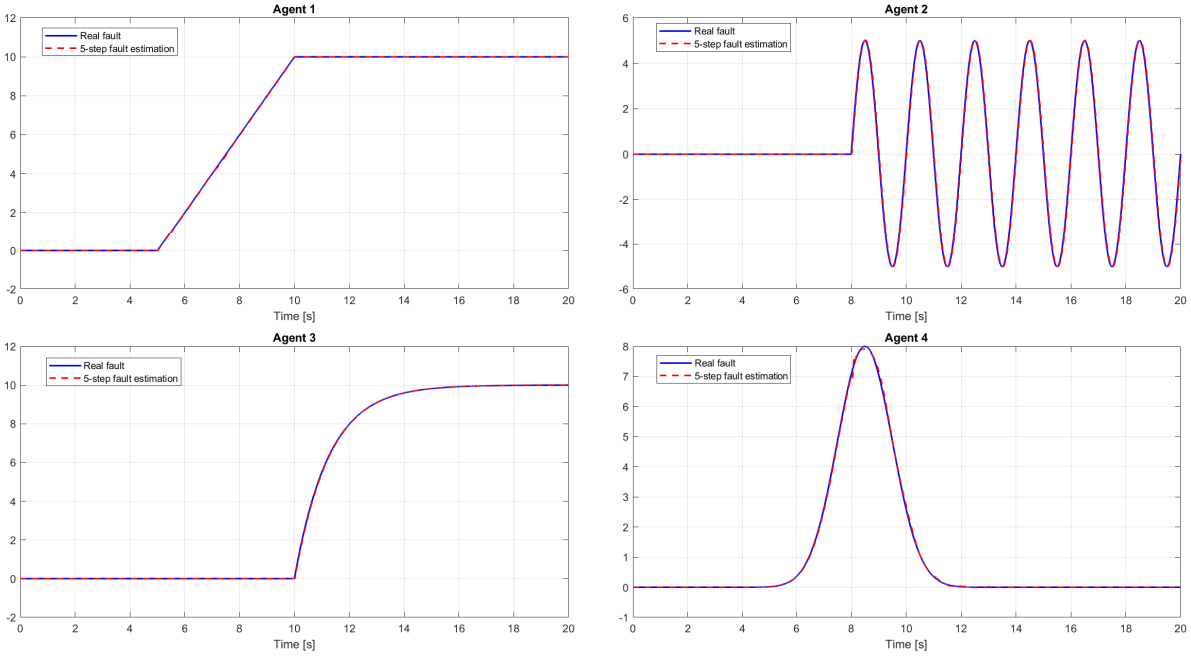


Figure 4.25: 5-step fault estimation of linear velocity input

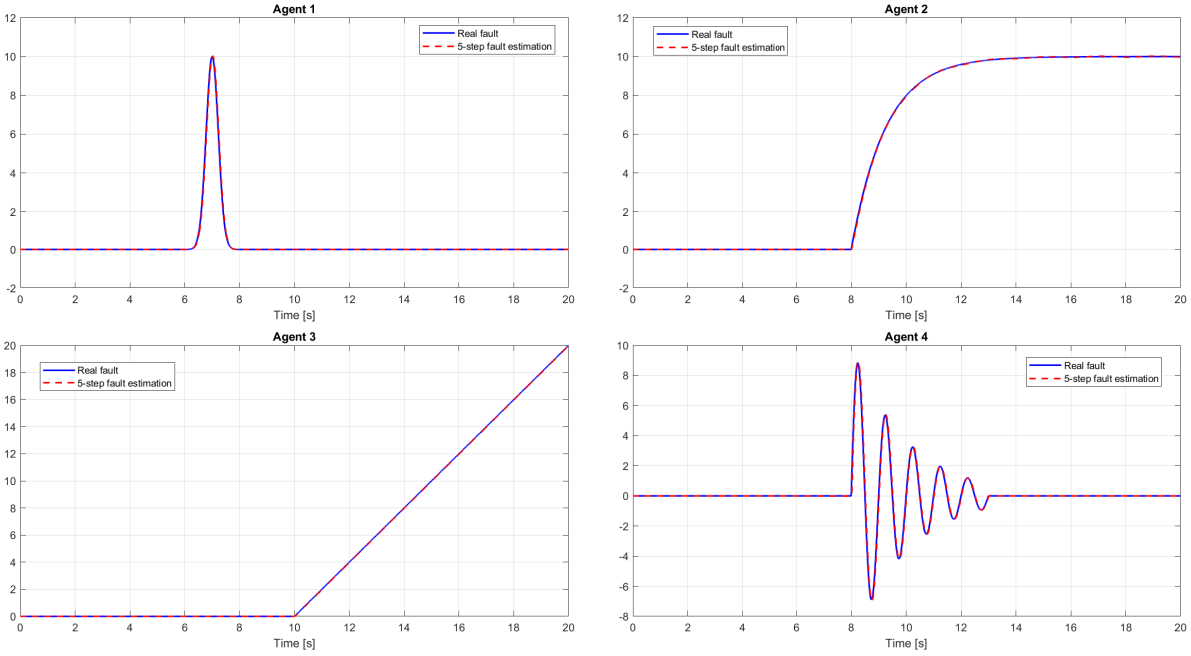


Figure 4.26: 5-step fault estimation of linear velocity input

As illustrated in Figure 4.27, trajectories of robots driven using only one-step fault estimates exhibit large deviations. In contrast, when faults are iteratively estimated (up to the 5th iteration), the corrected trajectories closely follow the desired paths, validating the superiority of multi-step fault estimation in coordinated multi-agent navigation.

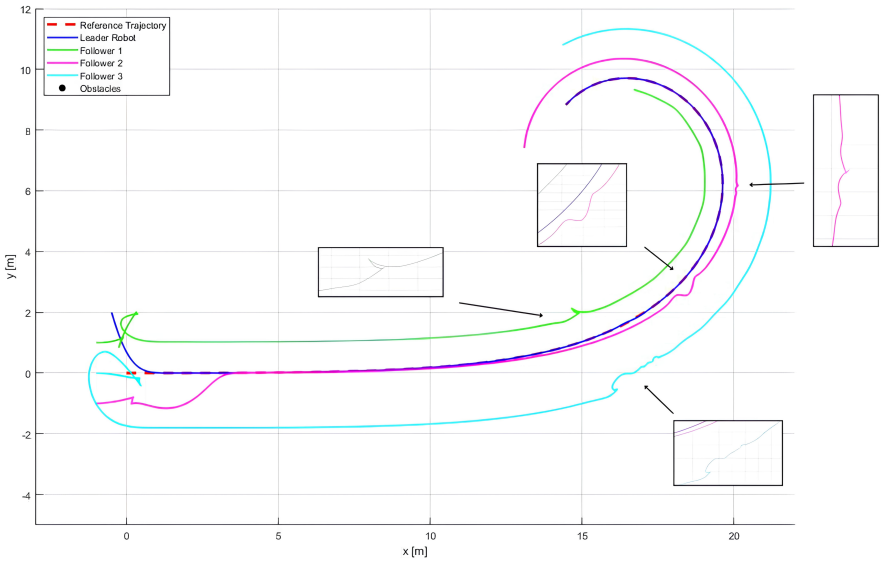


Figure 4.27: Trajectory evolution under one-step fault estimation

The estimation errors after six iterations are summarized below. The values correspond respectively to the linear and angular fault components (f_v, f_ω) for Robots 1 to 4:

- Robot 1: $f_v = 0.087101, \quad f_\omega = 0.448058$
- Robot 2: $f_v = 0.384276, \quad f_\omega = 0.189692$
- Robot 3: $f_v = 0.186134, \quad f_\omega = 0.069124$
- Robot 4: $f_v = 0.262504, \quad f_\omega = 1.395921$

We can now replot the trajectory to verify whether the robot accurately follows it. The figure 4.28 illustrates the result.

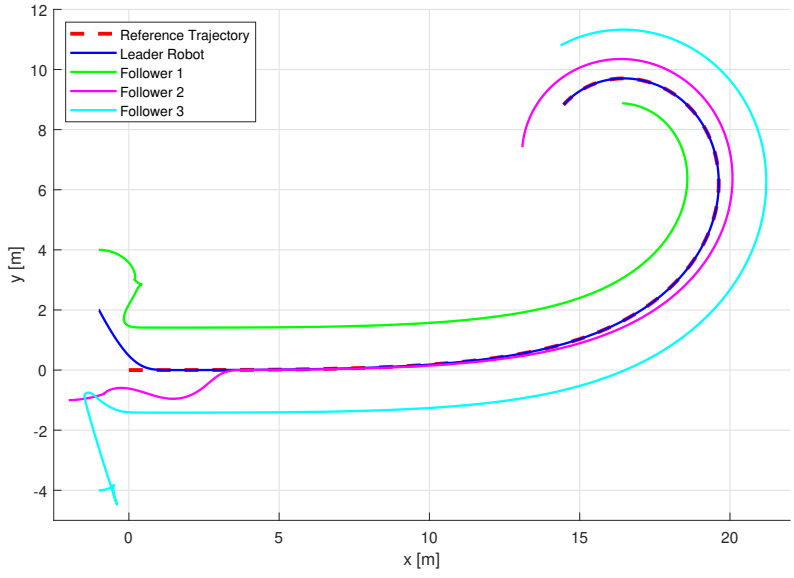


Figure 4.28: Trajectory evolution under 5-step fault estimation

Conclusion

This chapter has presented the implementation of a fault-tolerant control strategy for differential-drive mobile robots, addressing both single-agent and multi-agent configurations. Through a combination of fuzzy control, feedforward compensation, and iterative fault estimation, the proposed method effectively handles actuator faults while preserving trajectory tracking and obstacle avoidance capabilities.

In the single-agent scenario, the limitations of one-step fault estimation were identified, particularly in the presence of abrupt disturbances. The iterative observer demonstrated significantly improved estimation accuracy, enabling more reliable compensation and stable motion control.

The extension to a multi-agent system showcased the robustness and scalability of the approach. Despite the challenges posed by inter-agent dynamics and varying fault profiles, the distributed estimation strategy maintained satisfactory performance. Simulations confirmed that accurate fault identification after multiple iterations leads to enhanced coordination and safer navigation across all agents.

Building upon these results, the next step involves transitioning to realistic simulation environments to evaluate the approach under near real-world conditions. This will serve to experimentally validate the proposed framework and pave the way for practical deployment in robotic platforms.

Chapter 5

Implementation in ROS

Introduction

While previous chapters focused on the theoretical formulation and MATLAB-based validation of the proposed fault-tolerant control strategy, it is essential to assess its feasibility in real-time robotic applications. Therefore, this chapter is dedicated to implementing the iterative fault estimation observer in a practical setup using the Robot Operating System (ROS) framework and the Gazebo simulation environment. Although the tests are conducted in a virtual world, the setup closely mimics real-world conditions, including realistic sensors, dynamics, and inter-agent communication. This simulation-based deployment bridges the gap between theoretical modeling and real-world robotics.

The main objective is to validate whether the control and estimation strategies developed earlier can perform reliably under time and computation constraints. Special attention is paid to the integration of the control loop, sensor feedback, actuator commands, and inter-robot coordination in multi-agent scenarios. The implementation also explores key aspects such as trajectory tracking, obstacle avoidance, and fault compensation in a unified ROS architecture.

This chapter is structured to provide both single-agent and multi-agent validation results, highlighting the robustness and scalability of the proposed strategy. Detailed figures and signal plots are used to illustrate the effectiveness of the estimator under various fault scenarios and dynamic environments. The chapter concludes with an evaluation of performance metrics and a discussion of limitations encountered in practice.

5.1 Robot Operating System

Robot Operating System (ROS) is a widely adopted open-source framework designed to support the development, simulation, and deployment of robotic systems. Rather than being a traditional operating system, ROS provides a flexible middleware layer composed of libraries, tools, and communication protocols that simplify complex robotic programming tasks. It enables modular design by allowing different components of a robot—sensors, actuators, controllers—to communicate efficiently through a publish/subscribe messaging architecture.

ROS offers essential features such as hardware abstraction, low-level device control, inter-process communication, and tools for visualization and debugging. Its architecture supports distributed computation across multiple machines or robots, making it particularly suitable for

collaborative multi-agent systems.

Key advantages of ROS include:

- Seamless integration with various robot-specific platforms and embedded systems.
- Robust tools for visualization, simulation, and debugging (e.g., `RViz`, `Gazebo`).
- Support for multiple programming languages, primarily `C++` and `Python`.
- Extensive open-source package ecosystem for perception, navigation, control, and more.
- Active global community and strong support for research and industrial applications.

Thanks to its versatility and community-driven development, ROS has become the standard for robotic system design in both academic and industrial contexts.

5.1.1 ROS architecture

The architecture of ROS is based on a modular and distributed system where different components, called nodes, perform specific tasks and communicate with each other through topics using a publish/subscribe model. The ROS Master acts as a central coordinator, managing the registration of nodes and enabling them to locate and exchange information. Data exchanged between nodes is structured in predefined messages, which can contain various types of information such as sensor readings, control commands, or status updates.

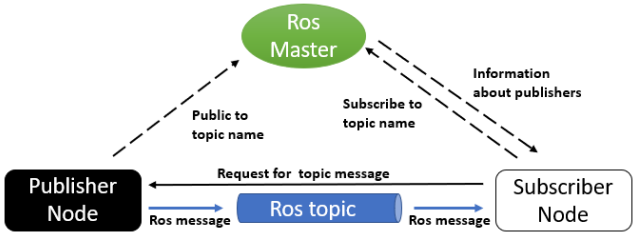


Figure 5.1: ROS Master Communication

To launch these components, ROS provides the `roscore` command for ROS1 to start the Master and essential services, and the `rosrun` or `roslaunch` commands to execute individual nodes or entire robotic systems defined in launch files. This architecture enables scalable, flexible, and reusable robotic software development.

5.1.1.1 ROS workspace

A ROS workspace is a directory structure where various ROS packages can be developed, built, and managed in an organized way. It simplifies the separation of multiple projects and their dependencies. The following steps describe how to create and configure a ROS workspace:

1. Create a workspace directory and a subdirectory for source files:

```
$ mkdir -p ~/workspace_name/src
$ cd ~/workspace_name/
```

2. Initialize the workspace using `catkin_make`, which will generate the necessary build and setup files:

   ```
   $ catkin_make
   ```

3. Source the setup file to load the environment configuration. This command must be executed every time a new terminal is opened:

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

After building the workspace, the following directory structure is created:

```
workspace_name/
|-- build
|-- devel
|-- src
```

- The `src` directory contains the source code of ROS packages.
- The `build` directory stores temporary build files.
- The `devel` directory holds development environment setup files.
- The `logs` directory keeps log files generated during the build process.

The following illustrates the structure of our workspace for this project. Each component will be described in the subsequent sections.

```
pfe/
|-- build
|-- devel
|-- src/
    |-- CMakeLists.txt
    |-- fuzzy_tracker
    |   |-- CMakeLists.txt
    |   |-- package.xml
    |   |-- launch
    |   |-- scripts
    |-- my_robot
        |-- CMakeLists.txt
        |-- package.xml
        |-- launch
        |-- rviz
        |-- urdf
        |-- world
```

5.1.1.2 ROS packages

To create a new package within a ROS workspace, the following steps should be executed in the terminal:

```
$ cd ws_name
$ cd src
$ catkin_create_pkg pkg_name rospy roscpp std_msgs   # dependencies
$ cd ..
$ catkin_make
```

In ROS, a package is the basic unit of software structure. It typically contains everything required for a specific functionality: nodes (executable files), libraries, configuration files, launch scripts, and dependencies. Organizing code into packages promotes modularity, reusability, and clarity across projects.

To be recognized as a valid `catkin` package and be usable in ROS, the following conditions must be met:

- A `package.xml` file must be present, containing meta-information such as the package name, version, dependencies, and maintainer details.
- A `CMakeLists.txt` file must be included to define the build process via `catkin`.
- Each package must reside in a separate directory; placing multiple packages in the same folder is not permitted.

5.1.2 GUI Tools

In addition to command-line tools, ROS provides powerful GUI tools that enhance development, visualization, and debugging.

- RViz : is a 3D visualization tool used in ROS to display sensor data, robot models, and simulation results in a virtual environment. It provides developers with an intuitive way to observe the robot's perception, localization, and planning in real-time, making it essential for system validation and testing.
- RQT : is a plugin-based GUI framework that offers a variety of tools for monitoring and controlling ROS-based systems. It includes utilities for debugging, visualizing data, plotting variables, and adjusting parameters dynamically. Thanks to its modular design, RQT can be extended with custom plugins to meet specific development needs.
- Gazebo : Gazebo is a powerful robotics simulator that offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. It provides the following features:
    1. High-fidelity Physics: Simulates the dynamics of robots and environments with high accuracy, including rigid body physics, contact, and sensor data.
    2. Sensor Simulation: Includes support for a variety of sensors, such as cameras, LIDAR, IMUs, and GPS.
    3. Extensible and Flexible: Integrates with various robotics frameworks.

4. 3D Visualization: Offers 3D visualization tools for designing and debugging complex robotic systems.

Using Gazebo, developers can test and refine their robotic algorithms in a controlled, virtual environment before deploying them to real hardware, thus reducing risk and accelerating development.

5.2 Virtual Robotic Modeling and Simulation Environments

In robotics, creating a realistic virtual environment is essential for validating algorithms and behaviors before deployment in the real world. Simulation environments like Gazebo provide a powerful interface for modeling physical interaction, sensor behavior, and motion planning in a way that closely mimics real-world conditions. A crucial step in this process is defining the robot's structure and dynamics using formats like URDF and integrating it within the simulated world.

5.2.1 URDF Modeling of the Differential Mobile Robot

The Unified Robot Description Format (URDF) is an XML-based format used to describe a robot's physical structure, including its links (rigid bodies), joints (connections), inertial properties, visuals, and collision geometry. To facilitate scalability and reusability, URDF files are often written using Xacro files, which allows the inclusion of variables, macros, and conditional statements.

Each element of the robot is defined in terms of geometric primitives (box, cylinder, sphere, mesh), their positions, and orientations. The URDF file also contains information about the robot's sensors, actuators, and optionally transmission interfaces when used with ROS Control. Once the URDF/Xacro model is complete, it can be loaded into ROS via launch files, published on the `/robot_description` parameter server, and instantiated in Gazebo using the `spawn_model` node. This enables simulation of both the robot's dynamics and sensor feedback in a 3D physics-based environment.

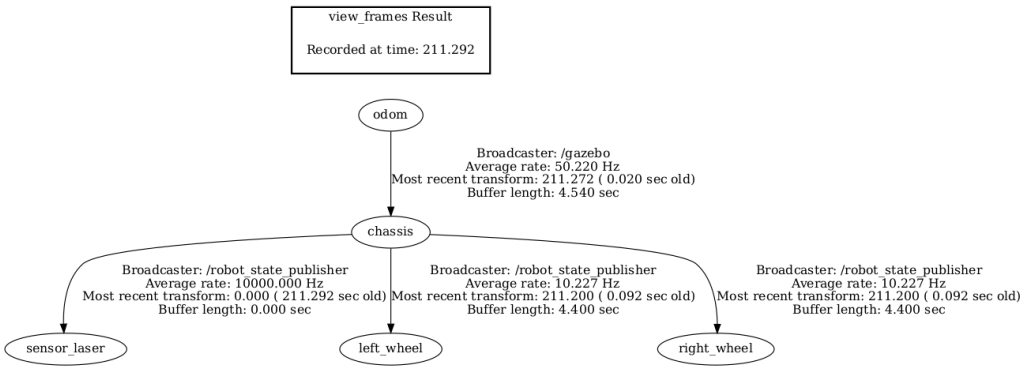


Figure 5.2: Transforms tree of a single robot

5.2.1.1 Geometric Modeling and Robot Description

In our project, we developed a differential drive mobile robot using Xacro. The robot consists of a main chassis, two driving wheels, and tow caster wheel. Each component is modeled with specific dimensions, mass, inertial properties, and visual representations. The coordinate frames of each link are clearly defined and connected through appropriate revolute and fixed joints. This structured description allows accurate physical and kinematic simulation in Gazebo.

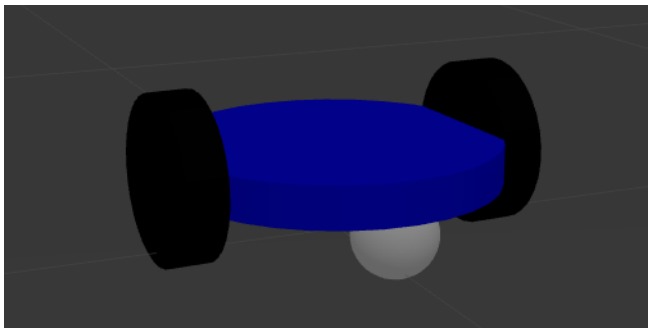


Figure 5.3: Robot Shape

5.2.1.2 Sensor Integration

The robot is equipped with simulated sensors including a LIDAR scanner and wheel odometry. The LIDAR is defined in the URDF using the $gazebo_ros_laser$ plugin, allowing the simulation of 2D range data for environment perception and obstacle detection. Wheel odometry is obtained through the differential drive motion model and is essential for estimating the robot's relative position over time.

This LIDAR is used for obstacle detection, and due to its measurement accuracy, it allows estimating the relative position of obstacles with respect to the robot. Given the reliable odometry, the acquired laser data can be confidently used for mapping and navigation tasks.

In the Gazebo configuration file, the sensor's scanning range was limited to an angular field of view from $-\frac{\pi}{2}$ to $+\frac{\pi}{2}$, which is sufficient for the robot to sweep the 10 meter area in front of it.

Unlike IMU based systems, our odometry relies on wheel encoders, which are simulated through Gazebo's differential drive plugin `gazebo_ros_diff_drive`. This plugin not only provides velocity commands to the wheels but also publishes odometry information on standard ROS topics.

Figure 5.4: Hokuyo UST-05LA 2D LiDAR sensor

This setup is critical for enabling basic localization, navigation, and closed-loop control in the simulated environment.

From these odometry messages, the robot's planar position (x, y) and its heading angle θ are derived. The position is directly extracted from the `pose.pose.position` field of the message. However, the orientation is initially provided as a quaternion (x, y, z, w), which is a four-dimensional representation of rotation in 3D space.

To interpret this orientation in 2D navigation, the quaternion is converted into Euler angles (roll, pitch, yaw) using a mathematical transformation. The yaw angle, corresponding to rotation around the vertical axis, is used as θ, representing the robot's orientation in the plane. This transformation enables real-time tracking of the robot's pose and is essential for tasks such as trajectory following, localization, and sensor alignment.

5.2.1.3 Deployment of the Multi Robot System

To simulate a multi-robot system using a single URDF/Xacro file, we leverage the concept of namespace prefixes. Each instance of the robot is assigned a unique prefix during launch, which ensures that all links, joints, and topics are properly namespaced and avoid conflicts. This approach allows multiple robots to coexist in the same simulation environment with isolated coordinate frames.

To facilitate coordination and visualization, we define static transforms between each robot's base frame and a common world or map frame. This global referencing is essential for tasks such as centralized monitoring, inter-robot communication, and multi-agent coordination.

We launch a Gazebo simulation using an empty world, to which we add both pre-defined and custom models. These include static and dynamic objects representing obstacles, urban elements, and natural features. Furthermore, realistic environmental elements such as terrain friction, lighting (sun), and atmospheric conditions can be configured to enhance realism and test the robustness of our algorithms in near real-world conditions.

5.2.2 Real Time Visualization

To ensure effective visualization and monitoring, reference trajectories are published for each robot to follow. Obstacle properties, such as position, diameter, and height, are also specified to match their visual and collision models, ensuring consistency between what is seen and what is simulated in terms of physics.

Careful attention is paid to the frame hierarchy of each robot. All links must be correctly connected and associated with their respective transforms to ensure proper rendering and interpretation in tools like RViz. Proper setup of the TF tree allows intuitive real-time tracking of robot states, sensor frames, and overall spatial coherence.

In addition, we employ visualization markers to indicate targets, paths, and sensor fields in RViz published in the global frame `odom`, aiding in debugging and performance evaluation. Synchronizing Gazebo and RViz via consistent topic and frame definitions is critical to maintaining a reliable and interpretable simulation environment.

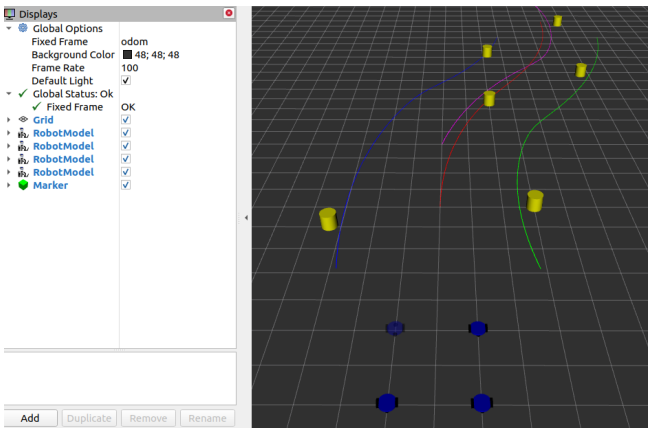


Figure 5.5: RViz Display Overview

5.3 Implementation of the control law

5.3.1 Control Strategy

In this section, we consider the case of a single-agent system. The control law structure developed in Chapter 4, based on fuzzy fault-tolerant control using trajectory-following logic, is now implemented in the ROS environment.

The central component of the implementation is a node named `controller`, written in Python. This node is responsible for receiving measurement data from the environment. It subscribes to the following topics:

- `/scan` (or `laser_scan`) – of type `sensor_msgs/LaserScan`, used for obstacle detection.
- `/odom` – of type `nav_msgs/Odometry`, providing the robot's position and orientation.

Based on these inputs, the `controller` node computes the control law to make the robot follow the reference trajectory. The calculated linear and angular velocities are published to the topic `/cmd_vel` using messages of type `geometry_msgs/Twist`. These messages are then transmitted to the robot model in Gazebo through an appropriate ROS plugin.

Another important node is `trajectory_publisher`, which publishes the reference trajectory and obstacle positions to the environment. This ensures that the visual world in Gazebo is consistent with the one used in control.

For the multi-agent case, the `controller` node is extended using ROS namespaces. Each robot has its own controller in the node `controller`, subscribing and publishing on topics specific to that robot. The logic remains the same as described in Chapter 4, including the generation of reference trajectories and movement strategies.

A key practical difference from the MATLAB simulation is how the trajectory is followed. In practice, the trajectory is divided into a series of steps, avoiding excessively dense sampling due to limitations in measurement precision. The robot selects the nearest reachable point from its current position and applies the fuzzy control law to reach it. Once the point is reached within a position and orientation tolerance, the trajectory index is incremented, and the process continues until the end of the path is reached. This tolerance provides flexibility and ensures smooth advancement without excessive computational load.

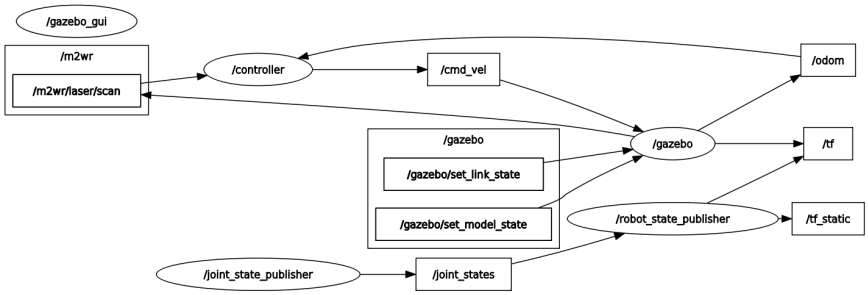


Figure 5.6: Communication Flow for a single agent system

Although ROS provides powerful tools for obstacle avoidance—such as the `move_base` package, which integrates a variety of global and local planning algorithms—we chose not to use these built-in solutions in our implementation. Instead, we applied the custom trajectory planner developed in the previous chapter in order to evaluate its feasibility and performance in a real-time simulation context.

5.3.2 Real-Time Command Synchronization

In ROS, real-time command synchronization is crucial to prevent divergence between the robot's motion and the reference. The function `rospy.get_time()` is used to align the execution of control commands with the reference signals, which are time-dependent. To ensure proper synchronization, the simulation starts only after the robot is fully initialized and ready to receive commands.

The control loop is executed at a frequency of 10 Hz. This update rate allows follower robots in a multi-agent system to obtain the leader's data and compute their control signals in a single iteration, avoiding latency or delays between agents.

5.3.3 Overcoming Dynamic Modeling

To avoid relying on the full dynamic model of the robot, two practical solutions are employed:

1. Using the `differential_drive_controller` plugin: This plugin allows control based solely on the robot's kinematic model. It receives velocity commands via the `/cmd_vel` topic and converts them into wheel velocities, applied directly to the joints. By reducing the robot's mass and inertia in the URDF, dynamic effects become negligible, simplifying implementation while preserving realistic motion.

2. Using recorded reference signals: In this approach, position and velocity data are recorded from a reference robot moving with predefined velocities. These signals are then reused as reference inputs in the control node, allowing the controller to replicate the desired behavior without requiring a dynamic model.

In our implementation, we used the first method with reduced dynamic parameters in the URDF to facilitate controller deployment.

5.3.4 Control Feasibility and Actuator Protection

To ensure actuator feasibility and safety, several measures were adopted. First, by selecting the closest trajectory point at each iteration, we prevent excessive control demands and maintain commands within the actuator's range.

In addition, we imposed explicit velocity limits:

- Linear velocity: limited between 0 and 6 m/s.
- Angular velocity: limited between -1.5 rad/s and 1.5 rad/s.

These limits help reduce oscillations during transient phases while respecting the robot's non-holonomic constraints.

Furthermore, the control gains were scaled by a factor of 0.01. This scaling reduces the risk of saturating the actuators and prevents the robot from deviating significantly from its reference trajectory, enhancing overall stability and robustness of the implementation.

5.4 Real-Time Fault Estimation and Detection

5.4.1 Real Faults in Mobile Robots

In real-world scenarios, mobile robots are subject to various fault sources that can severely degrade their performance and autonomy. These faults originate from electrical, mechanical, and energy supply subsystems, or may emerge due to aging components. In our simulation study, representative faults have been injected to test the robustness of the proposed fault estimation and control strategy. The classification provided here is based on a comprehensive analysis of DC motor behavior, as referenced in [43].

- **Electrical Faults :** Electrical faults in DC motor-driven robots include short-circuits causing overheating and torque loss, open-circuits leading to power failures, and worn brushes inducing unstable motion. Poor commutation and insulation degradation may also cause sparking and component damage.
- **Mechanical Faults :** Mechanical faults result from wear and degradation, such as worn bearings causing vibrations, rotor imbalance reducing accuracy, and friction from dust or poor lubrication leading to energy loss and overheating.
- **Power Supply Faults :** Power supply instabilities, such as voltage drops, transients, or current surges, can cause speed oscillations, shutdowns, or damage to motor drivers. Loose connections or faulty cables may also lead to intermittent faults and erratic robot behavior.

All of these fault scenarios were emulated during the simulation phase to evaluate the efficiency and resilience of the proposed fault-tolerant estimation and control strategy.

5.4.2 Real-Time Fault Estimation Strategies

In the context of real-time implementation, two distinct strategies can be adopted for fault estimation, each with its own advantages and trade-offs.

The first strategy, consists of running an online estimation algorithm that continuously detects faults as they occur, regardless of their type or occurrence time. This method provides real-time responsiveness and is particularly effective in unpredictable environments where fault characteristics are unknown. However, it requires frequent computation, and the use of iterative estimation techniques can significantly increase processing time, especially as the number of iterations grows. To manage computational load while maintaining reactivity, we adopted a control frequency of 10 Hz the estimation process will be in the same node of control.

The second strategy relies on prior knowledge of the possible fault types. In this case, a fixed number of known fault profiles are identified and analyzed offline. Estimation is performed in advance under controlled conditions to determine characteristic patterns. Then, during online operation, the system only needs to recognize and match incoming fault signals to these pre-characterized forms. Once identified, the compensation can be applied more efficiently and accurately. This approach yields higher precision and faster fault rejection but is only applicable when fault types are limited and well understood.

Each method serves different operational contexts: the first is suited for general, which was used in our simulation, unpredictable environments, while the second excels in controlled systems with predefined fault scenarios.

5.5 Simulation Results and Validation

To validate the different components developed throughout the implementation, we adopt a step-by-step evaluation approach. The validation begins with the single-agent case, where both the control performance and the fault estimation accuracy are analyzed in a controlled environment. This phase helps verify the core functionalities in isolation and under simplified conditions. Once satisfactory performance is confirmed, we extend the analysis to the multi-agent case. This transition enables us to assess the robustness, scalability, and coordination capabilities of the proposed strategies in a distributed setup. Each stage is supported by detailed visualizations, quantitative error analysis, and observations that highlight the system's behavior under realistic constraints.

5.5.1 Single Agent case

To validate the proposed control strategy, we first consider a single-agent scenario in a simple environment without any faults or obstacles. The objective is to assess whether the robot can successfully follow the reference trajectory from non-zero initial conditions using the fuzzy fault-tolerant controller.

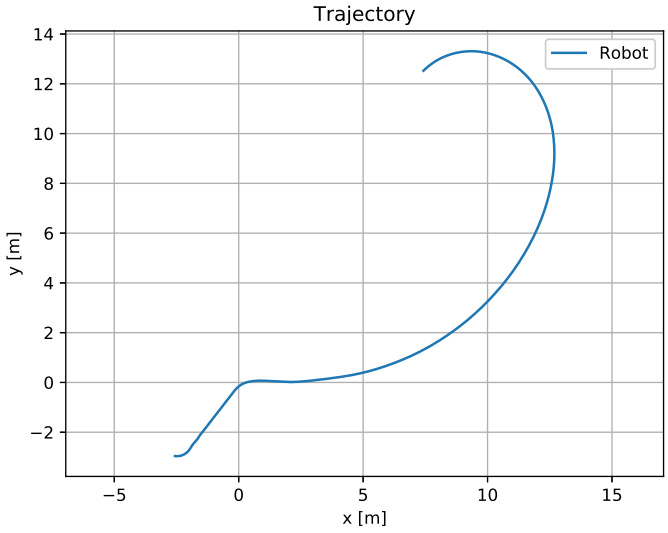
Several plots are generated to illustrate the behavior of the system:

5.5.1.1 Trajectory Tracking

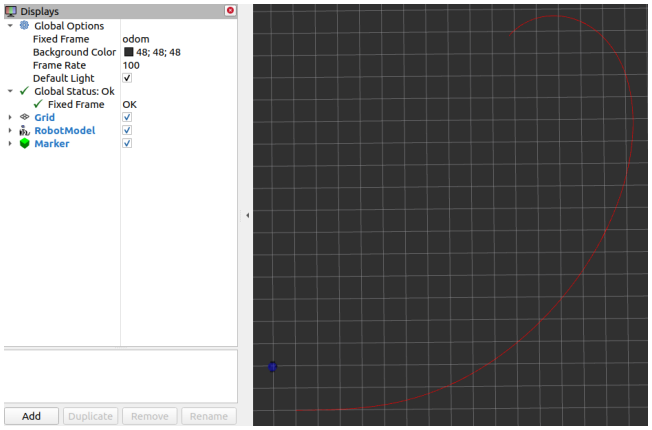
To verify whether the robot follows the predefined reference trajectory, it is necessary to visualize both the robot and the reference trajectory using the RViz visualization tool, as illustrated in Figure 5.7b. During the simulation, various signals are recorded for later analysis and display.

In this chapter, the same gain values and parameters are used for both the control and estimation processes, in both the single-agent and multi-agent scenarios. This allows for a consistent comparison of the method's performance between the theoretical results obtained from MATLAB simulations and the more realistic conditions of the practical simulation environment.

As illustrated in Figure 5.7a, even with non-zero initial conditions, the robot—as previously explained—searches for the closest point on the reference trajectory, which is (0,0). Once this point is reached, it proceeds to follow the trajectory as expected.



(a) Trajectory tracking



(b) robot state with reference trajectory

In Figure 5.8, the tracking errors e_x, e_y, e_θ converge to zero once the robot reaches the reference trajectory. Afterward, a small residual error appears, which corresponds to the tolerance specified in the algorithm. This level of error is acceptable and indicates that the robot can be considered to have reached the desired position.

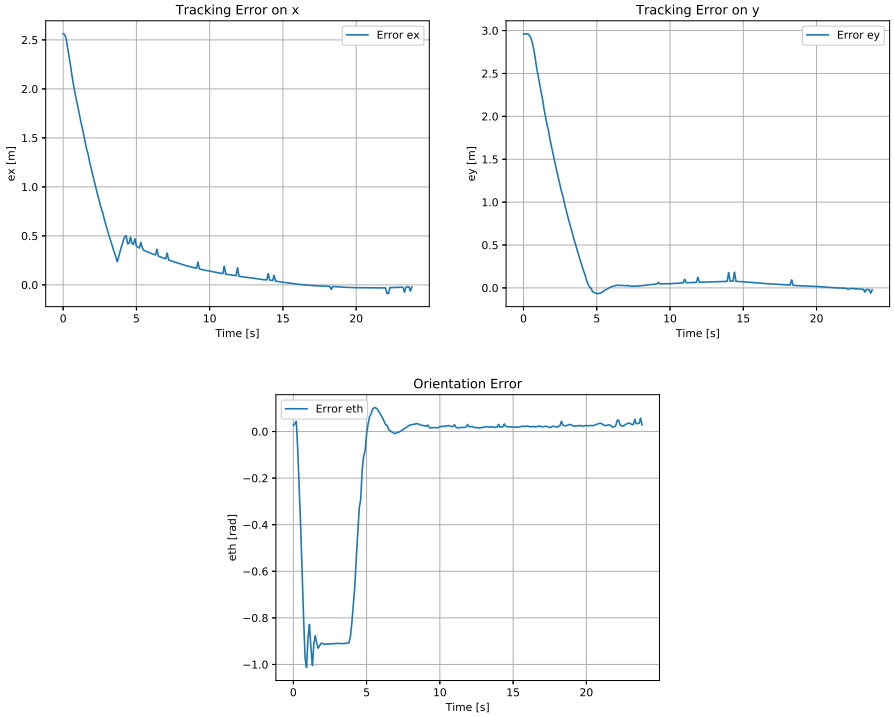


Figure 5.8: Tracking Errors

Figure 5.9 shows that the control velocities v and ω closely follow the shape of the reference inputs imposed by the trajectory planner. This similarity confirms that the controller successfully tracks the desired velocity profiles.

Moreover, the control inputs remain within admissible bounds and exhibit smooth transitions, indicating proper dynamic behavior without excessive oscillations or instability. The convergence toward the reference commands in both transient and steady-state phases validates the effectiveness of the implemented control strategy.

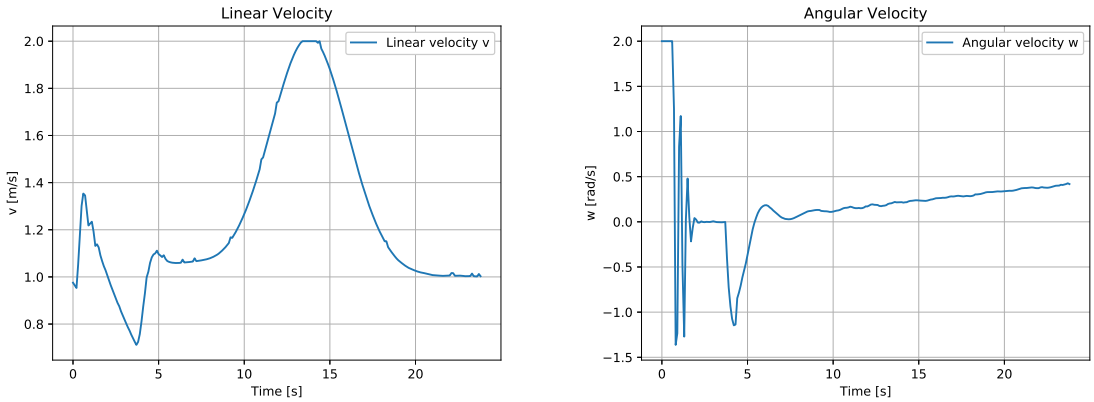


Figure 5.9: Control Signals (Velocities)

5.5.1.2 Obstacle avoidance

Trajectory tracking must be implemented in real time alongside the obstacle avoidance control. To evaluate the latter, static cylindrical obstacles with a diameter of 0.5 m are placed directly along the robot's reference path.

Several key signals are recorded to assess the reliability and effectiveness of the control strategy. As shown in Figure 5.10, the robot successfully detects and avoids obstacles that lie directly in its path.

As previously described, the fusion between the trajectory tracking and obstacle avoidance controllers is handled using a fuzzy logic scheme. This fusion ensures a smooth and balanced combination of both objectives, preventing one control objective from dominating the other, which could otherwise lead to divergence or failure in reaching the goal.

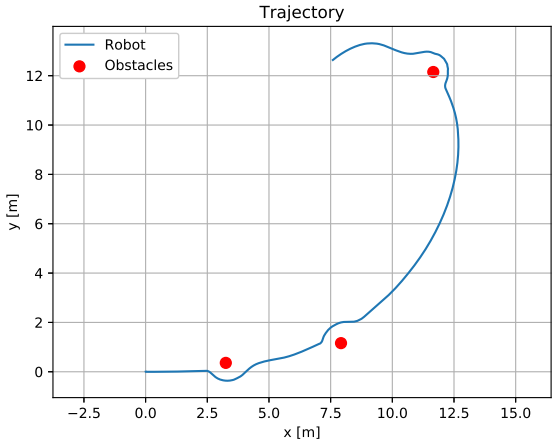
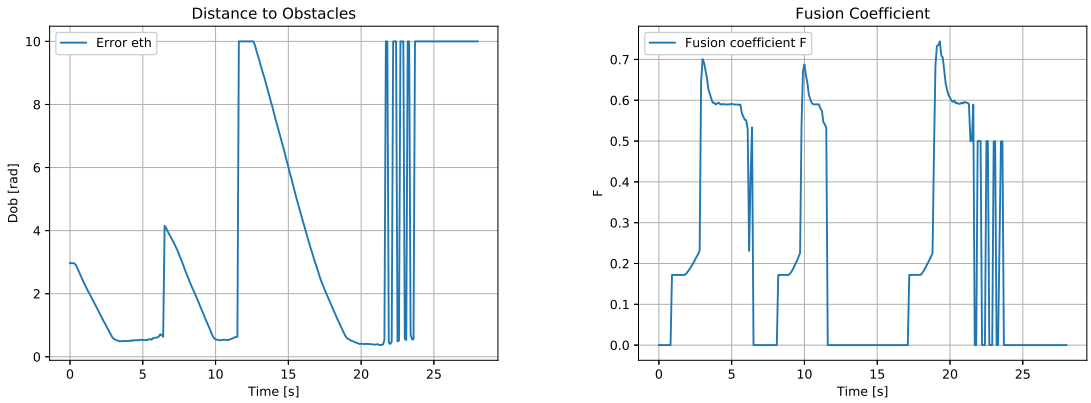


Figure 5.10: Obstacle avoidance

Figure 5.11a shows the measured distance between the robot and the closest obstacle, as detected by the onboard LiDAR sensor. To ensure numerical stability and bounded decision-making, the maximum distance was capped at 10 m instead of using an infinite value.

As the robot approaches obstacles—specifically around 5 s, 10 s, and 20 s sharp drops in distance are observed. In response, as illustrated in Figure 5.11b, the fusion coefficient F increases accordingly. This indicates a stronger influence of the obstacle avoidance controller during those time intervals.

The observed correlation between the LiDAR measurements and the variation in F confirms the efficiency and responsiveness of the fusion algorithm. It demonstrates that the control system dynamically prioritizes obstacle avoidance when necessary, while maintaining overall trajectory tracking integrity.



(a) Distance from the robot to obstacles

(b) Fusion Gain

A fusion coefficient F in the range of approximately 0.6 to 0.7 is sufficient to activate the obstacle

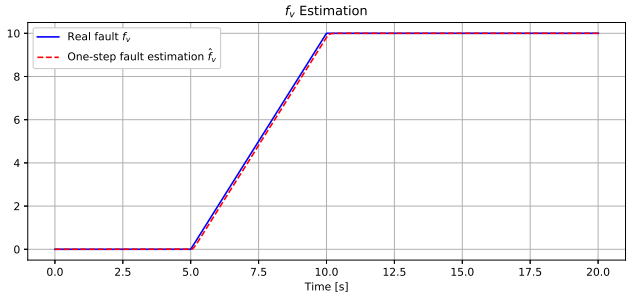
avoidance control. These values can be adjusted by redefining the membership functions in the fuzzy controller.

However, the value of F should not reach 1, as it is essential to maintain a minimum contribution from the trajectory tracking controller. This ensures that the robot continues to follow the global path while avoiding obstacles, preventing complete divergence from the intended trajectory.

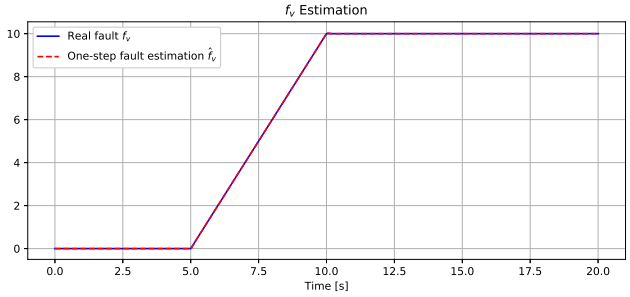
5.5.1.3 Fault Estimation

We now reach the main objective of this project: injecting faults with arbitrary shapes into the system. As previously discussed, this fault injection can be implemented both in simulation and in real-time scenarios.

In this section, we introduce actuator faults directly at the system input. These faults are the same as those used in the previous chapter. We estimate these faults using a single iteration of the k-step observer, and we analyze their impact on the robot's behavior, particularly in terms of trajectory tracking and control signals. As shown in Figures 5.12a and 5.13a, a small residual error remains compared to the true injected fault. These residual errors are approximately 0.195 and 2.323, respectively, for the linear and angular components.



(a) One-step Fault Estimation

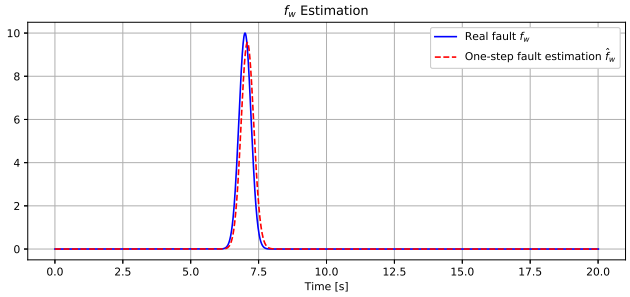


(b) 7-step Fault Estimation

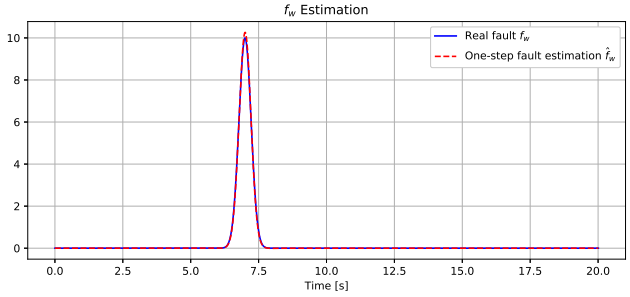
Figure 5.12: Input 1 fault estimation

After seven iterations, we observe in Figures 5.12b and 5.13b that the estimated fault signals closely match the actual faults. The residual errors are reduced to 0.048 and 0.276 for the linear and angular components, respectively.

Although the algorithm could be allowed to run for additional iterations, this is unnecessary in practice. As observed, further iterations have a negligible impact on the trajectory tracking performance, making the additional computational cost unjustified.



(a) One-step Fault Estimation

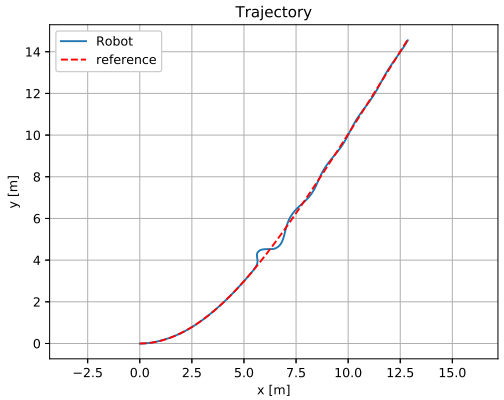


(b) 7-step Fault Estimation

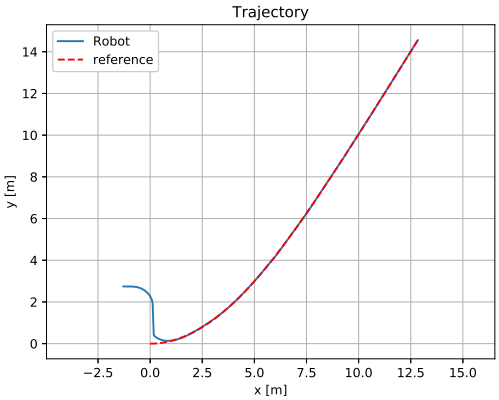
Figure 5.13: Input 1 fault estimation

As previously discussed, estimating the injected faults using only a single iteration is insufficient. This limitation is evident in Figure 5.14a, where a slight divergence from the reference trajectory is observed due to inadequate fault compensation.

However, after performing seven iterations, the effect of the fault becomes almost imperceptible at the trajectory level figure 5.14b. The compensation is effectively achieved, and the robot is able to maintain accurate trajectory tracking despite the presence of actuator faults.



(a) One-Step Fault Estimation



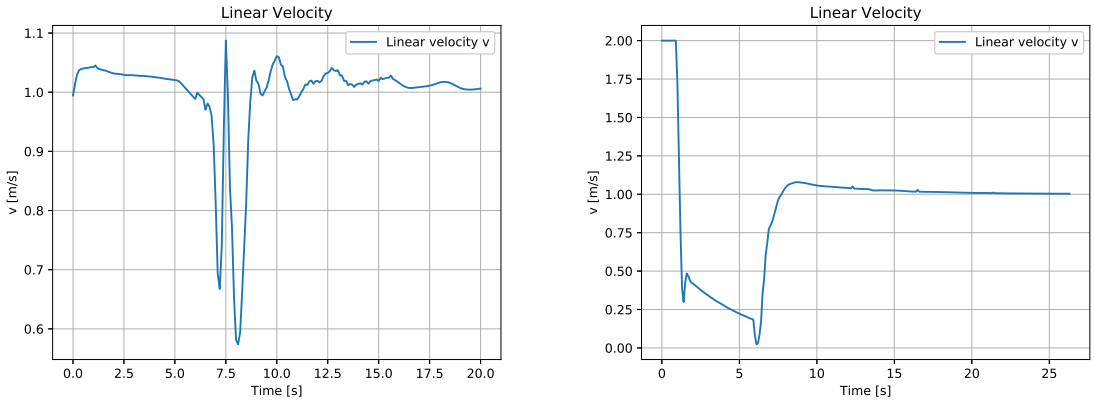
(b) 7-Step Fault Estimation

Figure 5.14: Trajectory tracking

The appearance of faults at the input level can destabilize the system. Since we are operating in an closed-loop configuration, the system tends to recover stability on its own, which in turn generates undesirable control signals—often deviating significantly from the predefined reference values.

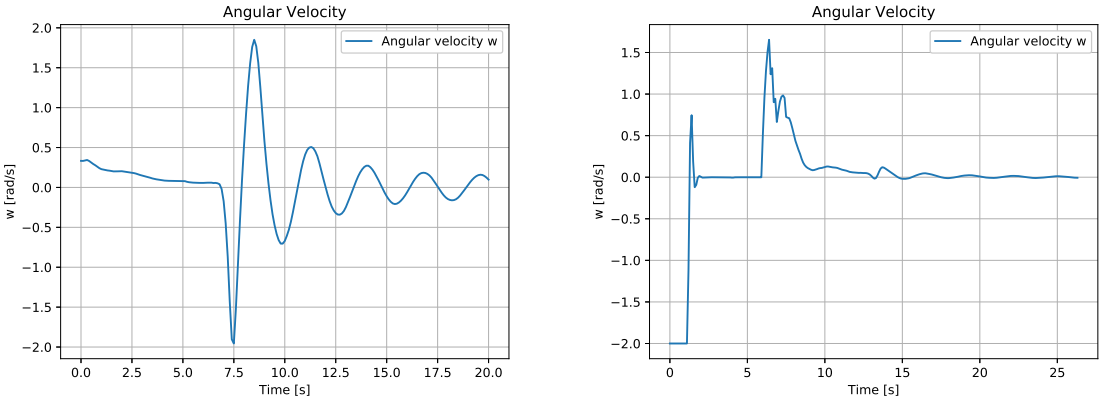
One of the key advantages of accurate fault estimation and compensation is the improvement

in the quality of the control inputs. As shown in Figure 5.15 for the linear velocity and Figure 5.16 for the angular velocity, the control signals become smoother and more consistent after compensation. This smoothness helps reduce mechanical stress on the actuators, thus enhancing their operational safety and longevity.



(a) One-Step Fault Estimation (b) 7-Step Fault Estimation

Figure 5.15: Control signals - Input 1



(a) One-Step Fault Estimation (b) 7-Step Fault Estimation

Figure 5.16: Control signals - Input 2

5.5.2 Multi Agent case

It is more relevant to extend the "k-step fault estimation" algorithm to multi-agent systems in order to evaluate its efficiency and reliability in a distributed context. This extension is justified by the fact that both single-agent and multi-agent frameworks share the same theoretical foundation and estimation logic.

We adopted the same simulation setup as in the previous chapter, using an identical communication prototype between the robots. To ensure better synchronization and reliable communication among the four robots, each agent was assigned a unique namespace prefix. This allowed us to reuse the same controller model across all agents within the simulation environment.

Figure 5.17 illustrates the communication structure between the ROS nodes, including the publication of reference trajectories in the visualization tool. This configuration provides complete insight into the behavior of each robot within the multi-agent network.

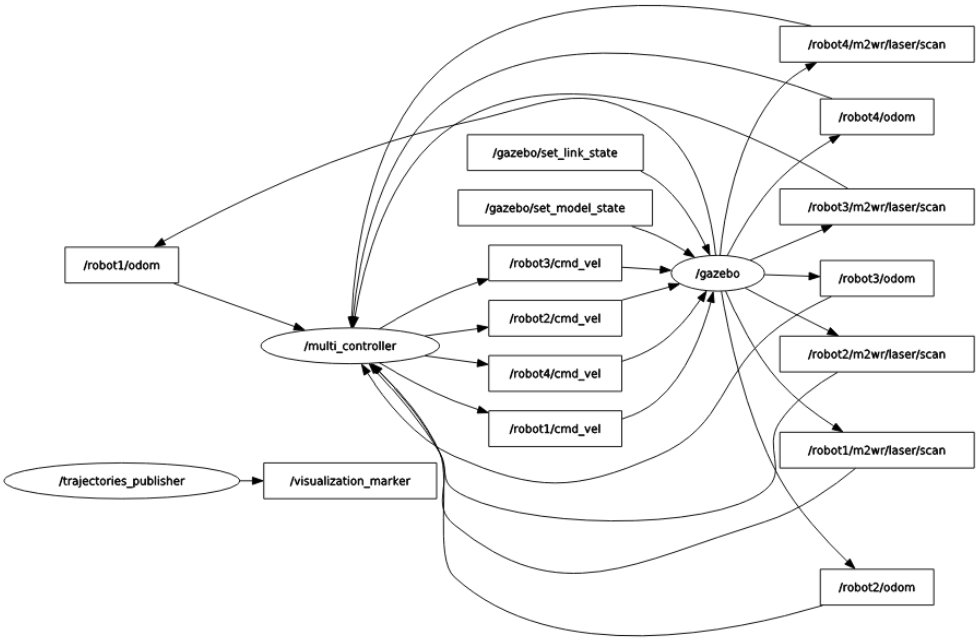


Figure 5.17: RQT graph for multi agent system

5.5.2.1 Trajectory Tracking

Robot 1 acts as the leader and guides the rest of the agents, as previously described. As shown in Figure 5.18, the multi-agent system successfully follows the reference trajectory under the following velocity profiles:

$$v_r = 1 \quad m/s,$$
$$w_r = 0.3\, e^{-0.2t} \quad rad/s$$

The results confirm that the coordinated tracking behavior is well maintained within the group, even under time-varying angular velocity references.

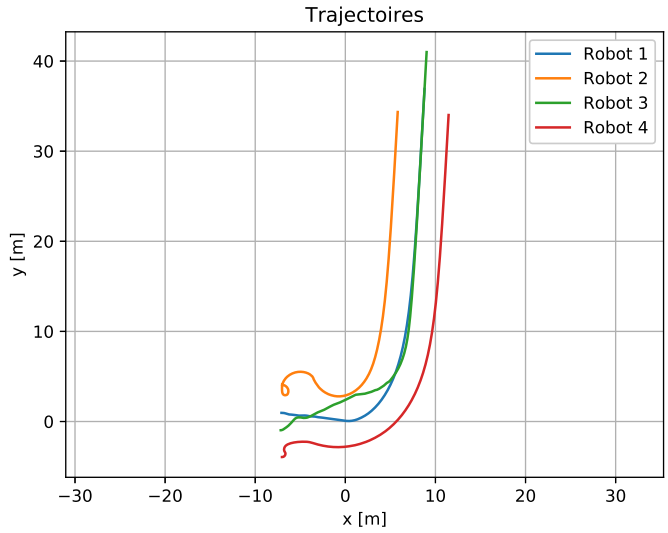


Figure 5.18: Trajectories tracking

As shown in Figure 5.19, the tracking errors gradually converge to zero after a transient phase. This behavior is mainly due to the anticipation mechanism used to determine the next target

position along the reference trajectory. As a result, the error requires some time to stabilize around zero.

Furthermore, the error curves are not perfectly smooth, which can be attributed to the numerical computation and the use of a relatively moderate control and update frequency. Despite this, the system exhibits satisfactory performance in terms of convergence and trajectory tracking accuracy.

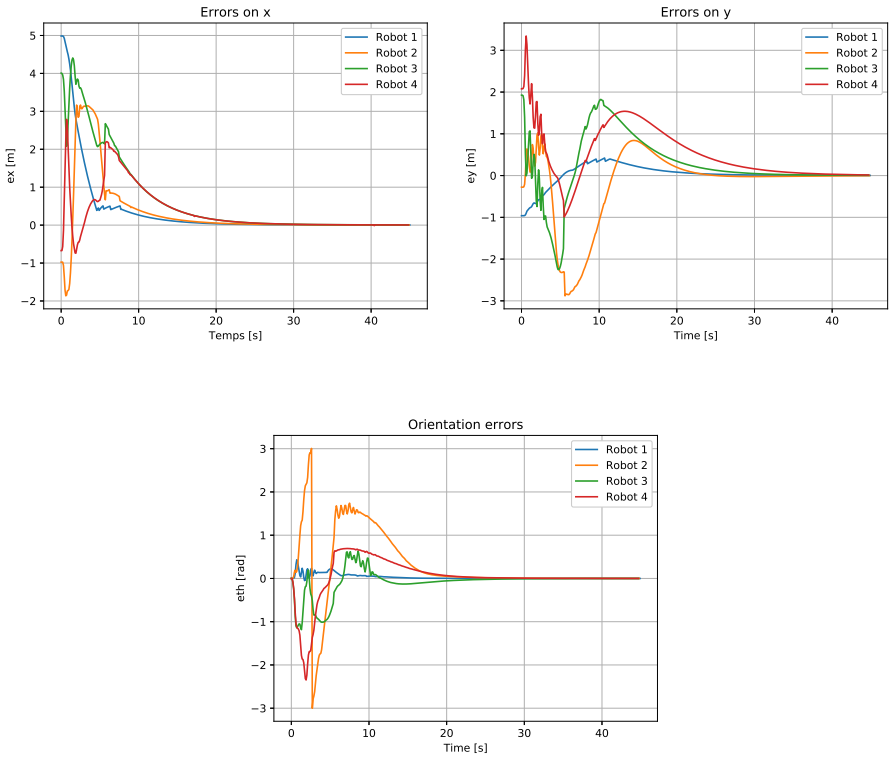
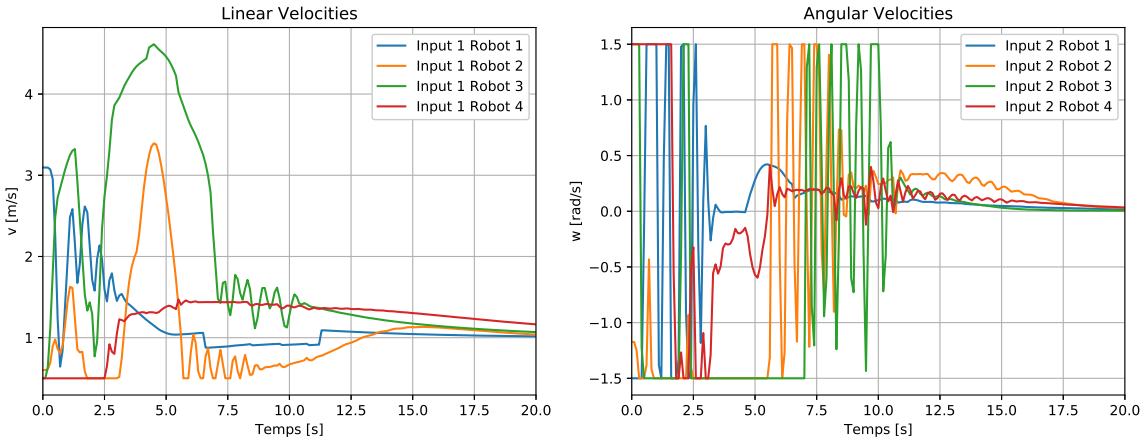


Figure 5.19: Tracking Errors

As observed in Figure 5.20, the velocity signals exhibit a transient phase during which the linear and angular velocities may reach relatively high values. To prevent potential damage to the robot's actuators and to ensure system stability, we imposed saturation limits on the control inputs.

Specifically, the angular velocity is constrained within the range $[-1.5, 1.5]$ rad/s, while the linear velocity is bounded between 0 and 6 m/s. These bounds serve to protect the hardware and to avoid generating excessively large velocities that could destabilize the overall system.



(a) Input 1 (b) Input 2

Figure 5.20: Control signals

To provide a more precise evaluation of formation control, we visualize the distance between the leader and each follower, as shown in Figure 5.21. It can be observed that all distances converge toward a constant value of 4 m, which was predefined in the control strategy.

This convergence confirms the effectiveness of the formation maintenance mechanism and highlights the stability of inter-agent coordination during trajectory tracking.

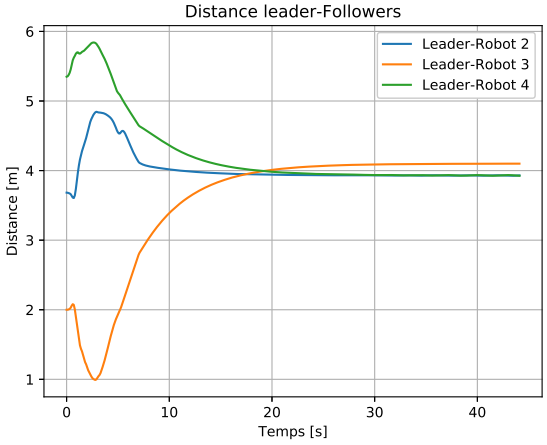


Figure 5.21: Distance between the leader and the followers

5.5.2.2 Obstacle avoidance

As we did in the single-agent case, the obstacle avoidance control must also be validated for each robot operating in real-time within the multi-agent configuration. A specific logic was defined to govern the behavior of this control mechanism:

- During the transient phase, if two robots come close to each other, they will treat one another as obstacles and perform mutual avoidance maneuvers.
- The follower robots track the actual position of the leader robot. If the leader avoids an obstacle, the followers continue tracking its updated trajectory. Once the leader returns to its original reference path, the followers also realign with it.

This logic was previously introduced and explained in detail in the preceding chapter, and is recalled here for clarity.

To test this behavior, we considered a simple scenario illustrated in Figure 5.22, where no obstacles are placed directly on the leader's reference trajectory. This choice helps to clearly visualize the trajectories of each robot and evaluate their interaction without interference from the leader's obstacle avoidance maneuvers,under the following velocity profiles:

$$v_r = 1 \quad m/s,$$
$$w_r = 0.06 \ \tanh(t - 10) \quad rad/s$$

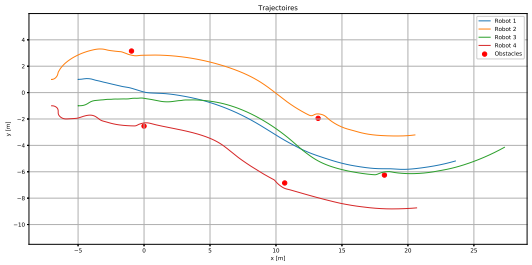
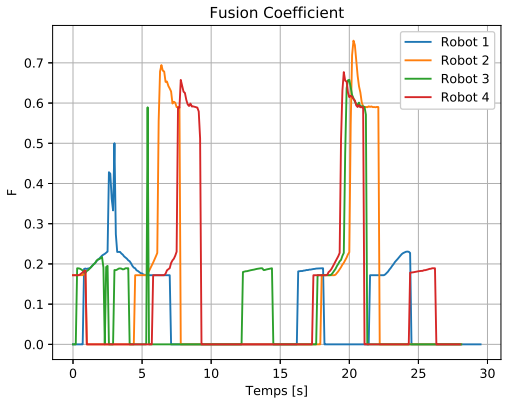
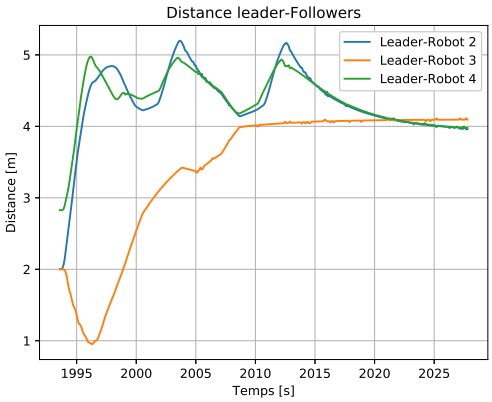


Figure 5.22: Obstacle avoidance - Multi Agent

Figures 5.23a and 5.23b demonstrate that the robots operate effectively in an environment containing static obstacles. The results confirm that the proposed control strategy ensures both reliable obstacle avoidance and consistent inter-agent coordination under realistic conditions.



(a) Fusion gain of all agents



(b) Distances to obstacles

5.5.2.3 Fault Estimation

To evaluate the performance of the fault estimator in the multi-agent system, we follow the same procedure as in the single-agent case.

5.5.2.3.1 One-Step Fault Estimation The same actuator faults used in the previous chapter were injected, but at different time instants for each of the four robots. The results of the one-step fault estimation are presented in Figures 5.24 and 5.25, which show the estimated faults applied to input 1 and input 2 of each robot, respectively.

These initial estimations provide a qualitative view of the estimator's ability to respond quickly, even in a distributed multi-agent context.

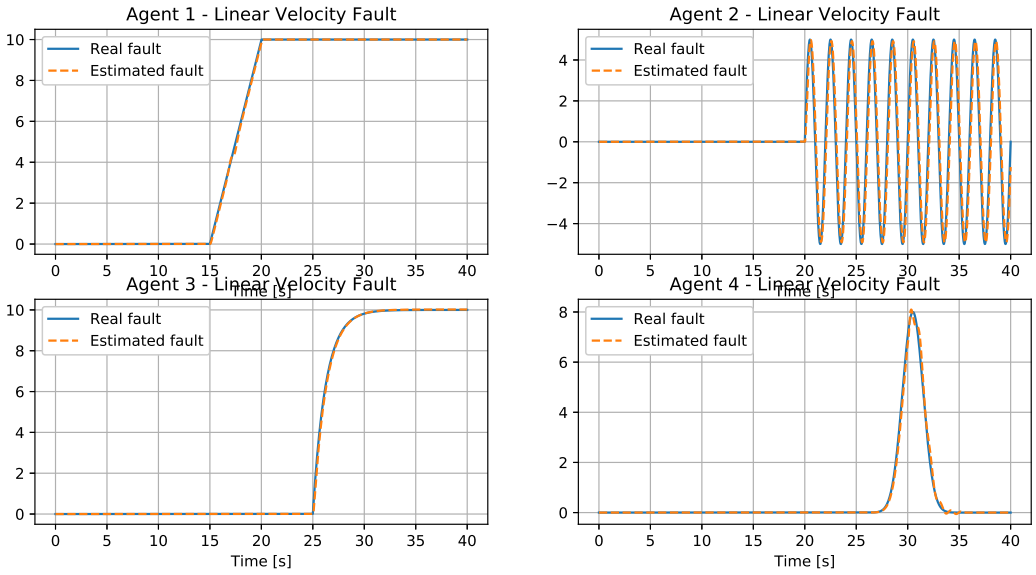


Figure 5.24: One-step fault estimation - Input 1

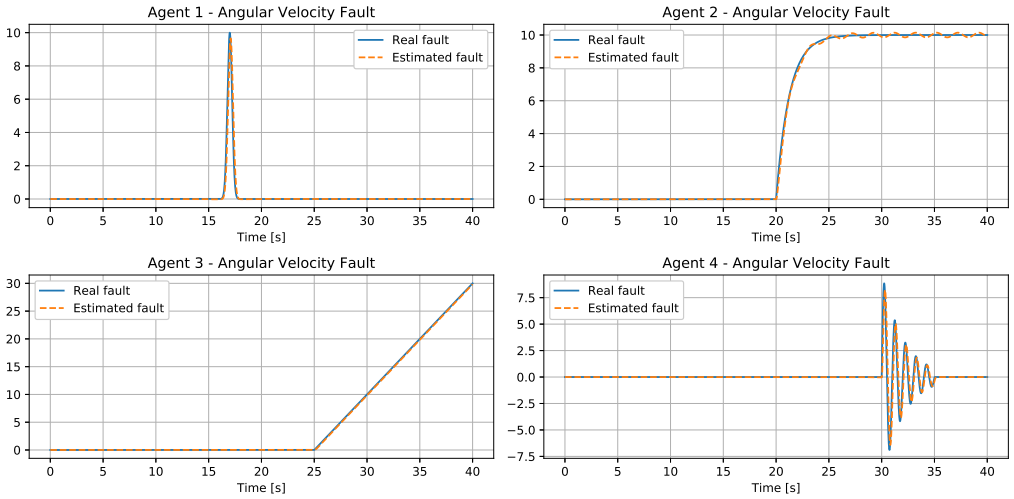


Figure 5.25: One-step fault estimation - Input 2

Figure 5.26 illustrates the trajectories of the four robots under a control strategy based on the one-step fault estimation. A noticeable deviation from the reference path is observed for each robot.

Such deviations are not acceptable in high-precision applications, as they indicate insufficient fault compensation when relying on a single iteration of the estimator.

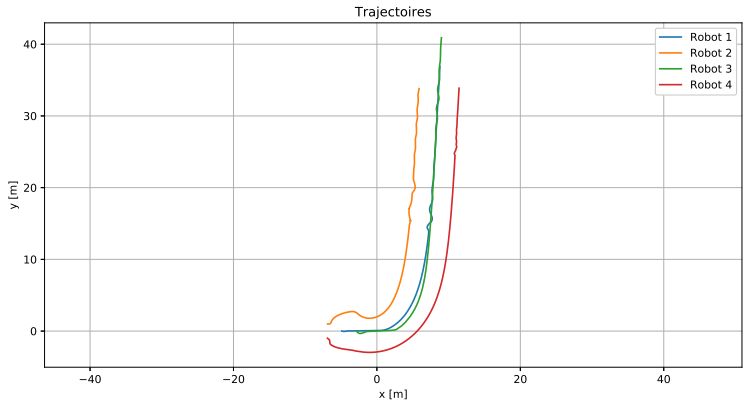


Figure 5.26: Trajectory tracking

For the same reference trajectory, Figure 5.27 shows that the tracking errors are directly affected by the presence of uncorrected faults, as also illustrated in the previous figure.

This confirms the need to increase the number of estimation iterations in order to improve the accuracy of the fault compensation and, consequently, the overall control performance.

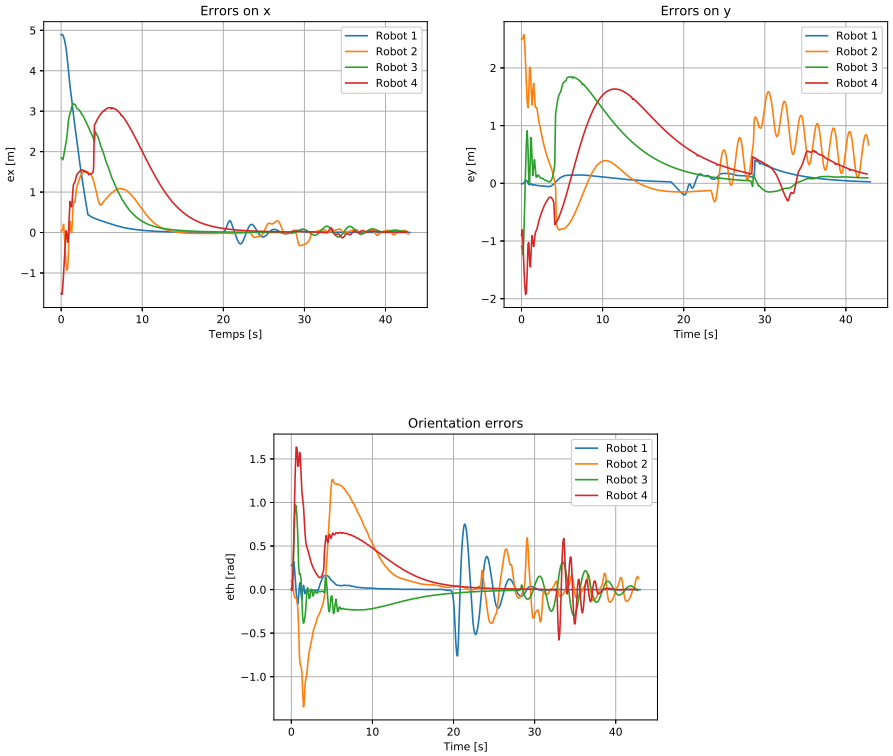
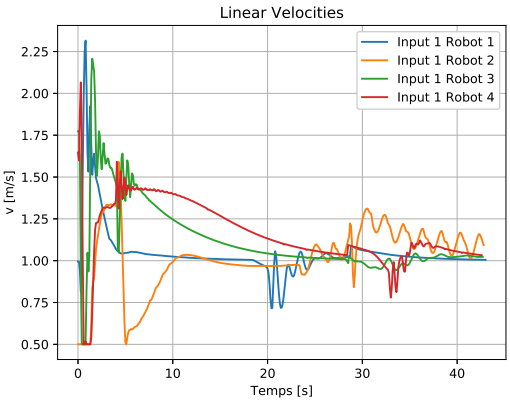


Figure 5.27: Tracking Errors

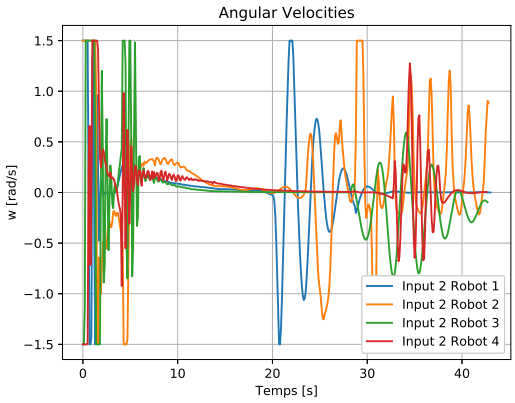
The presence of faults also affects the control signals, even after the system reaches the steady-

state phase, as shown in Figures 5.28a and 5.28b.

These disturbances in the control inputs highlight the impact of inaccurate fault estimation on actuator behavior, further emphasizing the necessity of refining the estimation process to ensure smooth and reliable control signals.



(a) input 1 for each robot



(b) input 2 for each robot

The estimation errors for the actuator faults after a single iteration are summarized below for each robot. The values correspond respectively to the linear and angular fault components (f_v, f_w) for Robots 1 to 4:

- Robot 1: $f_v = 0.399728, \quad f_\omega = 2.114611$
- Robot 2: $f_v = 1.296750, \quad f_\omega = 0.660917$
- Robot 3: $f_v = 0.555112, \quad f_\omega = 0.224343$
- Robot 4: $f_v = 0.688149, \quad f_\omega = 3.793464$

These results confirm that the estimation error remains significant for certain agents, especially for the angular faults. This justifies the need to perform more iterations to improve accuracy and ensure reliable control in fault-tolerant multi-agent scenarios.

5.5.2.3.2 4-step Fault Estimation After four iterations, the fault estimation errors were significantly reduced, with values as follows for each robot (f_v, f_ω):

- Robot 1: $f_v = 0.116411, \quad f_\omega = 0.473179$
- Robot 2: $f_v = 0.435226, \quad f_\omega = 0.159298$
- Robot 3: $f_v = 0.211115, \quad f_\omega = 0.042769$
- Robot 4: $f_v = 0.257568, \quad f_\omega = 1.566265$

These results are considered acceptable, as the estimation errors converge toward zero as expected. This improvement is clearly illustrated in Figures 5.29 and 5.30, which show the alignment between the estimated and actual fault signals after four iterations.

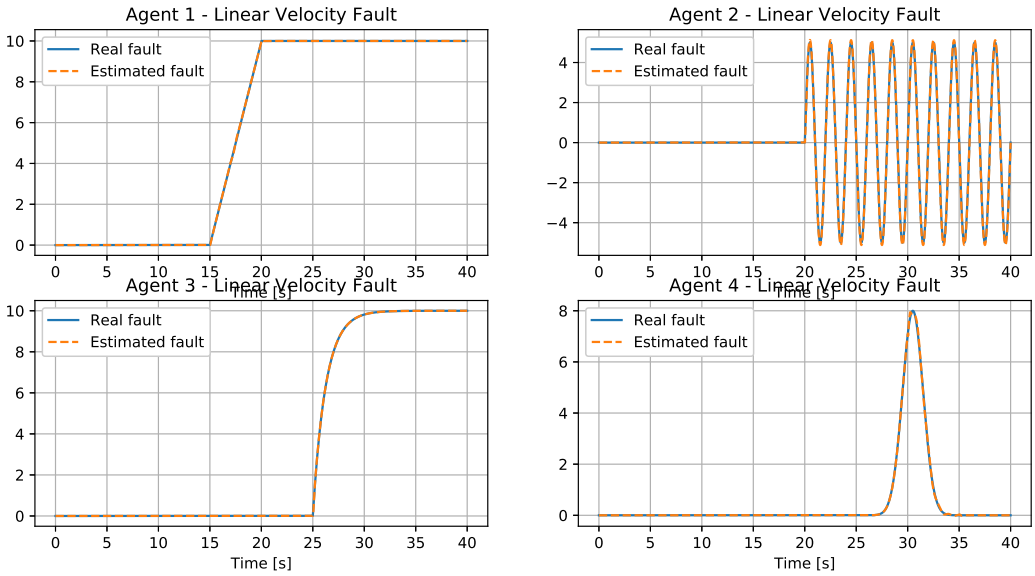


Figure 5.29: 4-step fault estimation - Input 1

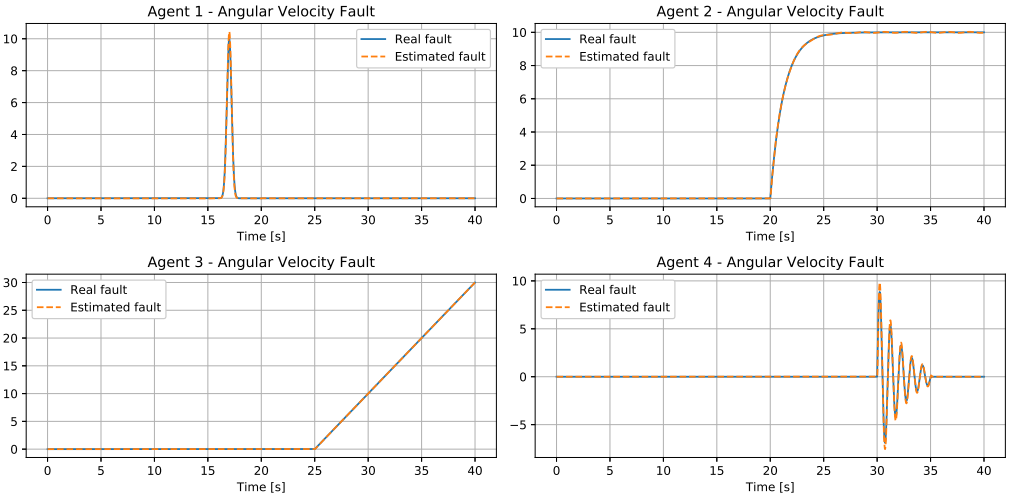


Figure 5.30: 4-step fault estimation - Input 2

As shown in Figure 5.31, the reference trajectory is accurately followed after four iterations of fault estimation. This confirms the effectiveness of the compensation mechanism and validates the robustness of the proposed approach in the presence of actuator faults within a multi-agent system.

The tracking errors clearly converge after four iterations of fault estimation, as illustrated in Figure 5.32. This demonstrates the capability of the estimator to effectively reduce the impact of actuator faults and restore accurate trajectory tracking performance.

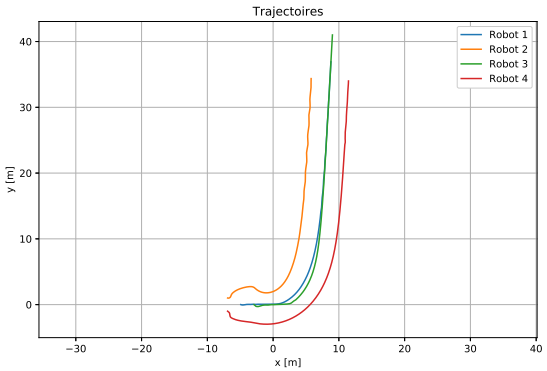


Figure 5.31: Trajectory tracking

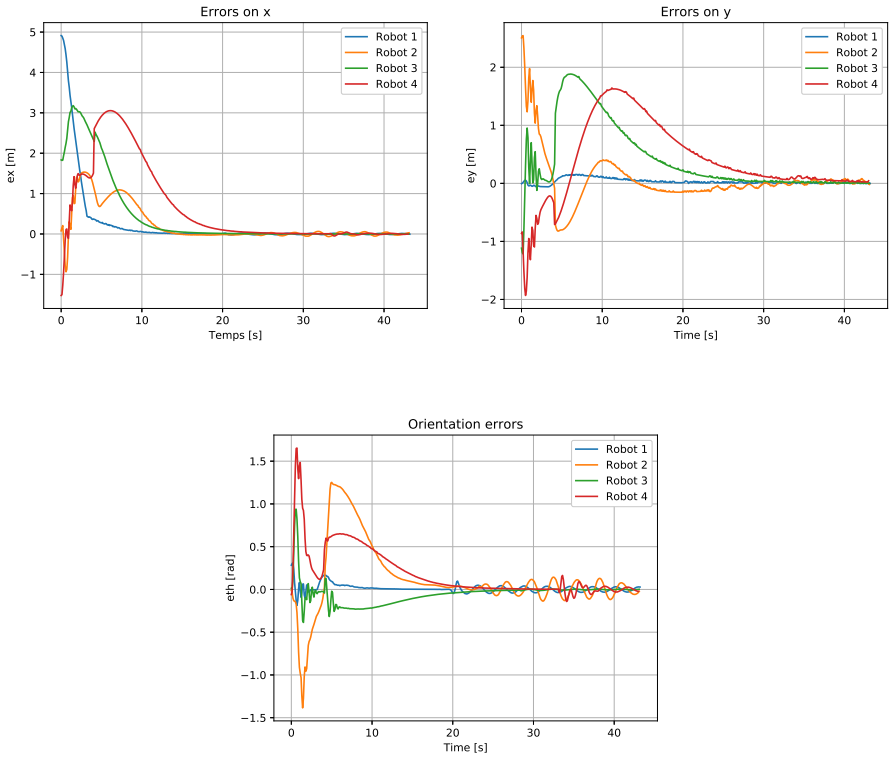
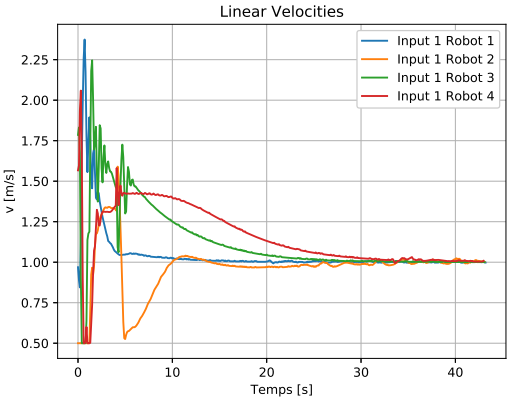


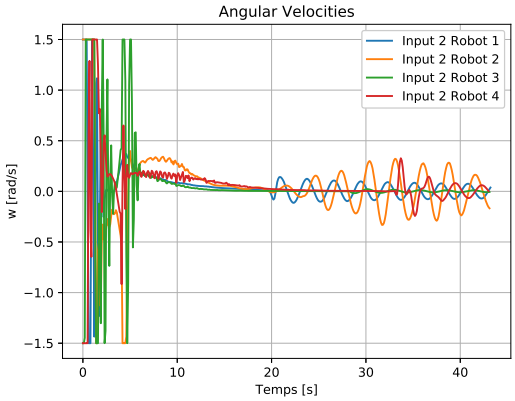
Figure 5.32: Tracking Errors

Finally, the control signals confirm the effectiveness of the fault compensation. As shown in Figure 5.33a, the linear velocity remains close to the desired value, indicating that the compensation mechanism has successfully restored proper control performance.

However, for the angular velocity, Figure 5.33b reveals that the signal is not yet fully stable. Although additional iterations of the estimator could further refine the compensation, this is not strictly necessary, as the main objective has already been achieved: the trajectory tracking is rigorous and robust, even in the presence of actuator faults.



(a) Input 1 for each robot



(b) Input 2 for each robot

Based on the results obtained, we can conclude that the proposed approach has been successfully validated in real-time conditions. However, compared to the results presented in the previous chapter, the real-time performance appears slightly degraded. This discrepancy can be attributed to practical constraints inherent to real-world implementation, such as limited computation frequency, sensor noise, communication delays, and actuation imperfections. Despite these factors, the system remains robust and the experimental outcomes are consistent with the theoretical expectations, thereby confirming the practical applicability of the method.

Specifically, the "k-step fault estimation" algorithm demonstrates reliable performance, providing accurate fault estimation and effective compensation within a multi-agent system. These results confirm that the estimator is suitable for applications requiring a given level of precision, while maintaining reasonable computational efficiency.

5.6 Limitations and Drawbacks of K-Step Fault Estimation

Despite its effectiveness in estimating faults, the implementation of the k-step fault estimation approach presents several limitations, particularly when extended beyond mobile robots:

- **Computational Load:** The method is computationally heavy, especially when processing real-time measurement signals from physical environments. High-performance hardware is often required to maintain real-time performance.
- **Communication Overhead in Multi-Agent Systems:** In multi-agent scenarios, reliable and fast inter-agent communication is crucial. The need to exchange state and estimation data frequently adds complexity and demands robust communication protocols.
- **Estimation Inaccuracy:** Estimation errors cannot be fully eliminated. The observer always retains a residual estimation gap that may affect control decisions.
- **Poor Performance with Fast-Varying Faults:** The estimator struggles with high-dynamic faults, which can cause overshoots or delayed responses, destabilizing the estimation loop and potentially the overall control system.

- **Dependency on Measurement Quality:** The performance is highly sensitive to noise, delays, and sensor inaccuracies. Any corruption in the measurements may compromise convergence or produce significant estimation errors.

- **Sampling Frequency Constraints:** There is a trade-off in selecting the sampling rate. A high frequency increases the computational burden, while a low frequency delays fault detection. Even with low frequencies, poor signal quality can lead to false estimations.

5.7 Future Work and Perspectives

Several directions can be explored to improve the robustness, efficiency, and applicability of the k-step fault estimation and control framework:

5.7.1 Enhancing the Estimator

- **Online adaptive gain computation:** The current implementation uses offline-computed observer gains based on fixed performance indices. A promising extension would be to design adaptive or reinforcement learning-based mechanisms that update gains in real-time, based on the system state and varying performance metrics.

- **Acceleration of the estimation process:** By allowing the observer to learn and store fault dynamic profiles, it may be possible to skip initial iterations and begin with a more accurate estimate, thereby reducing transient estimation errors.

- **Reduced-order or local-model-based observers:** For large-scale or complex systems, simplified observer structures can be used, where each observer is associated with a specific submodel, reducing the computational burden.

- **Hybrid approaches with AI-based estimators:** Fault recognition can be improved by combining traditional k-step observers with neural networks, support vector machines, or other machine learning techniques to enhance detection performance.

- **Development of a generic ROS package:** Creating a reusable ROS package dedicated to fault detection using k-step observers would promote broader use and ease integration across platforms.

- **Extension to stochastic and multiplicative faults:** In future work, we plan to study the case of stochastic or random faults, which pose a significant challenge due to their high variability and rapid dynamics. The current k-step observer has shown limitations when dealing with such fast-evolving fault profiles. Addressing this will require more robust estimation strategies. Additionally, we will investigate multiplicative fault models by modifying the fault representation in the system equations, which may offer a more realistic modeling framework for certain actuator or sensor degradations.

5.7.2 Advanced Applications

- **Deployment on real robots with noisy sensors:** Transitioning from Gazebo simulations to physical platforms like differential-drive robots equipped with real sensors would allow validation under realistic conditions with sensor fusion technics to improve state estimation accuracy.

- **Extension to more complex multi-agent systems:** The framework can be adapted to heterogeneous or dynamic agents such as UAVs or robotic arms in logistics in dynamic environment with moving obstacles .

- **Migration to ROS2:** Porting the architecture to ROS2 can leverage DDS, offering better communication latency and real-time performance support for large-scale applications.

Conclusion

This chapter demonstrated the successful implementation of the k-step fault estimation observer within a realistic robotic framework using ROS and Gazebo. The simulation environment provided a valuable platform to emulate real-world constraints such as sensor delays, noisy measurements, and actuator saturation, while maintaining full control over the testing scenarios.

The proposed estimator proved to be effective in both single and multi-agent configurations. It was capable of detecting and compensating for actuator faults in real-time, thereby maintaining accurate trajectory tracking and stable control inputs. The results showed that increasing the number of iterations significantly improved estimation accuracy and control performance.

Despite the absence of real hardware, the use of a physics-based simulator with realistic sensor models ensured that the findings remain valid and transferable to physical platforms. This practical validation confirms the robustness and adaptability of the fault-tolerant strategy in conditions that closely resemble real-world deployment.

The success of this implementation paves the way for future integration on actual robotic systems, with further enhancements such as adaptive gain tuning, learning-based estimation, and deployment on heterogeneous agent networks.

General Conclusion

This project successfully achieved its goal of developing and validating a fault-tolerant control strategy for nonlinear systems using iterative observer-based methods. By combining Takagi-Sugeno fuzzy modeling, iterative k-step observer design, and distributed estimation logic, the proposed framework provides an effective and robust solution for detecting and compensating actuator faults in both centralized and decentralized configurations.

Theoretical developments were carried out to design fault estimation observers that ensure convergence, robustness, and adaptability under various fault conditions. The observer structure was rigorously analyzed using Lyapunov based techniques, and synthesis conditions were derived using linear matrix inequalities. The iterative nature of the observer, which updates fault estimates at each time step, provides fast responsiveness an essential characteristic for embedded control systems and autonomous agents.

The proposed methodology was first validated in a single-agent configuration, where simulation results demonstrated accurate reconstruction of fault signals and rapid convergence of the estimation error. The approach was then extended to multi-agent systems, leveraging graph theory and Kronecker product representations to design a distributed estimation scheme. This extension enables each agent to reconstruct local faults while cooperating with neighboring agents, thus enhancing the overall fault-awareness of the networked system.

In the context of robotics, the framework was applied to differential drive mobile robots. These systems, commonly used in both research and industrial applications, exhibit nonlinear and coupled dynamics that make fault compensation particularly challenging. The integration of the observer-based FTC approach allowed the robots to maintain stability and trajectory tracking performance, even when actuator faults were introduced during operation.

Beyond theoretical and simulated environments, the project culminated in a real-time implementation using the Robot Operating System (ROS). The observer and control algorithms were deployed in a realistic robotic simulation architecture, demonstrating that the proposed solution is not only theoretically sound but also practically feasible. Real-time tests confirmed that the method performs well under realistic timing constraints, communication delays, and software integration requirements.

This work, therefore, bridges the gap between abstract theoretical control design and practical implementation on autonomous robotic platforms. It illustrates how modern FTC techniques can be made computationally efficient, modular, and adaptable to the real-world demands of distributed and embedded systems.

While the initial objectives of this project have been successfully met, several promising directions can be pursued as future work:

- Extending the fault estimation capability to include sensor faults and combining actuator and sensor fault reconstruction in a unified observer framework,
- Enhancing scalability for large-scale multi-agent systems, with time-varying communication topologies and event-triggered interactions,
- Integrating adaptive and learning-based mechanisms to allow online tuning of observer parameters using machine learning or data-driven methods,
- Transitioning from simulated robots to real physical platforms and evaluating the performance in hardware-in-the-loop or real-world scenarios,
- Exploring applications in other domains such as unmanned aerial vehicles, autonomous underwater robots, or cooperative industrial manipulators.

In conclusion, the developed fuzzy FTC approach for nonlinear systems using iterative observer represents a solid and scalable contribution to the field of modern control theory, particularly in the context of nonlinear and distributed systems. It combines rigorous mathematical foundations with practical validation, opening the door to reliable and autonomous operation in the presence of faults. As system complexity and autonomy continue to grow, such fault-tolerant strategies will play a critical role in ensuring resilience, safety, and continuous performance.

Bibliography

[1] Aboubakr Hetatche. Modélisation floue de type takagi-sugeno appliquée à un bioprocédé. Magister thesis, Université Ferhat Abbas - Sétif 1, 2014.

[2] Kamel Merahi. Estimation d'État et diagnostic de fonctionnement des systèmes non linéaires. Magister thesis, Université Badji Mokhtar Annaba, 2010.

[3] Aboubakr Hetatche. Modélisation floue de type takagi-sugeno appliquée à un bioprocédé. Master's thesis, Université de Sétif, mai 2018.

[4] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, volume 15, pages 116–121, 1985.

[5] Anca Maria Nagy. *Analyse et synthèse de multimodèles pour le diagnostic. Application à une station d'épuration.* PhD thesis, Institut National Polytechnique de Lorraine - INPL, 2010.

[6] D. Filev. Fuzzy modeling of complex systems. *International Journal of Approximate Reasoning*, 5(3):281–290, 1991.

[7] Mohamed HAZERCHI. *Contribution à la commande et l'estimation d'état des systèmes décrits par des multi-modèles.* Doctoral thesis, École Nationale Polytechnique, Alger, 2018.

[8] Mohammed Oudghiri. *Commande multi-modèles tolérante aux défauts: Application au contrôle de la dynamique d'un véhicule automobile.* PhD thesis, Université de Picardie Jules Verne, 2008.

[9] Kamyar Mehran. *Takagi-Sugeno Fuzzy Modeling for Process Control.* PhD thesis, Newcastle University, 2008.

[10] Bakour Haroune. Commande tolérante aux défauts des systèmes non linéaires par l'approche multimodèle : Application à un robot mobile de type unicycle. End of study project, École Nationale Polytechnique, octobre 2023.

[11] Badr Mansouri. *Contribution à la synthèse de lois de commandes en poursuite de trajectoire pour les systèmes flous de type Takagi-Sugeno incertains.* Doctoral thesis, Université de Reims Champagne-Ardenne, décembre 2005.

[12] Lamia Ben Hamouda. *Sur la synthèse de commandes prédictives tolérantes aux défauts à base de modèles T-S flous.* Doctoral thesis in international co -control, École Nationale d'Ingénieurs de Tunis et Université de Rouen, septembre 2015.

[13] K. Tanaka and H. Wang. *Fuzzy Control System Design and Analysis: A Linear Matrix Inequality Approach.* John Wiley and Sons Inc, 2001.

[14] P. Bergsten and R. Palm. Observers for takagi-sugeno fuzzy systems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 32:114–121, 2002.

[15] Morteza Seidi, Marzieh Hajiaghamemar, and Bruce Segee. *Fuzzy Control Systems: LMI-Based Design*, chapter 18. InTech, 2012.

[16] Lazeregue Abderraouf. Commande tolérante aux défauts par l'approche multimodèle : Application à un quadrirotor. End of study project, École Nationale Polytechnique, septembre 2020.

[17] Quan-Yong Fan and Guang-Hong Yang. Event-based fuzzy adaptive fault tolerant control for a class of nonlinear systems. *IEEE Transactions on Fuzzy Systems*, 26(5), 2018.

[18] Ron J. Patton. Fault-tolerant control: the 1997 situation. *IFAC Proceedings Volumes*, 30(18), 1997.

[19] Xiao He et al. Fault-tolerant control for an internet-based three-tank system: Accommodation to sensor bias faults. *IEEE Transactions on Industrial Electronics*, 64(3):2266–2275, 2017.

[20] Youmin Zhang and Jin Jiang. Issues on integration of fault diagnosis and reconfigurable control in active fault-tolerant control systems. *IFAC Proceedings Volumes*, 39(13):1437–1448, 2006.

[21] D. Ichalal. *Estimation et diagnostic de systèmes non linéaires décrits par un modèle de Takagi-Sugeno*. Thèse de doctorat, Institut National Polytechnique de Lorraine, Nancy, 2009.

[22] Hakim Achour. *Commande robuste et tolérante aux défauts : approche multimodèle*. Doctoral thesis, École Nationale Polytechnique, Alger, 2022.

[23] Mohamed Hazerchi and Rachid Illoul. Premise variables cases for lmi based ts fuzzy observers. *International Journal of Control and Automation*, 11(2):75–88, 2018.

[24] Mahmoud Chilali and Pascal Gahinet. H infinity design with pole placement constraints: An lmi approach. *IEEE Transactions on Automatic Control*, 41(3):358–367, March 1996.

[25] Sheng-Juan Huang and Guang-Hong Yang. Fault tolerant controller design for t–s fuzzy systems with time-varying delay and actuator faults: A k-step fault-estimation approach. *IEEE Transactions on Fuzzy Systems*, 22(6):1526–1536, 2014.

[26] Guohui Xu, Yimin Liu, Jianliang Chen, and Zixuan Li. Fault-tolerant control of a class of nonlinear multi-agent systems with actuator faults based on k-step observers. In *Proceedings of the 2022 China Automation Congress (CAC)*. IEEE, 2022.

[27] Yongsheng Ma, Mouquan Shen, Haiping Du, Yuesheng Ren, and Guangrui Bian. An iterative observer-based fault estimation for discrete-time t-s fuzzy systems. *International Journal of Systems Science*, 2020.

[28] Farhad Ghanipoor, Carlos Murguia, Peyman Mohajerin Esfahani, and Nathan van de Wouw. Ultra local nonlinear unknown input observers for robust fault reconstruction. *arXiv preprint arXiv:2204.01455*, 2022.

[29] Shuiwang Yuan and Lina Yao. Fault tolerant control design for stochastic distribution control systems based on k-step fault estimation. In *Proceedings of the 2023 IEEE Symposium on Fault Detection, Supervision, and Safety for Technical Processes (SAFEPROCESS)*. IEEE, 2023.

[30] Qinyuan Liu, Zidong Wang, Xiao He, and D. H. Zhou. Event-based h_∞ consensus control of multi-agent systems with relative output feedback: The finite-horizon case. *IEEE Transactions on Cybernetics*, 45(8):1574–1584, 2015.

[31] Ke Zhang, Bin Jiang, and Marcel Staroswiecki. Dynamic output feedback fault tolerant controller design for takagi–sugeno fuzzy systems with actuator faults. *IEEE Transactions on Fuzzy Systems*, 18(1):194–201, Feb. 2010.

[32] A. Teixeira, I. Shames, H. Sandberg, et al. Distributed fault detection and isolation resilient to network model uncertainties. *IEEE Transactions on Cybernetics*, 44(11):2024–2037, 2014.

[33] B. Farrera, F. R. Lopez-Estrada, et al. Distributed fault estimation of multi-agent systems using a proportional-integral observer: a leader-following application. *International Journal of Applied Mathematics and Computer Science*, 30(3):551–560, 2020.

[34] Y. Liu and G. H. Yang. Integrated design of fault estimation and fault tolerant control for linear multi-agent systems using relative outputs. *Neurocomputing*, 329(15):468–475, 2019.

[35] T. Youssef, M. Chadli, R. H. Karimi, and R. Wang. Actuator and sensor faults estimation based on proportional integral observer for ts fuzzy model. *Journal of the Franklin Institute*, 354(6):2524–2542, 2017.

[36] X. H. Liu, J. Han, and X. J. Wei. Intermediate observer based distributed fault estimation for multi-agents systems. *Acta Automatica Sinica*, 16(1):142–152, 2020.

[37] S. J. Huang and G. H. Yang. Fault tolerant controller design for t-s fuzzy systems with time-varying delay and actuator faults: a k-step fault-estimation approach. *IEEE Transactions on Fuzzy Systems*, 22(6):1526–1540, 2014.

[38] Mehran Mesbahi and Magnus Egerstedt. *Graph Theoretic Methods in Multiagent Networks*. Princeton University Press, 2010.

[39] Guohui Xu, Yimin Liu, Jianliang Chen, and Zixuan Li. Fault-tolerant control of a class of nonlinear multi-agent systems with actuator faults based on k-step observers. In *2022 China Automation Congress (CAC)*, pages 1368–1373. IEEE, 2022.

[40] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi. A stable tracking control method for an autonomous mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 384–389, Cincinnati, OH, USA, May 1990. IEEE.

[41] Khaled Akka and Farid Khaber. Optimal fuzzy tracking control with obstacles avoidance for a mobile robot based on takagi-sugeno fuzzy model. *Transactions of the Institute of Measurement and Control*, pages 1–10, 2018.

[42] Gregor Klančar and Igor Škrjanc. Tracking-error model-based predictive control for mobile robots in real time. *Robotics and Autonomous Systems*, 55(6):460–469, 2007.

[43] Austin Hughes and Bill Drury. *Electric Motors and Drives: Fundamentals, Types and Applications*. Newnes, 5th edition, 2023. Used as a reference for DC motor fault analysis in mobile robotics.

[44] Reza Olfati-Saber, J Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

Appendix A

$\mathcal{L}_2$ Attenuation and Lemmas

1.1 $\mathcal{L}_2$ Approach

Consider the linear system:

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t) + Du(t) \tag{A.1}$$

If the system is stable and the input $u(t)$ is bounded, then the output $y(t)$ is also bounded. There exists a scalar $\gamma > 0$ such that:

$$\int_0^{+\infty} y^T(t)y(t)dt \leq \gamma^2 \int_0^{+\infty} u^T(t)u(t)dt \tag{A.2}$$

The scalar γ is called the $\mathcal{L}_2$-*gain* of the system.

1.2 Lemmas

Lemma 1 (Bounded Real Lemma): Inequality (A.2) holds for all $u(t) \neq 0$ if and only if there exists a matrix $P > 0$ such that:

$$\begin{bmatrix} A^TP + PA + C^TC & PB + C^TD \\ B^TP + D^TC & D^TD - \gamma^2 I \end{bmatrix} < 0 \tag{A.3}$$

To minimize γ, one may solve the following convex optimization problem:

$$\min \gamma \tag{A.4}$$

subject to:

$$\begin{bmatrix} A^TP + PA + C^TC & PB + C^TD \\ B^TP + D^TC & D^TD - \gamma I \end{bmatrix} < 0, \quad P > 0 \tag{A.5}$$

Proof: The proof uses the Lyapunov function:

$$V(x(t)) = x^T(t)Px(t), \quad P > 0 \tag{A.6}$$

Integrating both sides:

$$\int_0^{+\infty} \dot{x}^T(t)Px(t)dt + \int_0^{+\infty} y^T(t)y(t)dt \leq \gamma^2 \int_0^{+\infty} u^T(t)u(t)dt \tag{A.7}$$

which leads to:

$$\int_0^{+\infty} \left(\dot{x}^T(t)Px(t) + y^T(t)y(t)\right) dt \leq \gamma^2 \int_0^{+\infty} u^T(t)u(t)dt \tag{A.8}$$

Which implies:

$$\dot{x}^T(t)Px(t) + y^T(t)y(t) \leq \gamma^2 u^T(t)u(t) \tag{A.9}$$

Using system equations, the quadratic form becomes:

$$\begin{aligned} &x^T(t)(A^TP + PA + C^TC)x(t) + 2x^T(t)(PB + C^TD)u(t) + \\ &u^T(t)(D^TD - \gamma^2 I)u(t) \leq 0 \end{aligned} \tag{A.10}$$

Matrix form:

$$\begin{bmatrix} x(t) \\ u(t) \end{bmatrix}^T \begin{bmatrix} A^TP + PA + C^TC & PB + C^TD \\ B^TP + D^TC & D^TD - \gamma^2 I \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \leq 0 \tag{A.11}$$

Thus inequality (A.12) is satisfied:

$$\begin{bmatrix} A^TP + PA + C^TC & PB + C^TD \\ B^TP + D^TC & D^TD - \gamma^2 I \end{bmatrix} < 0 \tag{A.12}$$

Lemma 2: For any matrices X, Y of compatible dimensions, the inequality:

$$X^TY + Y^TX \leq X^T\Omega^{-1}X + Y^T\Omega Y, \quad \Omega > 0 \tag{A.13}$$

is verified.

Lemma 3 (Schur Complement): Given matrices $\Psi(x)$, $S(x)$, $R(x)$ with $R(x) = R(x)^T$, the following LMIs are equivalent:

$$\begin{bmatrix} \Psi(x) & S(x) \\ S^T(x) & R(x) \end{bmatrix} > 0 \tag{A.14}$$

$$R(x) > 0, \quad \Psi(x) - S(x)R(x)^{-1}S^T(x) > 0 \tag{A.15}$$

Appendix B

LMI Regions

2.1 LMI Regions

The time-domain behavior of a linear system is linked to the location of its poles in the complex plane. In Takagi-Sugeno systems, this behavior depends on the location of the poles of the sub-models (polytope vertices). The real parts of the poles influence the convergence speed of the associated modes. The imaginary parts, on the other hand, affect oscillation presence and overshoot, as well as 5% settling time. Consequently, one effective approach to improve controller or observer performance is to place the poles of the closed-loop or observer system within LMI-defined complex-plane regions. These are known as LMI regions.

Definition 1 ([24])**.** *A region $\mathcal{S}$ in the complex plane is called an* LMI region *if there exists a symmetric matrix $M \in \mathbb{R}^{m\times m}$ and a matrix $N \in \mathbb{R}^{m\times m}$ such that:*

$$\mathcal{S} = \{z \in \mathbb{C} : f_S(z) < 0\} \tag{B.1}$$

with

$$f_S(z) = M + zN + z^*N^T \tag{B.2}$$

where z^* denotes the complex conjugate of z. The function $f_S(z)$ is called the characteristic function of the region.

In other words, an LMI region is a region of the complex plane characterized by an LMI depending on z and z^*, where $a = \Re(z)$ and $b = \Im(z)$. LMI regions are thus convex sets.

2.2 Examples of LMI Regions

Let $a = \Re(z)$ and $b = \Im(z)$, then:

$$a = \frac{z + z^*}{2}, \qquad b = \frac{z - z^*}{2j} \tag{B.3}$$

The left half complex plane can be characterized by $a < 0$, hence its LMI region representation is:

$$f_S(z) = z + z^* \tag{B.4}$$

The region S_1 (left half-plane shifted by α):

$$f_{S_1}(z) = z + z^* + 2\alpha \tag{B.5}$$

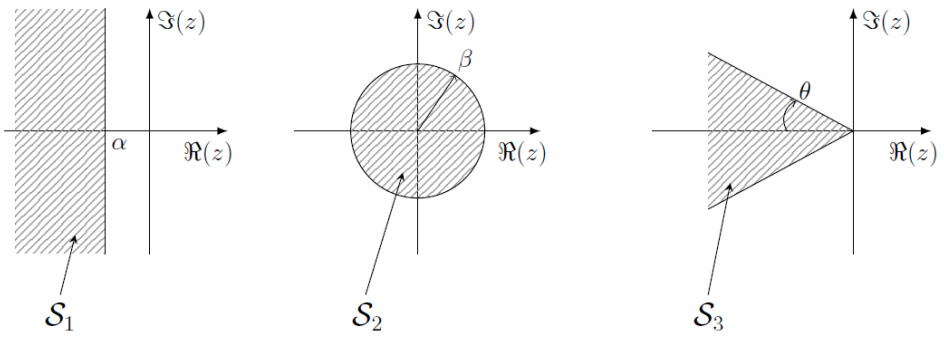


Figure 2.1: Examples of LMI regions.

The disk centered at the origin S_2 is characterized by:

$$z^* z - \beta^2 < 0 \tag{B.6}$$

which can be rewritten using the Schur complement:

$$f_{S_2}(z) = \begin{bmatrix} -\beta^2 & z \\ z^* & -1 \end{bmatrix} < 0 \tag{B.7}$$

The sector S_3 with angle 2θ defined by $|\arg(z)| < \theta$ has LMI form:

$$f_{S_3}(z) = \begin{bmatrix} \sin(\theta)(z + z^*) & \cos(\theta)(z - z^*) \\ \cos(\theta)(z^* - z) & \sin(\theta)(z + z^*) \end{bmatrix} < 0 \tag{B.8}$$

2.2.1 Pole Placement Using LMI Regions

Theorem 7 ([24]). *The eigenvalues of a real matrix M lie inside the region $\mathcal{S}$ if and only if there exists a symmetric matrix $X > 0$ such that:*

$$M_S(M, X) = A \otimes X + B \otimes MX + B^T \otimes XM^T < 0 \tag{B.9}$$

Here, $\otimes$ denotes the Kronecker product.

To use this in practice, the function $f_S(z)$ is substituted with:

$$(X, MX, XM^T) \mapsto (1, z, z^*) \tag{B.10}$$

For example, if the eigenvalues of M must lie in S_1, we require:

$$\exists X > 0 : MX + XM^T + 2\alpha X < 0 \tag{B.11}$$

For region S_2, we require:

$$\exists X > 0 : \begin{bmatrix} -\beta X & MX \\ XM^T & -\beta X \end{bmatrix} < 0 \tag{B.12}$$

For region S_3, we require:

$$\exists X > 0 : \begin{bmatrix} \sin(\theta)(MX + XM^T) & \cos(\theta)(MX - XM^T) \\ \cos(\theta)(XM^T - MX) & \sin(\theta)(MX + XM^T) \end{bmatrix} < 0 \tag{B.13}$$

Theorem 8 ([24]). *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two LMI regions. The eigenvalues of M lie in $\mathcal{S}_1 \cap \mathcal{S}_2$ if and only if there exists $X > 0$ such that:*

$$M_{S_1}(M, X) < 0, \quad M_{S_2}(M, X) < 0 \tag{B.14}$$

2.3 Multi Agent Faults

The following faults were injected in the Chapter 2.3.3 in the part of the multi-agent system.

Agent	**Linear Velocity Fault** $f_v(t)$	**Angular Velocity Fault** $f_\omega(t)$
1	$\begin{cases} 2(t-5), & 5 \le t \le 10 \\ 10, & t > 10 \\ 0, & \text{otherwise} \end{cases}$	$\begin{cases} 10e^{-10(t-7)^2}, & 6 \le t \le 8 \\ 0, & \text{otherwise} \end{cases}$
2	$\begin{cases} 10, & 8 \le t \le 13 \\ 0, & \text{otherwise} \end{cases}$	$\begin{cases} 10(1-e^{-0.8(t-8)}), & 8 \le t \le 13 \\ 10, & t > 13 \\ 0, & \text{otherwise} \end{cases}$
3	$\begin{cases} 10(1-e^{-0.8(t-10)}), & 10 \le t \le 15 \\ 10, & t > 15 \\ 0, & \text{otherwise} \end{cases}$	$\begin{cases} 2(t-10), & 10 \le t \le 15 \\ 10, & t > 15 \\ 0, & \text{otherwise} \end{cases}$
4	$8 \cdot e^{-\frac{(t-8.5)^2}{2}}$	$\begin{cases} 10e^{-0.5(t-8)}\sin(2\pi(t-8)), & 8 \le t \le 11 \\ 0, & \text{otherwise} \end{cases}$

Table 2.1: Injected fault signals for each robot's linear and angular velocity inputs.

Appendix C

Graph Theory Notations

Information exchange among agents in a multi-agent system is typically modeled using directed or undirected graphs [38]. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V} = \{1, 2, \ldots, p\}$ is the node set (each node representing an agent), and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set.

In a directed graph, the edge $(i, j) \in \mathcal{E}$ indicates that agent j receives information from agent i, but not necessarily vice versa. In contrast, an undirected graph assumes mutual information exchange, i.e., if $(i, j) \in \mathcal{E}$, then $(j, i) \in \mathcal{E}$ as well. Self-loops (i, i) are typically excluded unless otherwise specified.

A neighbor of node j is any node i such that $(i, j) \in \mathcal{E}$, and the set of neighbors of node j is denoted by $\mathcal{N}_j$. A weighted graph assigns a nonnegative weight to each edge. The union of a collection of graphs is defined by the union of their respective node and edge sets.

A directed path is a sequence of directed edges such as $(i_1, i_2), (i_2, i_3), \ldots$. A cycle is a directed path that starts and ends at the same node. A directed graph is said to be strongly connected if there exists a directed path from every node to every other node. An undirected graph is said to be connected if there exists a path between any two distinct nodes.

A directed tree is a directed graph in which every node has exactly one parent, except for one node called the *root*, which has no incoming edge and has a directed path to all other nodes. A tree in an undirected graph is a connected graph with no cycles.

A subgraph $(\mathcal{V}^s, \mathcal{E}^s)$ of a graph $(\mathcal{V}, \mathcal{E})$ satisfies $\mathcal{V}^s \subseteq \mathcal{V}$ and $\mathcal{E}^s \subseteq \mathcal{E} \cap (\mathcal{V}^s \times \mathcal{V}^s)$. A directed spanning tree is a subgraph that is a directed tree and covers all nodes in $\mathcal{V}$. The existence of a directed spanning tree implies that there is at least one node from which a directed path exists to all other nodes.

Adjacency Matrix and Laplacian Matrix: Let $A = [a_{ij}] \in \mathbb{R}^{p \times p}$ denote the adjacency matrix of $\mathcal{G}$, where $a_{ij} > 0$ if $(i, j) \in \mathcal{E}$, and $a_{ij} = 0$ otherwise. For undirected graphs, $a_{ij} = a_{ji}$, and the adjacency matrix is symmetric. The graph is called balanced if $\sum_{j=1}^{p} a_{ij} = \sum_{j=1}^{p} a_{ji}$ for all i.

The Laplacian matrix $L = [\ell_{ij}] \in \mathbb{R}^{p \times p}$ is defined as:

$$\ell_{ii} = \sum_{\substack{j=1 \\ j \neq i}}^{p} a_{ij}, \quad \ell_{ij} = -a_{ij} \quad \text{for } i \neq j$$

This can also be written as $L = D - A$, where D is the degree matrix, defined by $D = \text{diag}(d_1, \ldots, d_p)$, with $d_i = \sum_{j=1}^{p} a_{ij}$. For undirected graphs, L is symmetric and positive semi-definite. The smallest eigenvalue is zero, and its multiplicity equals the number of connected components in the graph.

In both directed and undirected cases [44], the Laplacian matrix has row sums equal to zero. All nonzero eigenvalues of an undirected Laplacian are real and nonnegative. The second smallest eigenvalue $\lambda_2(L)$, known as the *algebraic connectivity*, is positive if and only if the graph is connected, and it quantifies the convergence rate in consensus algorithms.

Finally, for a given weight matrix $S = [s_{ij}]$, the directed graph $\Gamma(S)$ is defined by placing an edge from node j to node i if $s_{ij} \neq 0$, and no edge otherwise.

Business Model Canvas

1. Key Partners	**2. Key Activities**	**3. Value Proposition**
• Academic research labs • ROS and open-source communities • Robotics hardware suppliers	• Fault-tolerant control design • Real-time implementation in ROS • Simulation and validation of faults	• Robustness against faults • Online estimation and compensation • Compatible with ROS and Arduino • Energy efficiency and safety increase
4. Customer Relationships	**5. Customer Segments**	**6. Channels**
• Personalized technical support • Co-development with partners • Workshops and tutorials • Clear and interactive documentation	• Robotics and mechatronics startups • Embedded critical systems • Research in autonomous systems • Biomedical robotics	• GitHub (open-source) • Scientific publications • Professional networks (LinkedIn) • Conferences and exhibitions
7. Key Resources	**8. Cost Structure**	**9. Revenue Streams**
• FTC and adaptive control expertise • ROS + MATLAB code base • Test platforms (Arduino, robots) • Experimental data	• Hardware components • Development and testing time • Software licenses (e.g. MATLAB) • Conference participation	• Licensing of advanced modules • Custom integration services • Industrial partnerships • Training and consulting