

Optimal Sensor Protection and Measurement Corruption Detection in Safety-Critical Systems

by

Chuan Tian Zhang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2025

© Chuan Tian Zhang 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis addresses the detection and mitigation of sensor faults in safety-critical systems through secure estimation techniques. Sensor faults, whether accidental or adversarial, pose significant risks in autonomous vehicles, aviation, robotics, and other domains heavily reliant on accurate sensor data for safe operation. Traditional fault tolerance methods typically depend on hardware redundancy or ad-hoc designs for specific systems, approaches that can be prohibitively costly, or simplified assumptions about fault conditions that may not hold in practice. Recent advances in secure estimation provide a general framework with provable guarantees against sensor faults; however, their central requirement—sparse observability—is a worst-case scenario analysis, limiting their applicability in practical systems.

To address this limitation, this thesis introduces the concept of sensor protection, explicitly modeling selected sensors as immune to faults. This serves as an initial step toward capturing the practical scenario where some sensors are more fault-tolerant than others. Although prior studies have implicitly assumed sensor protection by restricting potential fault locations, explicit modeling of sensor protection and its theoretical implications for fault tolerance have not been formally explored. This thesis extends the sparse observability framework to include protected sensors, broadening the applicability of secure estimation’s theoretical guarantees. Additionally, a metric termed the *safety factor* is introduced to quantify a system’s resilience to sensor faults, enabling targeted enhancements in robustness under practical resource constraints.

Further, this thesis adapts secure state-reconstruction methods to develop a robust fault detection algorithm suitable for nonlinear systems through linearization. We validate our methods extensively through simulation studies, retrospective analysis of a real-world autonomous vehicle racing incident, and practical implementation on a skid-steer robot. Results demonstrate significant improvements in real-time fault detection and operational safety under diverse fault conditions.

Overall, this work bridges theoretical advances in secure estimation with real-world deployment considerations, providing a structured methodology to enhance the reliability and safety of autonomous systems.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Dr. Christopher Nielsen and Dr. Derek Rayside. Derek provided me the opportunity to engage with exciting autonomous vehicle projects, one of which directly inspired this thesis. Chris consistently challenged me with rigorous academic standards and offered invaluable support throughout this work.

I am also grateful to my teammates at WATonomous, Waterloo Autonomous Racing, and MIT-PITT-RW. Their collaboration and camaraderie created unforgettable experiences. In particular, I thank Charles Zhang for introducing me to autonomous vehicles, Rowan Dempster for his exemplary early leadership, and Andrew Saba, Andy Keats, Cindy Heredia, Andrew Tresansky, Brian Mao, Nayana Suvarna, along with many others from around the globe for the numerous early mornings and late nights spent at the Indianapolis Motor Speedway and the Las Vegas Motor Speedway.

I would also like to thank the staff at the Waterloo RoboHub—Brandon DeHart, Robert Wagner, and Alexander Werner—for their generous support in enabling the deployment component of this research. They provided access to workspace and equipment, and facilitated the use of the Jackal robot and the VICON system for physical deployment. Their technical expertise and flexibility in accommodating the evolving needs of the project were instrumental in realizing the practical aspects of this work.

My heartfelt appreciation goes to my friends, whose support and encouragement have been invaluable throughout this journey. Special thanks to Benson Guo for guiding me through periods of burnout, Anna Zheng for her insightful discussions about life and the world, and Stephen Wang for being the tidiest roommate imaginable.

Finally, I extend my deepest gratitude to my family for their unwavering support in all my endeavors. Thank you to my parents for providing a nurturing environment to grow and thrive, and to my brother, grandparents, and extended family members for their constant love and encouragement.

Dedication

To my family, for their unwavering support; to those who believed in me, even when I didn't; and to all the robots who didn't crash — and to those that did, for teaching me more.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	x
List of Tables	xiii
List of Abbreviations	xiv
List of Symbols	xv
1 Introduction	1
1.1 Contributions	3
1.1.1 Sparse Observability with Sensor Protection	3
1.1.2 Optimal Sensor Protection Strategy	3
1.1.3 Fault Detection and Real-Time Mitigation	4
1.2 Organization	4
1.3 Notation	5

2	Related Works	6
2.1	Theoretical Development	6
2.1.1	Early Approaches	6
2.1.2	Secure Estimation for Linear Time-Invariant Systems	7
2.1.3	Secure Estimation for Noisy Systems	8
2.1.4	Secure Estimation for Linear Time-Varying and Nonlinear Systems	9
2.1.5	Additional Lines of Work	9
2.2	Applications	10
3	Methodology	13
3.1	Sparse observability	13
3.2	Protected Sensors	16
3.3	Safety Factor	20
3.4	The Optimal Sensor Protection Problem	22
3.5	Properties of the Safety Factor Function	25
3.5.1	Monotonicity	25
3.5.2	Non-submodularity and non-supermodularity	26
3.6	The Measurement Corruption Detection Problem	28
3.6.1	Secure State-Reconstruction	29
3.6.2	Handling Uncertainty	31
3.6.3	Implementation	31
3.6.4	Computational Complexity	34
3.7	Time-Varying Systems	36
4	Path-Following for a Car-Like Vehicle: Simulations and Case Study	40
4.1	Nominal Path Following for a Car-like Vehicle	40
4.1.1	System Model	40
4.1.2	Control Objective	42

4.1.3	State Estimation	48
4.1.4	Control Design	48
4.2	Linearization and Discretization	50
4.3	Optimal Sensor Protection	54
4.4	Measurement Corruption Detection	55
4.5	Simulation	56
4.5.1	Initial Conditions	56
4.5.2	Fault Injection	56
4.5.3	Detector Configuration	56
4.5.4	Simulation Setup	59
4.5.5	Results	59
4.5.6	Timing Analysis	63
4.6	Case Study	64
4.6.1	The IAC Racing Incident	66
4.6.2	Vehicle Model	69
4.6.3	Optimal Sensor Protection and Measurement Corruption Detection	73
5	Deployment to a Skid-Steer Robot	75
5.1	Hardware and Sensor Setup	75
5.2	Mathematical Model	76
5.2.1	Control Objective	78
5.2.2	Model Linearization and Discretization	80
5.3	Optimal Sensor Protection	81
5.4	Fault Detection and Mitigation	82
5.5	Fault Injection and Results	84
5.6	Discussions	85

6	Conclusions	88
6.1	Key Contributions	88
6.1.1	Sparse Observability with Protection	88
6.1.2	Optimal Sensor Protection	89
6.1.3	Fault Detection in Nonlinear Systems	89
6.2	Summary of Findings	89
6.3	Future Directions	90
6.4	Final Remarks	91
	References	92

List of Figures

1.1	A screenshot from the live broadcast of the Indy Autonomous Challenge (IAC) time trial where the vehicle crashed into a wall due to a sensor malfunction [26].	2
1.2	A summary of the data from the IAC crash, showing the bird's-eye view of the crash, the heading over time, and the heading standard deviation over time.	2
3.1	Example of an autonomous system with state estimation.	32
3.2	Example of an autonomous system with state estimation and sensor fault detection.	33
3.3	Example of a control loop with sensor fault detection.	34
4.1	Figure eight path.	43
4.2	Distance along the figure eight path with respect to the path parameter λ	45
4.3	Curvature profile of the figure eight path.	46
4.4	Velocity profile of the vehicle following the figure eight path.	47
4.5	Block diagram of the Unscented Kalman Filter (UKF) denoting the inputs (sensor measurements) and outputs (state estimates).	48
4.6	Lookahead point on the path. Derived from [22].	49
4.7	Illustration of the pure pursuit algorithm showing the lookahead point, the vehicle's position, the instantaneous center of rotation (ICR), and the desired steering angle. The orange line represents the expected trajectory of the (x, y) position of the vehicle when following the desired steering angle. Derived from [22].	51

4.8	Nominal simulation of the vehicle following the figure eight path.	52
4.9	Examples of sensor faults	57
4.10	Average precision and recall for different λ_ϵ values	60
4.11	Detection rate for bias faults in sensor 3 (heading sensor θ_m). Notice that the detection rate experiences a sharp change when the fault magnitude exceeds a threshold.	62
4.12	Detector running time. Faults are injected at the 30-second mark.	65
4.13	Racing line for the time-trial event. The vehicle starts in the pit area at the top straight and travels counterclockwise. Coordinates are in the Local Tangent Frame. Background image from [19].	67
4.14	Sensor setup for AV-21 during the time-trial. 1. GNSS antennas, 2. steering angle sensor (inside the cabin), 3. wheel encoder (near the axle), 4. INS (inside the cabin).	68
4.15	Bird's-eye view of the vehicle trajectory, highlighting the points where the vehicle experienced the fault and collided with the wall.	70
4.16	Heading and velocity data, highlighting the fault and crash times.	71
4.17	Detection times for the velocity and heading faults.	74
5.1	A schematic showing the side and front views of the Jackal [15].	76
5.2	A photograph of the Jackal platform used in the study.	77
5.3	The reference path, a circle with radius $R = 2$ meters.	79
5.4	A nominal control pipeline for the Jackal.	83
5.5	A control pipeline for the Jackal with fault detection and mitigation. . . .	83
5.6	Illustration of the rollback mechanism. The red dashed rectangle indicates the measurement window in which a fault is detected. Within this window, the detector identifies the set of faulty sensors. These identified faults allow us to reconstruct modified measurements, $Y'_{k-N+1}, \dots, Y'_k$, obtained by excluding the faulty sensors from the original measurements. The estimator is then rewound to $\hat{X}_{k-N}$, the last estimate confirmed to be fault-free. From there, the modified measurements are replayed to produce new state estimates, $\hat{X}'_{k-N+1}, \dots, \hat{X}'_k$	84

5.7	Comparison of trajectory and heading under nominal and fault conditions. (a–b) System under nominal conditions, without any fault injected. (c–d) System during a sensor fault: fault detection and mitigation module is in- active. (e–f) System during a sensor fault: fault detection and mitigation module is active. Fault injection times are marked in purple, and detection times are shown in green.	86
-----	--	----

List of Tables

2.1	Representative applications of Fault Tolerant Control (FTC) and Secure Estimation (SE) by system type	12
3.1	Safety factor of the pair $(\mathbf{C}, \mathbf{A})$ with different sensor protection configurations.	23
4.1	Parameters of the vehicle model in (4.1) and (4.2)	41
4.2	Parameters of the figure eight path (4.3)	42
4.3	Fault Parameter Ranges for Sensor Fault Injection	58
4.4	Summary of detection rate and detection delay for single-sensor faults, separated by threshold values for fault magnitude or drift rate. Below-threshold and above-threshold statistics are shown for each fault type and sensor. . .	61
4.5	Detection precision and recall (mean $\pm$ standard deviation) for two-sensor faults. Above threshold. Top: Precision, Bottom: Recall.	63
4.6	Detection precision and recall (mean $\pm$ standard deviation) for two-sensor faults. Below threshold. Top: Precision, Bottom: Recall.	64
4.7	Sensor and actuator frequencies for AV-21.	67
5.1	Summary of sensors and their measurement rates.	76

List of Abbreviations

AV Autonomous Vehicle [1](#)

BATADAL BATtle of the Attack Detection ALgorithms [9](#), [10](#)

FDI Fault Detection and Isolation [7](#)

FTC Fault Tolerant Control [xiii](#), [7](#), [11](#), [12](#)

GNSS Global Navigation Satellite System [66](#)

IAC Indy Autonomous Challenge [x](#), [1](#), [2](#), [4](#), [40](#), [41](#), [64](#), [66](#), [85](#), [87–90](#)

IMS Indianapolis Motor Speedway [41](#), [42](#), [64](#)

IMU Inertial Measurement Unit [66](#)

INS Inertial Navigation System [1](#), [66](#), [67](#), [69](#)

ROS Robot Operating System [76](#)

SE Secure Estimation [xiii](#), [3](#), [4](#), [6–12](#)

SMT Satisfiability Modulo Theory [8](#)

List of Symbols

- $\mathbb{R}_{\geq 0}$ The set of non-negative real numbers 5, 24
- $\mathbb{N}_0$ The set of natural numbers, including zero 5, 13–17, 19, 33, 34, 36, 37, 39
- $\mathbb{R}$ The set of real numbers xv, xvi, 5, 13–15, 17, 19, 21, 24–26, 29–32, 34, 35, 37–39, 78
- $\mathbf{0}$ The zero matrix of appropriate dimension 14, 15, 20, 24
- $\mathbf{A}_d$ The system matrix (discrete-time) 54, 72
- $\mathbf{A}$ The system matrix xiii, 5, 13–38, 53–55, 69, 72, 80, 81, 83
- $\mathbf{B}_d$ The control matrix (discrete-time) 54, 72
- $\mathbf{B}$ The control matrix 33–35, 39, 53, 54, 58, 69, 72, 80, 81, 83
- $\mathbf{C}$ The measurement matrix xiii, 13–39, 53, 54, 58, 69, 72, 80, 81, 83
- $\mathbf{I}$ The identity matrix 17, 18, 21, 23, 24, 37, 54, 81, 82
- $\mathbf{L}$ The observer gain matrix 14
- $\mathbf{P}$ The sensor protection matrix 16–27, 29–34, 37, 54, 82, 83
- $\hat{\mathbf{x}}$ The state estimate vector 29–34, 38, 39, 83
- $\mathbf{e}$ The measurement corruption vector 13–17, 19, 20, 29–34, 83
- $\mathbf{u}$ The control input vector 33–35, 39, 53, 54, 72, 80, 81, 83
- $\mathbf{x}$ The state vector 5, 13–16, 19, 20, 28–34, 36, 53, 54, 69, 72, 80, 81, 83
- $\mathbf{y}$ The measurement vector 13, 14, 16, 28–35, 38, 39, 53, 54, 69, 72, 80, 81, 83

$\mathcal{P}$ The power set [xvi](#), [5](#)

supp The support (non-zero elements) of a vector [5](#), [29–31](#), [36](#), [38](#)

c A function $\mathcal{P}(S) \rightarrow \mathbb{R}$ that assigns a cost to protecting a set of sensors S [24](#), [54](#), [73](#), [82](#)

L The wheelbase of the vehicle [41](#), [50](#), [69](#)

N The window size for the detector [58](#)

$a_{\text{lat,max}}$ The maximum lateral acceleration of the vehicle [47](#)

a_{max} The maximum acceleration of the vehicle [41](#)

a_{min} The minimum acceleration of the vehicle [41](#)

α The angle to the target point, relative to the vehicle’s heading [50](#)

a^* The desired acceleration of the vehicle [50](#)

a The acceleration of the vehicle [41](#)

$\hat{\delta}$ The estimated steering angle of the vehicle [48](#), [50](#)

δ_{max} The maximum steering angle of the vehicle [41](#)

δ^* The desired steering angle of the vehicle [50](#), [51](#)

δ The steering angle of the vehicle [41](#), [42](#), [48](#), [50](#), [69](#), [72](#)

ω_{δ}^* The desired steering rate of the vehicle [50](#)

λ_{ϵ} A scaler to adjust the detector’s tolerance to modelling errors [xi](#), [58–61](#)

ϵ The noise threshold for the detector [58](#), [59](#), [61](#), [73](#), [83](#)

κ The curvature of the path [47](#)

$d_{\text{lookahead}}$ The lookahead distance [49](#), [50](#)

T The sampling time [54](#), [72](#)

ω_{δ} The steering rate of the vehicle [41](#)

$\hat{\theta}$ The estimated heading angle of the vehicle [48](#)

θ The heading angle of the vehicle [xi](#), [41](#), [48](#), [62](#), [69](#), [72](#)
 $\hat{v}$ The estimated velocity of the vehicle [48–50](#)
 $v_{\max}$ The target maximum velocity of the vehicle [41](#), [47](#)
 v^* The desired velocity of the vehicle [50](#)
 v The velocity of the vehicle [41](#), [48](#), [50](#), [53](#), [69](#), [72](#)
 $\hat{x}$ The estimated x coordinate [48–51](#), [53](#), [80](#)
 x The x coordinate [x](#), [40–42](#), [44](#), [48–51](#), [69](#), [72](#)
 $\hat{y}$ The estimated y coordinate [48–51](#), [53](#), [80](#)
 y The y coordinate [x](#), [40–42](#), [44](#), [48–51](#), [69](#), [72](#)

Chapter 1

Introduction

Modern systems rely extensively on sensor measurements to estimate system state and support critical decisions, particularly in safety-critical domains such as aviation, robotics, and autonomous vehicles. Ensuring the reliable operation of these systems is challenging, especially since sensor failures are difficult to anticipate or fully capture during testing. Undetected sensor faults can lead to incorrect state estimates and unsafe control actions, sometimes with catastrophic outcomes.

For example, in the tragic 2018 Lion Air Flight 610 crash, a malfunctioning angle-of-attack sensor repeatedly fed incorrect data to the aircraft’s automated flight control system, causing persistent nose-down commands that led to the airplane crashing into the Java Sea, killing all 189 people onboard [17]. Similarly, during the 2025 Potomac River mid-air collision near Reagan National Airport, discrepancies between the helicopter’s radio and barometric altimeters may have misled the pilots, causing the helicopter to fly at an incorrect altitude and contributing to a collision with a commercial airliner, resulting in 67 fatalities [8].

Autonomous Vehicles (AVs), which heavily depend on sensor measurements and automated decision-making, are similarly vulnerable to these types of faults. The author’s team experienced this firsthand during a time trial at the IAC, where severe engine vibrations and inadequate mechanical isolation caused the heading signal from the Inertial Navigation System (INS) to drift by over 60 degrees while still reporting low internal uncertainty. Initially rejected, the faulty measurements were eventually accepted by the estimator, causing the autonomous vehicle to steer off course and crash into a wall (see Figures 1.1 and 1.2).

These examples illustrate a common pattern: sensor failures often manifest in subtle and unexpected ways, even as the sensors continue to report low internal uncertainty.

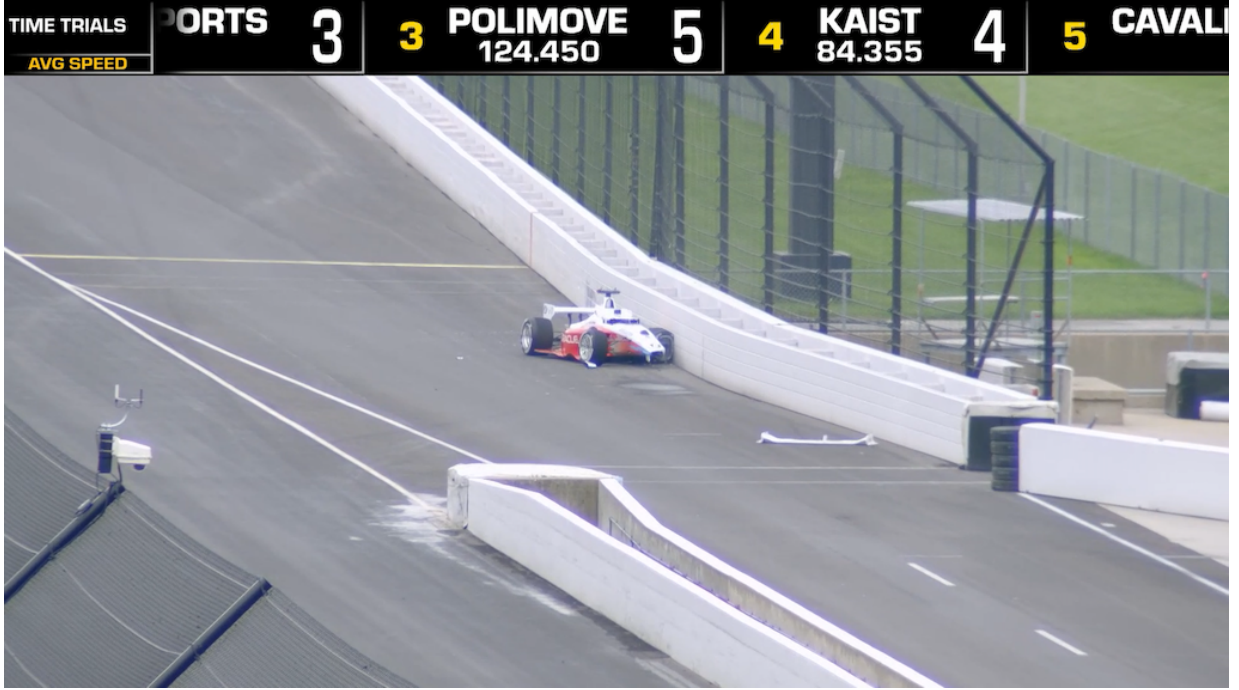


Figure 1.1: A screenshot from the live broadcast of the IAC time trial where the vehicle crashed into a wall due to a sensor malfunction [26].

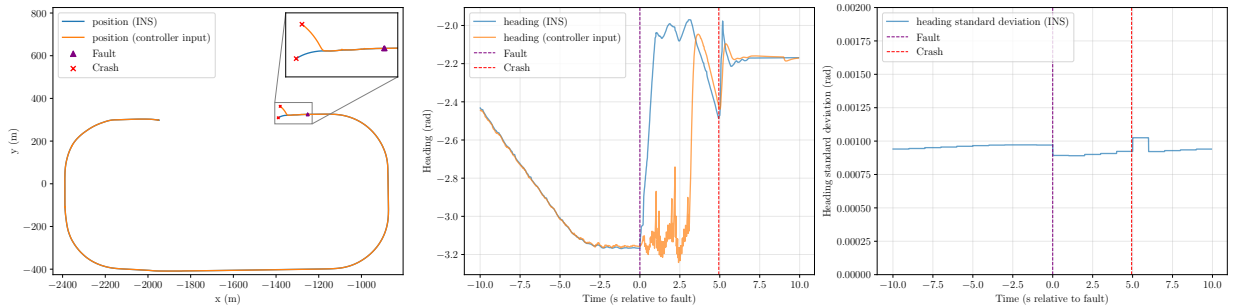


Figure 1.2: A summary of the data from the IAC crash, showing the bird's-eye view of the crash, the heading over time, and the heading standard deviation over time.

Because these health metrics are based on internal diagnostics rather than external validation, they can give a false sense of reliability and delay fault detection. These incidents highlight the need for sensor malfunction detection algorithms that don't place restrictive assumptions on failure modes.

1.1 Contributions

This thesis addresses the challenge of detecting and mitigating sensor faults in autonomous systems. Our contributions span theoretical foundations, algorithmic development, and empirical validation, with an emphasis on applying these ideas to real-world systems through a structured notion of sensor protection. The key contributions are summarized below.

1.1.1 Sparse Observability with Sensor Protection

We propose a framework to analyze the importance of individual sensors by evaluating their role in system fault tolerance. Extending the concept of s -sparse observability from the SE literature, we introduce a model of *sensor protection*, where certain sensors are assumed to be immune to faults—either through hardware redundancy or other design choices. This binary protection model allows us to reason formally about fault-tolerant estimation and guides sensor selection strategies for improved robustness. We further generalize the framework to time-varying systems, making it suitable for a broad range of autonomous platforms.

1.1.2 Optimal Sensor Protection Strategy

We propose a method for finding the most important sensors in terms of fault tolerance. This method is formulated as an optimization task: identify the most critical sensors to protect given a fixed budget. While the problem is NP-hard, it can be solved offline and is computationally tractable for systems with a modest number of sensors. This approach allows system designers to prioritize resources toward enhancing the fault tolerance of the most influential sensors.

1.1.3 Fault Detection and Real-Time Mitigation

We extend a SE algorithm from the linear time-invariant setting to handle time-varying systems via linearization. This enables fault detection in nonlinear autonomous systems. The proposed method is validated in three settings: (1) simulation experiments under diverse fault conditions, (2) a retrospective case study involving a real-world IAC racing incident, and (3) a real-time deployment on a physical robot. In the final scenario, we demonstrate a simple, mostly plug-and-play mitigation strategy that enables the robot to continue operating safely despite sensor faults.

A paper detailing the proposed sensor-protection framework and its simulation-based evaluation (Chapters 3 and 4) has been accepted for presentation at the 2025 IEEE Conference on Control Technology and Applications (CCTA 2025) [69].

1.2 Organization

The remainder of this thesis is organized as follows:

- **Chapter 2** provides background on relevant work in SE and related fields. It outlines the limitations of existing methods and motivates the proposed contributions.
- **Chapter 3** presents the core theoretical developments. It introduces the formal system model, extends the concept of sparse observability to incorporate protected sensors, and defines the safety factor as a quantitative measure of fault tolerance. It also formulates the Optimal Sensor Protection Problem and the Measurement Corruption Detection Problem, and introduces modifications needed for time-varying systems.
- **Chapter 4** applies the proposed methods to a car-like autonomous vehicle in simulation. It uses the Optimal Sensor Protection Problem to identify the most important sensors in terms of fault tolerance and demonstrates the effectiveness of the fault detection algorithm introduced in Chapter 3 under a range of injected sensor faults. A case study based on the IAC crash is also presented.
- **Chapter 5** describes the implementation of the proposed algorithm on a skid-steer mobile robot platform. It details the hardware and software architecture, presents the important sensors identified by the Optimal Sensor Protection Problem, and shows the integration of the fault detection algorithm with a mitigation strategy into an existing control pipeline.

- **Chapter 6** summarizes the key contributions, discusses the main findings, and outlines directions for future work, including extensions to parameter tuning methods, additional applications, and real-world implementation challenges such as multi-rate sampling.

1.3 Notation

Throughout this thesis, we use the following notation:

- Scalars are denoted by lowercase letters (e.g., $x \in \mathbb{R}$).
- Vectors are denoted by bold lowercase letters (e.g., $\mathbf{x} \in \mathbb{R}^n$).
- Matrices are denoted by bold uppercase letters (e.g., $\mathbf{A} \in \mathbb{R}^{n \times n}$).
- Sets are denoted by uppercase calligraphic letters (e.g., M, P).
- Given a vector $\mathbf{v}$, $\mathbf{v}_i$ denotes its i -th component.
- Given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and a set of row indices $S \subseteq \{1, \dots, m\}$, $\mathbf{A}_S$ denotes the submatrix of $\mathbf{A}$ consisting of the rows indexed by S .
- $\mathbb{R}$ denotes the set of real numbers.
- $\mathbb{N}_0$ denotes the set of natural numbers including zero.
- $\mathbb{R}_{\geq 0}$ denotes the set of non-negative real numbers.
- $\mathcal{P}(S)$ denotes the power set of set S .
- For a vector $\mathbf{v}$, $\text{supp}(\mathbf{v})$ denotes its support (the set of indices of non-zero elements).
- For a set S , $|S|$ denotes its cardinality.
- $\emptyset$ denotes the empty set.

Additional notation is introduced as needed throughout the thesis. For a comprehensive list of symbols and abbreviations, please refer to the List of Symbols and List of Abbreviations at the beginning of this thesis.

Chapter 2

Related Works

This thesis addresses the problem of detecting sensor failures in autonomous systems using a Secure Estimation (SE) framework. SE focuses on estimating the state of a dynamical system in the presence of adversarial or arbitrarily corrupted sensor measurements. Situated at the intersection of control theory, signal processing, and cybersecurity, the field has developed methods that offer provable guarantees under broad and unstructured fault models. This chapter reviews the theoretical foundations and practical advancements of SE, providing context for the contributions presented in this thesis.

2.1 Theoretical Development

To understand the advancements in SE, it is essential to first examine its theoretical evolution, beginning with foundational approaches in fault-tolerant control and robust estimation.

2.1.1 Early Approaches

Early approaches to fault tolerance employed hardware redundancy—using multiple sensors to measure the same physical quantity [47]. A well-known example is N-modular redundancy, in which N identical modules operate in parallel, and a majority voting mechanism selects the output. While effective at masking faults, the hardware costs associated with such redundancy can be prohibitive [59]. Consequently, research shifted toward analytical redundancy, where faults are detected using mathematical models of the system

rather than duplicate hardware [47]. Foundational work by Beard, Willsky, and others in the 1970s and 1980s formalized these ideas and established the basis for model-based Fault Detection and Isolation (FDI) [6, 14, 65]. Driven by advances in model-based FDI and increased computational capabilities, the development of robust state estimation and FDI techniques accelerated through the 1990s. Notably, in their book [53], Patton, Frank, and Clark curated a broad spectrum of emerging approaches to analytical redundancy, encompassing observer-based methods, parity space techniques, and probabilistic frameworks such as Markovian models. Efforts to consolidate and standardize terminology were subsequently advanced in [11, 54]. These works clearly categorized fault-tolerant control into two main approaches: passive Fault Tolerant Control (FTC), where control systems are inherently designed to withstand certain faults, and active FTC, which integrates FDI with supervisory mechanisms for fault detection and control system reconfiguration.

Traditional FTC and robust estimation methods laid the groundwork by addressing random errors and structured fault models (e.g., bounded magnitudes or known locations) [7, 71]. However, these techniques are often tailored to specific systems and lack provable guarantees under general scenarios. This gap motivated the emergence of SE, a model-based approach aiming to establish rigorous theoretical guarantees under general and potentially adversarial sensor corruption scenarios.

2.1.2 Secure Estimation for Linear Time-Invariant Systems

Fawzi *et al.* laid the foundations for this significant development, initially presenting their core concepts for discrete-time linear time-invariant (LTI) systems in a 2011 conference paper [20], and subsequently expanding upon them in their seminal journal article [21]. They modeled sensor corruptions as additive measurement disturbances of arbitrary form and magnitude, establishing necessary and sufficient conditions under which the true system state could be uniquely recovered despite sensor faults. Drawing inspiration from compressed sensing, they proposed an l_0 -optimization-based estimation algorithm guaranteeing exact state recovery, provided the number of compromised sensors remained below a certain threshold. Additionally, they proposed an l_1 relaxation of the l_0 problem, which is convex and can be solved efficiently.

Building on this, Shoukry and Tabuada [61] coined the term *s-sparse observability*, providing a concise characterization of the conditions needed for secure state reconstruction under sparse sensor attacks. In parallel, Chong *et al.* [13] extended the underlying observability analysis to continuous-time LTI systems, offering an equivalent proof of the necessary and sufficient conditions for exact state recovery. In addition, they proposed

practical estimation strategies for continuous-time systems, including sliding-window and switched observers capable of tolerating a fixed number of adversarial sensor corruptions.

While the computational hardness of secure state reconstruction was hinted at in earlier works such as [20], Mao *et al.* [39] formally proved that the secure state reconstruction problem is NP-hard in general, and that checking the sparse observability condition is coNP-complete. They introduced the notion of *eigenvalue observability*—a stronger, efficiently verifiable condition—and showed that systems satisfying sufficient eigenvalue observability can be estimated securely in polynomial time.

2.1.3 Secure Estimation for Noisy Systems

While early results focused on exact recovery under idealized conditions, later research addressed the more realistic setting where process and measurement noise are present. Notably, Pajic *et al.* (2017) [48] developed an SE framework for noisy discrete-time LTI systems under bounded disturbances and sparse sensor attacks. Similar to [21], they formulated the problem as an l_0 optimization problem along with a convex l_1 relaxation. Notably, they established an explicit bound on the estimation error when using the optimization-based algorithms, showing that some performance guarantees hold even in the presence of process and measurement noise.

Around the same time, additional algorithms for SE emerged, such as the Satisfiability Modulo Theory (SMT)-based observer design [60], a graph-search based algorithm [38], a set partitioning based algorithm [4], and a Kalman decomposition-based partial observer architecture [35].

In 2018, Nakahira *et al.* [45] and An *et al.* [3] independently introduced the concept of s -detectability, showing that it provides similar guarantees to fault tolerance as s -sparse-observability, but is a weaker condition, thus applies to a larger class of systems. Additionally, both groups proposed algorithms for secure state estimation under noise. Nakahira *et al.* used multiple Luenberger observers with residual checks and a fusion step to reject attacked sensors, providing performance guarantees in various norms. An *et al.* designed an adaptive observer with a switching mechanism that disables corrupted channels based on a performance index, offering a low-complexity alternative. Both methods rely on $2s$ -detectability to ensure bounded estimation error.

2.1.4 Secure Estimation for Linear Time-Varying and Nonlinear Systems

More recently, considerable effort has been devoted to extending SE algorithms to linear time-varying (LTV) and nonlinear systems. Building on their earlier results for continuous-time linear systems, Chong *et al.* [12] proposed an SE framework for nonlinear systems. They proved that the sparse observability condition enabling SE in linear systems [13] extends to certain classes of nonlinear systems, and introduced an estimation algorithm based on a bank of input-to-state stable observers. Kim *et al.* [30, 32] observed that attack detection (but not necessarily identification) required a weaker sparse observability condition than state reconstruction. To mitigate the combinatorial blow-up of traditional secure-estimation schemes, they decomposed the system into smaller subsystems—each preserving sparse observability—and performed attack identification locally within each, greatly reducing the number of observers required.

2.1.5 Additional Lines of Work

Beyond the core categories of SE for various system types, several theoretical extensions explore complementary directions.

One such line of research focuses on *distributed secure estimation*, where a network of agents—each responsible for a local region of the system—collaborates to detect and isolate attacks using local measurements, limited model knowledge, and inter-agent communication [5, 31, 52]. These approaches aim to reduce computational complexity while preserving attack identifiability by leveraging system structure and communication among neighboring agents. This is especially relevant in large-scale systems where centralized estimation would be computationally prohibitive.

A further challenge arises from asynchronous and non-uniform sensor sampling. In real-world systems, sensors may sample at different times and even transmit data with manipulated timestamps. Li *et al.* [37] proposed an SE framework for such conditions. Their method uses local estimators based on observability space decomposition and a median-based fusion rule to produce a global estimate that tolerates a bounded number of corrupted and asynchronous sensors.

Efforts to standardize the evaluation of attack detection algorithms have also emerged. The BATtle of the Attack Detection ALgorithms (BATADAL) [64] benchmark was developed for the study of attack detection in water distribution networks. Built around a simulated water distribution network—C-Town—the dataset includes supervisory control

and data acquisition (SCADA) measurements under both normal operation and a range of simulated cyber-attacks, such as replay attacks and malicious actuator manipulation. Participating teams were evaluated on their ability to detect attacks and minimize false alarms, and optionally identify compromised components. Submitted algorithms were evaluated using clearly defined metrics for time-to-detection and classification accuracy, enabling meaningful comparison across diverse approaches. While not based on SE, the winner of BATADAL [25] was the only model-based submission, outperforming data-driven approaches.

Another emerging direction considers resource-aware security investment in the context of secure state estimation. Shinohara and Namerikawa [58] introduced the Optimal Security Investment Problem (OSIP), which formalizes the trade-off between the cost of assigning security-aware actions to sensors and the resulting improvement in system resilience. Each sensor can be configured with either a low-cost, nonsecurity-aware action—under which it may be compromised—or a higher-cost, security-aware action that is assumed to perfectly prevent the sensor from being compromised. The authors quantify resilience using a generalized sparse observability index and define an objective function that balances this resilience metric against the total investment cost. This line of inquiry integrates economic reasoning into resilient system design, enabling system operators to make principled decisions about how to allocate limited security resources for maximal impact.

These additional directions illustrate the breadth of current research in SE, addressing challenges such as distributed coordination, asynchronous sensing, standardized evaluation, and resource-aware security design. While this thesis focuses on centralized SE for linear time-invariant systems and linearized nonlinear systems, particularly building upon the theoretical foundations introduced in [13, 20, 21, 48], the broader context presented here highlights the practical considerations and system-level trade-offs that increasingly shape research in SE.

2.2 Applications

In the 1970s, researchers began to apply the concepts of fault tolerance to control systems. Notably, as a part of the NASA Digital Fly-By-Wire (DFBW) program, Desai and Deckert *et al.* [16, 18] deployed a sensor failure detection and identification system for the F-8 aircraft using dual-redundant sensors and early developments in analytical redundancy. By comparing pairs of redundant sensors, the system can detect failures based on mismatch in their readings. Then, the system utilizes kinematic and functional relationships between sensors to identify the failed sensor.

In the 1980s, the Self-Repairing Flight Control System (SRFCS) program [63] built on the analytical redundancy approach validated in the F-8 DFBW project, focusing on control reconfiguration after actuator failures. This approach was demonstrated on the F-15 aircraft, where it maintained control authority despite actuator malfunctions.

Other notable examples of FTC include applications to unconventional vehicles such as the X-36 tailless aircraft [9], simulation studies of actuator fault detection for reusable rocket engines [44], and implementations in automotive drive-by-wire systems [28]. A more comprehensive list of FTC applications can be found in [70].

Recent developments in SE have also led to a wide range of simulation studies and some real-world deployments. Early works [20, 61] validated optimization-based and observer-based algorithms on random system models, demonstrating resilience against sensor attacks. Subsequent studies extended SE to simulated power systems and water networks [2, 37, 50, 51, 60]. In robotics, SE has been applied to unmanned ground vehicles (UGVs) for longitudinal control under attack [48, 61], hexarotor unmanned aerial vehicles (UAVs) for fault-tolerant control [40], and distributed estimation in simulated vehicle platoons [24, 62].

A summary of representative applications of FTC and SE is presented in Table 2.1. FTC methods are typically designed *ad hoc* for specific systems and are most often deployed in real-world applications. In contrast, SE offers a more general framework with provable guarantees, but its adoption has been largely limited to simulation studies. This thesis aims to bridge this gap by demonstrating the practical use of SE algorithms on real systems.

Table 2.1: Representative applications of FTC and SE by system type

System	Reference(s)	Validation	Notes
Aerospace, Automotive	[9, 16, 18, 28, 44, 63]	S, D	Sensor failure detection, identification, and control reconfiguration in drive-by-wire vehicles and aircrafts, and simulated rocket engines
Power Systems	[2, 36, 50, 51, 60]	S	SE and attack mitigation in large-scale power networks (9-bus, 14-bus, multi-area AGC); distributed and scalable observer designs
Water Networks	[51]	S	Distributed SE in water distribution networks
Mobile Robots	[24, 40, 48, 61, 62]	S, D	SE and control for UGVs (hardware and simulation), and distributed estimation/control in vehicle platooning (simulation)
Random Systems	[20, 61]	S	Fundamental algorithm validation under sparse sensor attack models

S: Validation performed in a simulated environment, not on physical hardware.

D: Validation performed on physical hardware.

Chapter 3

Methodology

In this chapter, we assess and enhance the robustness of discrete-time linear time-invariant (LTI) systems against measurement corruptions, such as sensor failures or adversarial attacks. We start by formalizing the system model and examining the limitations of standard state estimation algorithms in the presence of corrupted measurements. Recognizing these limitations, we present the concept of s -sparse observability from the study of secure estimation, which allows us to quantify a system's resilience to sensor faults. Using this framework, we introduce the notions of protected sensors and safety factor, to allow for more granular analysis of fault tolerance. Building on these concepts, we show how a technique from secure estimation, known as secure state-reconstruction, can be adapted to serve as a sensor fault detection mechanism. Finally, we show how this mechanism can be integrated into existing state estimation algorithms to enhance system resilience.

3.1 Sparse observability

We consider discrete-time, linear, time-invariant systems with measurement corruptions of the following form:

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{A}\mathbf{x}[k], & k \in \mathbb{N}_0, & \quad \mathbf{x}[0] = \mathbf{x}_0, \\ \mathbf{y}[k] &= \mathbf{C}\mathbf{x}[k] + \mathbf{e}[k] \end{aligned} \tag{3.1}$$

where k is the time index, $\mathbf{x}[k] \in \mathbb{R}^n$, $\mathbf{y}[k] \in \mathbb{R}^p$, and $\mathbf{e}[k] \in \mathbb{R}^p$ are the state, measurement, and corruption vectors at time k , respectively, and $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{C} \in \mathbb{R}^{p \times n}$ are the system and measurement matrices, respectively.

The signal $\mathbf{e} : \mathbb{N}_0 \rightarrow \mathbb{R}^p$ is an additive corruption which could model, for instance, sensor failure and/or adversarial attacks. Standard state estimation algorithms for this class of systems, e.g., steady-state Kalman filter, Luenberger observers, have the general form

$$\begin{aligned}\hat{\mathbf{x}}[k+1] &= \mathbf{A}\hat{\mathbf{x}}[k] + \mathbf{L}(\hat{\mathbf{y}}[k] - \mathbf{y}[k]) \\ \hat{\mathbf{y}}[k] &= \mathbf{C}\hat{\mathbf{x}}[k].\end{aligned}$$

In the absence of measurement corruptions, $\mathbf{e}[\cdot] \equiv \mathbf{0}$, such algorithms asymptotically converge to the system state $\mathbf{x}$ if, and only if there exists an $\mathbf{L}$ so that $\mathbf{A} + \mathbf{LC}$ is stable. Equivalently, if, and only if, the pair $(\mathbf{C}, \mathbf{A})$ is detectable.

However, the performance of these algorithms suffers when there are measurement corruptions, sometimes resulting in highly inaccurate state estimates. Identifying and mitigating the effects corrupted sensors could potentially enhance the performance of these critical algorithms as well as control algorithms that make use of state estimates.

In this thesis, we do not make any assumptions about the function space that $\mathbf{e} : \mathbb{N}_0 \rightarrow \mathbb{R}^p$ belongs to and we define, for $k_0, k_1 \in \mathbb{N}_0$ with $k_0 \leq k_1$,

$$K_{[k_0, k_1]}(\mathbf{e}) := \{i \mid \exists k \in [k_0, k_1], \mathbf{e}_i[k] \neq 0\} \subseteq \{1, 2, \dots, p\}$$

to be the set of sensor indices that have been corrupted over $k \in [k_0, k_1]$.

We now introduce the concept of s -sparse observability, which allows us to quantify a system's ability to handle measurement corruptions. We make use of the following notation throughout this thesis: let $S \subseteq \{1, 2, \dots, p\}$ be a set of sensors and let $\mathbf{C}_S$ denote the submatrix of $\mathbf{C} \in \mathbb{R}^{p \times n}$ formed by selecting the rows indexed by S .

Definition 1. ([39, 61]) *The **sparse observability index** of the pair $(\mathbf{C}, \mathbf{A})$, denoted by $\sigma_o(\mathbf{C}, \mathbf{A})$, is the largest integer s such that for every set $M \subseteq \{1, 2, \dots, p\}$ with $|M| \leq s$, the pair $(\mathbf{C}_{\{1, 2, \dots, p\} \setminus M}, \mathbf{A})$ is observable. The pair $(\mathbf{C}, \mathbf{A})$ is **s-sparse observable** if $\sigma_o(\mathbf{C}, \mathbf{A}) = s$.*

Remark 1. Recall that a pair $(\mathbf{C}, \mathbf{A})$ is observable if, and only if, the observability matrix

$$\begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{n-1} \end{bmatrix}$$

has full column rank (i.e., rank n). Equivalently, the unobservable subspace of the pair $(\mathbf{C}, \mathbf{A})$ is $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{C}\mathbf{A}^k\mathbf{x} = \mathbf{0}, k \in [0, n-1]\}$, which is trivial (i.e., $\emptyset$) if, and only if, the pair is observable.

◆

Example 1. Consider the following pair:

$$\mathbf{A} = \begin{bmatrix} 1 & 0.1 & 0 & 0 \\ 0 & 1 & 0.1 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The sparse observability index of the pair $(\mathbf{C}, \mathbf{A})$ is 1, because we can remove up to 1 arbitrary row of $\mathbf{C}$ and still maintain observability. This pair is not 2-sparse observable because removing rows 1, 2 of $\mathbf{C}$ results in an unobservable pair.

▲

A naive algorithm to determine the sparse observability index of a pair $(\mathbf{C}, \mathbf{A})$ is to check all possible sensor combinations (combinations of rows of $\mathbf{C}$) and verify observability for each. This requires up to 2^p observability tests. In [21], the authors suggest that it's very unlikely that there is a more efficient way. Nevertheless, determining the sparse observability index for a given pair is usually a one-time operation, and for systems with a small number of sensors, like ones discussed in this thesis, this is a feasible approach.

In [13], the authors proved the following theorem that connects sparse observability to the system's resilience to measurement corruptions.

Theorem 1 ([13, Theorem 1]). *For every pair $(\mathbf{C}, \mathbf{A})$ and $s \in \mathbb{N}_0$, the following are equivalent:*

1. $\forall M \subseteq \{1, 2, \dots, p\}$ with $|M| \leq 2s$, $(\mathbf{C}_{\{1,2,\dots,p\} \setminus M}, \mathbf{A})$ is observable.
2. For every initial conditions $\mathbf{x}_1[0], \mathbf{x}_2[0] \in \mathbb{R}^n$ and corruption vectors $\mathbf{e}_1, \mathbf{e}_2 : \mathbb{N}_0 \rightarrow \mathbb{R}^p$ with $|K_{[0,n-1]}(\mathbf{e}_1)| \leq s$ and $|K_{[0,n-1]}(\mathbf{e}_2)| \leq s$, if $\mathbf{C}\mathbf{A}^k\mathbf{x}_1[0] + \mathbf{e}_1[k] = \mathbf{C}\mathbf{A}^k\mathbf{x}_2[0] + \mathbf{e}_2[k]$ for $k \in [0, n-1]$, then $\mathbf{x}_1[0] = \mathbf{x}_2[0]$.

In other words, Theorem 1 states that if the pair $(\mathbf{C}, \mathbf{A})$ is $2s$ -sparse observable, then it's possible to uniquely determine the states from the measurements, even when up to s sensors are corrupted, and vice versa. This theorem is consistent with the results in [13, 61].

Example 2. The pair in Example 1 is 1-sparse observable, thus by Theorem 1, we cannot uniquely determine the initial state from the measurements when there is measurement corruption.

▲

Theorem 1 establishes a link between s -sparse observability and the system’s resilience to measurement corruptions. This theoretical framework has influenced the design of secure estimators in several studies [13, 21, 48, 61]. Nevertheless, s -sparse observability represents a worst-case scenario analysis. In considering all sensors as equally likely to fail or be attacked, s -sparse observability ignores the reality that some sensors are more resilient to faults than others. In practice, certain sensors may prove especially critical in maintaining observability, and system designers will put in more resources to ensure their reliability. For instance, while the velocity of an object can be inferred from its position measurements over time, assuming no external forces are acting on it, the converse—deriving position from velocity—is not feasible. Recognizing the varying importance of sensors, we can enhance system resilience by protecting these critical sensors against faults. In the following section, we introduce the concept of “sensor protection” as a binary model of a sensor’s resilience to failure and begin to explore how safeguarding key sensors can improve a system’s ability to handle measurement corruptions.

3.2 Protected Sensors

In this section, we extend (3.1) to include the concept of “sensor protection” or “sensor reliability.”¹ A sensor is considered protected if it is immune to faults, whether due to physical security measures, redundancy, error-correcting codes, or other protective strategies. We intentionally abstract away the specific methods by which these sensors are protected to concentrate on the theoretical implications of their immunity. This abstraction allows us to focus on analyzing the system’s ability to handle faults in the remaining unprotected sensors without getting entangled in implementation details. Our primary interest lies in understanding how the protection of certain sensors enhances overall system observability and resilience to measurement corruptions.

Let $\mathbf{P} \in \{0, 1\}^{p \times p}$ be a binary diagonal matrix. Consider the following system:

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{A}\mathbf{x}[k], & k \in \mathbb{N}_0, & \quad \mathbf{x}[0] = \mathbf{x}_0, \\ \mathbf{y}[k] &= \mathbf{C}\mathbf{x}[k] + \mathbf{P}\mathbf{e}[k] \end{aligned} \tag{3.2}$$

¹This work overlaps with the independent work by Shinohara *et al.* [58], who introduce a similar notion of sensor protection but employ a different metric (Section 3.3).

where $\mathbf{e}[k] \in \mathbb{R}^p$ is the corruption vector at time k and $\mathbf{P}\mathbf{e}[k]$ models the effect of sensor protection, where $\mathbf{P}_{ii} = 0$ if sensor i is protected and $\mathbf{P}_{ii} = 1$ otherwise.

For convenience, we define the set-equivalent of $\mathbf{P}$ as $P = \{i \mid \mathbf{P}_{ii} = 0\}$. Since $\mathbf{P}$ is a binary diagonal matrix, P is uniquely determined by $\mathbf{P}$ and vice versa. In this thesis, we will use P and $\mathbf{P}$ interchangeably except when the distinction is important.

Definition 2. Let $P \subseteq \{1, 2, \dots, p\}$. The **sparse observability index with sensor protection on P** of the pair $(\mathbf{C}, \mathbf{A})$, denoted by $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P})$, is the largest integer s such that for every set $M \subseteq \{1, 2, \dots, p\} \setminus P$ with $|M| \leq s$, the pair $(\mathbf{C}_{\{1, 2, \dots, p\} \setminus M}, \mathbf{A})$ is observable. The pair $(\mathbf{C}, \mathbf{A})$ is **s -sparse observable with sensor protection on P** if $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P}) = s$.

The above definition extends the concept of sparse observability (Definition 1) by incorporating the notion of sensor protection. By designating a subset of sensors P as protected—meaning they are immune to faults—we effectively partition the sensor set into protected and unprotected sensors. The sparse observability index with sensor protection, $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P})$, quantifies the system’s ability to maintain observability despite faults in the unprotected sensors.

Remark 2. Consider the following decision problem: given a pair $(\mathbf{C}, \mathbf{A})$ and an integer $s \in \mathbb{N}_0$, determine whether $\sigma_o(\mathbf{C}, \mathbf{A}) \geq s$. This is the decision problem of determining sparse observability (Definition 1), which is known to be coNP-complete [39].

Now, consider a generalization of this problem that incorporates sensor protection: given a pair $(\mathbf{C}, \mathbf{A})$, a protection matrix $\mathbf{P} \in \{0, 1\}^{p \times p}$, and an integer $s \in \mathbb{N}_0$, determine whether $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P}) \geq s$. We will show that this problem is also coNP-complete.

First, this problem is in coNP because a “no” instance can be efficiently certified. Specifically, one can provide a set of unprotected sensors $M \subseteq \{1, \dots, p\} \setminus P$ of size greater than s such that removing the corresponding rows from $\mathbf{C}$ makes the system unobservable. This can be verified by constructing the observability matrix and checking its rank, which is a polynomial-time operation.

Second, we show that this problem is coNP-hard by reducing from the original sparse observability problem. Given an instance $(\mathbf{C}, \mathbf{A}, s)$ of the original problem, we construct an instance of the sparse observability with sensor protection problem by setting $\mathbf{P} = \mathbf{I}_p$, i.e., no sensors are protected. In this case, the constructed instance has the same observability property as the original. Therefore, the sparse observability with sensor protection problem is at least as hard as the original sparse observability problem, implying coNP-hardness.

Combining both parts, we conclude that the sparse observability problem with sensor protection is coNP-complete. ◆

Protecting even a small number of sensors can dramatically improve the sparse observability index. We illustrate this with the following example.

Example 3. Consider the same pair $(\mathbf{C}, \mathbf{A})$ from Example 1, where

$$\mathbf{A} = \begin{bmatrix} 1 & 0.1 & 0 & 0 \\ 0 & 1 & 0.1 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

When we protect no sensors, i.e., $P = \emptyset$ and $\mathbf{P} = \mathbf{I}$, the pair is 1-sparse observable as shown in Example 1. By Theorem 1, this means the system cannot handle any measurement corruptions while maintaining observability. Now, suppose we protect sensor 1, so that

$$P = \{1\}, \quad \mathbf{P} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

In this case, the set M from Definition 2 is now restricted to the subsets of the unprotected sensors, $\{2, 3, 4, 5\}$. To find the sparse observability index, we need to determine the largest integer s such that the system remains observable when any s unprotected sensors fail. Consider the maximal set, $M = \{2, 3, 4, 5\}$, which includes all unprotected sensors. We examine the observability of the pair $(\mathbf{C}_{\{1,2,3,4,5\} \setminus M}, \mathbf{A}) = (\mathbf{C}_{\{1\}}, \mathbf{A})$. The observability matrix for this pair is

$$\begin{bmatrix} \mathbf{C}_{\{1\}} \\ \mathbf{C}_{\{1\}}\mathbf{A} \\ \mathbf{C}_{\{1\}}\mathbf{A}^2 \\ \mathbf{C}_{\{1\}}\mathbf{A}^3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0.1 & 0 & 0 \\ 1 & 0.2 & 0.01 & 0 \\ 1 & 0.3 & 0.03 & 0.001 \end{bmatrix}.$$

This matrix is full rank, indicating that the pair is observable even when all unprotected sensors are removed. Thus, by Definition 2, the pair $(\mathbf{C}, \mathbf{A})$ is 4-sparse observable with sensor protection on $\{1\}$. ▲

The next theorem shows that this improvement in the sparse observability index translates to an enhanced ability to handle measurement corruptions on the remaining sensors.

Theorem 2. *For every pair $(\mathbf{C}, \mathbf{A})$, $s \in \mathbb{N}_0$, and $P \subseteq \{1, 2, \dots, p\}$, the following are equivalent:*

1. *The pair $(\mathbf{C}, \mathbf{A})$ is $2s$ -sparse observable with sensor protection on P .*
2. *For every initial conditions $\mathbf{x}_1[0], \mathbf{x}_2[0] \in \mathbb{R}^n$ and corruption vectors $\mathbf{Pe}_1, \mathbf{Pe}_2 : \mathbb{N}_0 \rightarrow \mathbb{R}^p$ with $|K_{[0, n-1]}(\mathbf{Pe}_1)| \leq s$ and $|K_{[0, n-1]}(\mathbf{Pe}_2)| \leq s$, if $\mathbf{CA}^k \mathbf{x}_1[0] + \mathbf{Pe}_1[k] = \mathbf{CA}^k \mathbf{x}_2[0] + \mathbf{Pe}_2[k]$ for $k \in [0, n-1]$, then $\mathbf{x}_1[0] = \mathbf{x}_2[0]$.*

In Example 3, the pair $(\mathbf{C}, \mathbf{A})$ is 4-sparse observable with sensor protection on $\{1\}$. Thus, by Theorem 2, the pair can handle up to 2 corruptions of the unprotected sensors while still being able to uniquely determine the states from the measurements.

This is a significant improvement compared to the scenario without sensor protection, where no guarantees of tolerance can be made due to the need to handle all possible sensor corruptions. This suggests that some sensors in the pair are more important than others for maintaining observability. We will explore this idea further in Section 3.3.

The proof of Theorem 2 can be adapted from the proof of Theorem 1 from [13].

Proof. The conditions in Theorem 2 can be equivalently stated as follows:

1. $\forall M \subseteq \{1, 2, \dots, p\} \setminus P$ with $|M| \leq 2s$, $\mathbf{C}_{\{1, 2, \dots, p\} \setminus M} \mathbf{A}^k \mathbf{x}[0] = 0, k \in [0, n-1] \implies \mathbf{x}[0] = 0$.
2. $\forall \mathbf{Pe}_1, \mathbf{Pe}_2 : \mathbb{N}_0 \rightarrow \mathbb{R}^p$ with $|K_{[0, n-1]}(\mathbf{Pe}_1)| \leq s$ and $|K_{[0, n-1]}(\mathbf{Pe}_2)| \leq s$, $\mathbf{CA}^k \mathbf{x}_1[0] + \mathbf{Pe}_1[k] = \mathbf{CA}^k \mathbf{x}_2[0] + \mathbf{Pe}_2[k], k \in [0, n-1] \implies \mathbf{x}_1[0] = \mathbf{x}_2[0]$.

To show that condition 1 implies condition 2, we prove the contrapositive. Assume there exists $\mathbf{Pe}_1, \mathbf{Pe}_2 : \mathbb{N}_0 \rightarrow \mathbb{R}^p$ with $|K_{[0, n-1]}(\mathbf{Pe}_1)| \leq s$, $|K_{[0, n-1]}(\mathbf{Pe}_2)| \leq s$, such that for $k \in [0, n-1]$:

$$\mathbf{CA}^k \mathbf{x}_1[0] + \mathbf{Pe}_1[k] = \mathbf{CA}^k \mathbf{x}_2[0] + \mathbf{Pe}_2[k] \text{ and } \mathbf{x}_1[0] \neq \mathbf{x}_2[0].$$

Rearrange the equation to obtain:

$$\mathbf{CA}^k (\mathbf{x}_1[0] - \mathbf{x}_2[0]) + \mathbf{Pe}_1[k] - \mathbf{Pe}_2[k] = 0 \text{ and } \mathbf{x}_1[0] \neq \mathbf{x}_2[0].$$

Let $M = K_{[0,n-1]}(\mathbf{P}\mathbf{e}_1) \cup K_{[0,n-1]}(\mathbf{P}\mathbf{e}_2)$. Note that $\mathbf{P}\mathbf{e}_{1,\{1,2,\dots,p\}\setminus M}[k] = \mathbf{P}\mathbf{e}_{2,\{1,2,\dots,p\}\setminus M}[k] = \mathbf{0}$, where $\mathbf{e}_{i,S}[k]$ denotes the subvector of $\mathbf{e}_i[k]$ indexed by S . Then:

$$\mathbf{C}_{\{1,2,\dots,p\}\setminus M} \mathbf{A}^k (\mathbf{x}_1[0] - \mathbf{x}_2[0]) = \mathbf{0} \text{ and } \mathbf{x}_1[0] - \mathbf{x}_2[0] \neq \mathbf{0}.$$

which is the negation of condition 1.

To show that condition 2 implies condition 1, we again prove the contrapositive. Assume there exists $M \subseteq \{1, 2, \dots, p\} \setminus P$ with $|M| \leq 2s$ such that:

$$\mathbf{C}_{\{1,2,\dots,p\}\setminus M} \mathbf{A}^k \mathbf{x}[0] = \mathbf{0}, k \in [0, n-1] \text{ and } \mathbf{x}[0] \neq \mathbf{0}. \quad (3.3)$$

Choose $M_1, M_2 \subseteq \{1, 2, \dots, p\} \setminus P$ such that $|M_1| \leq s$, $|M_2| \leq s$, $M_1 \cap M_2 = \emptyset$ and $M_1 \cup M_2 = M$. Define $\mathbf{e}_1, \mathbf{e}_2$ such that $\mathbf{e}_{1,M_1}[k] = -\mathbf{C}_{M_1} \mathbf{A}^k \mathbf{x}[0]$ and $\mathbf{e}_{2,M_2}[k] = \mathbf{C}_{M_2} \mathbf{A}^k \mathbf{x}[0]$, where $\mathbf{e}_{i,M_j}[k]$ denotes the subvector of $\mathbf{e}_i[k]$ indexed by M_j . Then for $k \in [0, n-1]$:

$$\begin{aligned} \mathbf{C}_{M_1} \mathbf{A}^k \mathbf{x}[0] + \mathbf{e}_{1,M_1}[k] &= \mathbf{0}, \\ \mathbf{e}_{2,M_1}[k] &= \mathbf{0}, \\ \mathbf{C}_{M_2} \mathbf{A}^k \mathbf{x}[0] &= \mathbf{e}_{2,M_2}[k], \\ \mathbf{e}_{1,M_2}[k] &= \mathbf{0}, \end{aligned}$$

and combined with (3.3), we have:

$$\begin{aligned} \mathbf{C} \mathbf{A}^k \mathbf{x}[0] + \mathbf{e}_1[k] &= \mathbf{e}_2[k], \\ \implies \mathbf{C} \mathbf{A}^k \mathbf{x}[0] + \mathbf{e}_1[k] &= \mathbf{C} \mathbf{A}^k \mathbf{0} + \mathbf{e}_2[k]. \end{aligned}$$

Since $\mathbf{x}[0] \neq \mathbf{0}$, this is the negation of condition 2. ■

3.3 Safety Factor

As we've seen in Example 3, protecting even a small number of sensors can drastically improve the guaranteed fault tolerance of a system. In this section, we introduce the concept of "safety factor" to help us quantify the importance of sensors in maintaining observability. The safety factor provides a metric for evaluating how well a system can tolerate sensor faults, especially when certain sensors are protected, which will ultimately help us choose which sensors to protect.

Definition 3. The *safety factor* of the pair $(\mathbf{C}, \mathbf{A})$ with sensor protection on P is the value returned by the *safety factor function* $\gamma : \mathbb{R}^{p \times n} \times \mathbb{R}^{n \times n} \times \mathbb{R}^{p \times p} \rightarrow [0, 1]$:

$$\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P}) := \frac{\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P})/2 + \text{tr}(\mathbf{I} - \mathbf{P})}{p}$$

The $\text{tr}(\mathbf{I} - \mathbf{P})$ term is the number of protected sensors. It can also be written as $|P|$ when using the set-equivalent of $\mathbf{P}$.

The safety factor takes into account both the number of sensors that can be corrupted while maintaining observability (as quantified by the sparse observability index) and the number of protected sensors. When all sensors are unprotected (i.e., $P = \emptyset$), the safety factor reduces to the proportion of sensors that can be corrupted while maintaining observability:

$$\gamma(\mathbf{C}, \mathbf{A}, \mathbf{I}) = \frac{\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{I})/2}{p}.$$

Conversely, when all sensors are protected (i.e., $P = \{1, 2, \dots, p\}$), the sparse observability index $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P})$ becomes 0 (since no unprotected sensors are left to fail), and the safety factor achieves its maximum value of 1:

$$\gamma(\mathbf{C}, \mathbf{A}, \mathbf{I}) = \frac{0 + p}{p} = 1.$$

In other words, the safety factor quantifies the proportion of sensors that are either immune to corruption (due to the assumption that protected sensors are fault-free) or whose corruption can be tolerated without losing observability. The higher the safety factor, the greater the system's capacity to maintain observability despite sensor faults, making it more fault-tolerant.

Remark 3. We showed in Remark 2 that the decision problem of determining sparse observability with sensor protection is coNP-complete. This result extends naturally to the decision problem of determining whether the safety factor exceeds a given threshold.

Formally, consider the following decision problem: given a pair $(\mathbf{C}, \mathbf{A})$, a protection matrix $\mathbf{P} \in \{0, 1\}^{p \times p}$, and a threshold $\alpha \in [0, 1]$, determine whether $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P}) \geq \alpha$.

This problem is in coNP. A “no” instance, i.e., when $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P}) < \alpha$, implies that $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P}) < 2(\alpha p - |P|)$. We can produce the certificate in the same way as in Remark 2: a set $M \subseteq \{1, \dots, p\} \setminus P$ of size greater than $2(\alpha p - |P|)$ for which the system is not observable, which can be verified in polynomial time by checking observability.

To show that this problem is also coNP-hard, we reduce from the sparse observability with protection problem. Given an instance $(\mathbf{C}, \mathbf{A}, \mathbf{P}, s)$, we construct an instance of the safety factor decision problem by setting:

$$\alpha = \frac{s/2 + |\mathbf{P}|}{p}.$$

Then $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P}) \geq \alpha$ if, and only if, $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P}) \geq s$, so the safety factor decision problem is at least as hard. Hence, the decision problem of determining whether the safety factor exceeds a given threshold is coNP-hard.

Combining both parts, we conclude that the decision problem of determining whether the safety factor exceeds a given threshold is coNP-complete.

◆

In the presence of sensor protection, the safety factor provides a more intuitive and practical measure of a system's resilience to sensor faults than the sparse observability index (Definition 2). This distinction arises because sensor protection reduces the upper limit of the sparse observability index. Consequently, a system with a lower sparse observability index but substantial sensor protection may appear less robust than it actually is. By incorporating both the sparse observability index and the number of protected sensors, the safety factor provides a more holistic evaluation of fault tolerance. For this reason, in the next section, we aim to maximize the safety factor instead of the sparse observability index.

3.4 The Optimal Sensor Protection Problem

With the goal of maximizing the fault tolerance of a system in mind, we can use the safety factor to inform decisions on where to allocate resources for sensor protection.

Example 4. Consider the pair $(\mathbf{C}, \mathbf{A})$ from Example 1. Table 3.1 contains the safety factor of the pair with different sensor protection configurations. As we can see from the table, sensors 1 or 2 appear the most critical because protecting either results in the highest increase in safety factor.

▲

Table 3.1: Safety factor of the pair $(\mathbf{C}, \mathbf{A})$ with different sensor protection configurations.

P	$\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P})$	$\text{tr}(\mathbf{I} - \mathbf{P})$	$\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P})$
$\{\}$	1	0	0.10
$\{1\}$	4	1	0.60
$\{2\}$	4	1	0.60
$\{3\}$	1	1	0.30
$\{4\}$	1	1	0.30
$\{5\}$	1	1	0.30
$\{1, 2\}$	3	2	0.70
$\{1, 3\}$	3	2	0.70
$\{1, 4\}$	3	2	0.70
$\{1, 5\}$	3	2	0.70
$\{2, 3\}$	3	2	0.70
$\{2, 4\}$	3	2	0.70
$\{2, 5\}$	3	2	0.70
$\{3, 4\}$	1	2	0.50
$\{3, 5\}$	1	2	0.50
$\{4, 5\}$	1	2	0.50
$\{1, 2, 3\}$	2	3	0.80
$\{1, 2, 4\}$	2	3	0.80
$\{1, 2, 5\}$	2	3	0.80
$\{1, 3, 4\}$	2	3	0.80
$\{1, 3, 5\}$	2	3	0.80
$\{1, 4, 5\}$	2	3	0.80
$\{2, 3, 4\}$	2	3	0.80
$\{2, 3, 5\}$	2	3	0.80
$\{2, 4, 5\}$	2	3	0.80
$\{3, 4, 5\}$	1	3	0.70
$\{1, 2, 3, 4\}$	1	4	0.90
$\{1, 2, 3, 5\}$	1	4	0.90
$\{1, 2, 4, 5\}$	1	4	0.90
$\{1, 3, 4, 5\}$	1	4	0.90
$\{2, 3, 4, 5\}$	1	4	0.90
$\{1, 2, 3, 4, 5\}$	0	5	1.00

Now we propose a problem that seeks to find the optimal set of sensors to protect to maximize the system's ability to handle sensor faults. We will refer to this as the “optimal sensor protection problem”.

Problem 1 (Optimal Sensor Protection Problem). *Given a pair $(\mathbf{C}, \mathbf{A})$ and a budget $b \in \mathbb{R}_{\geq 0}$, let $c : \mathbb{R}^{p \times p} \rightarrow \mathbb{R}_{\geq 0}$ be a function that assigns a cost to protecting a set of sensors represented by a protection matrix $\mathbf{P}$ and satisfies $c(\mathbf{0}) = 0$. Solve the following optimization problem:*

$$\begin{aligned} & \underset{\mathbf{P} \in \text{diag}(\{0, 1\}^p)}{\text{maximize}} && \gamma(\mathbf{C}, \mathbf{A}, \mathbf{P}) \\ & \text{subject to} && c(\mathbf{P}) \leq b. \end{aligned} \tag{3.4}$$

Problem 1 is a combinatorial optimization problem that seeks to find the optimal set of sensors to protect to maximize an existing system's ability to handle sensor faults. The solution to this problem can inform decisions on where budget should be allocated to maximize resilience.

Example 5. Consider the pair $(\mathbf{C}, \mathbf{A})$ from Example 4. Table 3.1 shows the safety factor of the pair with different sensor protection configurations. Suppose we define the cost function $c(\mathbf{P}) = \text{tr}(\mathbf{I} - \mathbf{P}) = |P|$ and set the budget $b = 1$. Then the optimal sensor protection set is either $\{1\}$ or $\{2\}$, both of which achieve a safety factor of 0.60. This result suggests that under the given budget constraint, protecting either sensor 1 or sensor 2 is the best strategy for maximizing the system's resilience to sensor faults.

▲

Remark 4. *Consider the decision variant of Problem 1: Given a pair $(\mathbf{C}, \mathbf{A})$, a budget $b \in \mathbb{R}_{\geq 0}$, a cost function $c : \mathbb{R}^{p \times p} \rightarrow \mathbb{R}_{\geq 0}$, and a threshold $\alpha \in [0, 1]$, determine whether there exists a protection matrix $\mathbf{P} \in \{0, 1\}^{p \times p}$ such that $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P}) \geq \alpha$ and $c(\mathbf{P}) \leq b$.*

This problem is coNP-hard. We can reduce from the safety factor decision problem, which is coNP-complete by Remark 3. Given an instance $(\mathbf{C}, \mathbf{A}, \mathbf{P}_0, \alpha)$ of the safety factor decision problem, construct an instance of the optimal sensor protection decision problem by setting:

$$c(\mathbf{P}) = \begin{cases} 0 & \text{if } \mathbf{P} = \mathbf{P}_0, \\ 1 & \text{otherwise,} \end{cases} \quad b := 0.$$

Because the budget is 0, $\mathbf{P} = \mathbf{P}_0$ is the only feasible choice, so the new instance is “yes” if, and only if, the original instance is “yes.” This mapping is computable in polynomial time, hence the optimal sensor protection decision problem is coNP-hard. Determining whether it is coNP-complete is left open.

◆

A naive approach to solving Problem 1 is demonstrated in Example 5, where we enumerate all possible protection sets and find the one that maximizes the safety factor while satisfying the budget constraint. The number of possible protection sets is 2^p , where p is the number of sensors. Thus, the naive approach requires evaluating the safety factor function $O(2^p)$ times. This exponential growth is consistent with the theoretical hardness of the decision version of the problem, which is coNP-hard (Remark 4). For large problems, such brute-force methods become infeasible. This motivates us to examine properties of the safety factor function and explore ways to efficiently compute the optimal sensor protection set.

3.5 Properties of the Safety Factor Function

In this section, we explore whether the safety factor function $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P})$ is submodular or supermodular. If it is, we can leverage extensive research on submodular/supermodular optimization (for example, [33]) to efficiently solve or approximate Problem 1.

3.5.1 Monotonicity

The problem of maximizing monotone submodular functions can be approximated efficiently using a greedy algorithm [46]. We first establish that the safety factor function $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P})$ is monotone with respect to the protected set P .

Proposition 1. *Consider the pair $(\mathbf{C}, \mathbf{A})$ where $\mathbf{C} \in \mathbb{R}^{p \times n}$ and $\mathbf{A} \in \mathbb{R}^{n \times n}$. Then $\forall P_1, P_2 \subseteq \{1, 2, \dots, p\}$, if $P_1 \subseteq P_2$, then $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P}_1) \leq \gamma(\mathbf{C}, \mathbf{A}, \mathbf{P}_2)$.*

Proof. It suffices to show that $\forall P \subseteq \{1, 2, \dots, p\}$ and $x \in \{1, 2, \dots, p\} \setminus P$, $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P}) \leq \gamma(\mathbf{C}, \mathbf{A}, \mathbf{P} \cup \{x\})$.

We first reduce this expression to a simpler form:

$$\begin{aligned}
\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P}) &\leq \gamma(\mathbf{C}, \mathbf{A}, \mathbf{P} \cup \{x\}) \\
&\iff \frac{\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P})/2 + |P|}{p} \leq \frac{\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P} \cup \{x\})/2 + |P| + 1}{p} \\
&\iff \sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P})/2 \leq \sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P} \cup \{x\})/2 + 1 \\
&\iff \sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P} \cup \{x\}) - \sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P}) \geq -2.
\end{aligned} \tag{3.5}$$

In other words, we want to show that adding a sensor to the protected set P will not decrease the sparse observability index by more than 2.

Let $\mathcal{M}(P, s) = \{M \subseteq \{1, 2, \dots, p\} \setminus P \mid |M| \leq s\}$ be the collection of all possible sets of missing sensors with size at most s for the pair $(\mathbf{C}, \mathbf{A})$ with sensor protection on P . Then $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P}) = \max\{s \mid \forall M \in \mathcal{M}(P, s), (\mathbf{C}_{\{1, 2, \dots, p\} \setminus M}, \mathbf{A}) \text{ is observable}\}$ and $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P} \cup \{x\}) = \max\{s' \mid \forall M' \in \mathcal{M}(P \cup \{x\}, s'), (\mathbf{C}_{\{1, 2, \dots, p\} \setminus M'}, \mathbf{A}) \text{ is observable}\}$. Let $s^* = \sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P})$, by (3.5), it suffices to show that there exists an $s' \leq \sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P} \cup \{x\})$ such that $s' - s^* \geq -2$. In fact, we will show a stronger result: $s' - s^* \geq -1$.

Let $s' = s^* - 1$ and $M' \in \mathcal{M}(P \cup \{x\}, s')$. Notice that $M' \in \mathcal{M}(P, s^*)$ as well. Thus, $\mathcal{M}(P \cup \{x\}, s') \subseteq \mathcal{M}(P, s^*)$. Since $(\mathbf{C}_{\{1, 2, \dots, p\} \setminus M}, \mathbf{A})$ is observable for all $M \in \mathcal{M}(P, s^*)$, $(\mathbf{C}_{\{1, 2, \dots, p\} \setminus M'}, \mathbf{A})$ is observable as well. This implies that $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P} \cup \{x\}) \geq s'$. Since $s' = s^* - 1$, we have $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P} \cup \{x\}) - \sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P}) \geq -1$, which satisfies (3.5) and completes the proof. ■

3.5.2 Non-submodularity and non-supermodularity

For some choices of $\mathbf{C}$ and $\mathbf{A}$, the safety factor function $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P})$ is submodular with respect to P . In other scenarios, the function exhibits a supermodular behavior. However, in general, the safety factor function is neither submodular nor supermodular. We demonstrate this with a few examples.

First, recall the definitions of submodularity and supermodularity for a set function $f : 2^V \rightarrow \mathbb{R}$.

Definition 4. A set function $f : 2^V \rightarrow \mathbb{R}$ is **submodular** if for all $A, B \subseteq V$ such that $A \subseteq B$ and $x \in V \setminus B$:

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B).$$

Oftentimes, submodularity is interpreted as a property of diminishing returns: adding an element to a set has at least as large an impact as adding it to a superset of that set.

Definition 5. A set function $f : 2^V \rightarrow \mathbb{R}$ is **supermodular** if for all $A, B \subseteq V$ such that $A \subseteq B$ and $x \in V \setminus B$:

$$f(A \cup \{x\}) - f(A) \leq f(B \cup \{x\}) - f(B).$$

Supermodularity is the opposite of submodularity: adding an element to a set has at most as large an impact as adding it to a superset of that set. It is often used to model complementarity: the value of adding an element to a set increases when the set already contains another specific element.

We will now show that for some choices of $\mathbf{C}$ and $\mathbf{A}$, the safety factor function $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P})$ is submodular with respect to P .

Example 6. Consider the pair $(\mathbf{C}, \mathbf{A})$ from Example 1:

$$\mathbf{A} = \begin{bmatrix} 1 & 0.1 & 0 & 0 \\ 0 & 1 & 0.1 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The safety factor function $\gamma(\mathbf{C}, \mathbf{A}, \mathbf{P})$ is submodular with respect to P . We can see this through exhaustive testing. Let $V = \{1, 2, 3, 4\}$ and for all $P_B \subseteq V, P_A \subseteq P_B$, and $x \in V \setminus P_B$, we can verify that the inequality $\gamma(\mathbf{C}, \mathbf{A}, P_A \cup \{x\}) - \gamma(\mathbf{C}, \mathbf{A}, P_A) \geq \gamma(\mathbf{C}, \mathbf{A}, P_B \cup \{x\}) - \gamma(\mathbf{C}, \mathbf{A}, P_B)$ holds. Thus, the safety factor function is submodular with respect to P with this choice of $\mathbf{C}$ and $\mathbf{A}$. ▲

However, the safety factor function is not submodular in general. We demonstrate this with the following example.

Example 7. Consider the pair $(\mathbf{C}, \mathbf{A})$ where

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0.1 \\ 0 & 1 & 0.1 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Let $P_1 = \emptyset$ and $P_2 = \{1\}$. Let $x = 2$. Then:

$$\begin{aligned} \gamma(\mathbf{C}, \mathbf{A}, P_1 \cup \{x\}) - \gamma(\mathbf{C}, \mathbf{A}, P_1) &= \frac{1}{3} - 0 = \frac{1}{3}, \\ \gamma(\mathbf{C}, \mathbf{A}, P_2 \cup \{x\}) - \gamma(\mathbf{C}, \mathbf{A}, P_2) &= \frac{2.5}{3} - \frac{1}{3} = 0.5. \end{aligned}$$

The marginal gain of adding sensor 2 to the larger set P_2 is greater than the marginal gain of adding it to the smaller set P_1 . Thus, the safety factor function is not submodular in this case. ▲

One reason Example 7 is not submodular is the presence of complementary sensors. Sensors 1 and 2 are complementary in the sense that protecting both of them is more beneficial than protecting either one individually. This may suggest that the safety factor function is supermodular in this case. However, it can be shown that the safety factor function is not supermodular either. We demonstrate this with the following example.

Example 8. Consider the pair $(\mathbf{C}, \mathbf{A})$ from Example 7. Let $P_1 = \emptyset$ and $P_2 = \{1, 2\}$. Let $x = 3$. Then:

$$\begin{aligned}\gamma(\mathbf{C}, \mathbf{A}, P_1 \cup \{x\}) - \gamma(\mathbf{C}, \mathbf{A}, P_1) &= \frac{1}{3} - 0 = \frac{1}{3}, \\ \gamma(\mathbf{C}, \mathbf{A}, P_2 \cup \{x\}) - \gamma(\mathbf{C}, \mathbf{A}, P_2) &= 1 - \frac{2.5}{3} = \frac{1}{6}.\end{aligned}$$

The marginal gain of adding sensor 3 to the larger set P_2 is less than the marginal gain of adding it to the smaller set P_1 . Thus, by Definition 5, the safety factor function is not supermodular in this case.

▲

In general, the safety factor function exhibits characteristics of both submodularity and supermodularity: adding complementary sensors to the protected set can lead to a notable increase in value, while at other times, the function shows diminishing returns. However, the safety factor function does not satisfy the criteria for submodularity or supermodularity in a general sense. This negative result suggests that we cannot directly apply submodular or supermodular optimization techniques to solve Problem 1, and we must explore other methods to efficiently solve the problem or resort to the naive approach as shown in Example 5.

3.6 The Measurement Corruption Detection Problem

In Section 3.4, we introduced the optimal sensor protection problem, which seeks to find the optimal set of sensors to protect to maximize a system's ability to handle sensor faults. In this section, we consider the problem of detecting sensor faults in a system, which is a complementary problem to the optimal sensor protection problem.

Problem 2. *Given system (3.2) with protection on P and measurement vectors $\{\mathbf{y}[k], k \in [0, n-1]\}$, find all sensors $i \in \{1, 2, \dots, p\} \setminus P$ such that:*

$$\exists k \in [0, n-1] \text{ where } y_{\{i\}}[k] \neq \mathbf{C}_{\{i\}}\mathbf{x}[k].$$

In other words, we seek to find all sensors that have been corrupted at some point in the measurement window ($k \in [0, n-1]$). Once this problem is solved, we can use the resulting set of corrupted sensors to inform existing state estimation or fault tolerance algorithms to take corrective actions. We will discuss this further in Section 3.6.3 and demonstrate it in Chapter 5.

If the pair $(\mathbf{C}, \mathbf{A})$ is $2s$ -sparse observable with protection on $\mathbf{P}$ and $|K_{[0,n-1]}(\mathbf{P}\mathbf{e})| \leq s$, then by Theorem 2, $\{\mathbf{y}[k], k \in [0, n-1]\}$ uniquely determines the initial state $\mathbf{x}[0]$. From $\mathbf{x}[0]$, we can recover $\mathbf{x}[k]$ for all $k \in [0, n-1]$ using $\mathbf{A}$ and subsequently obtain the effects of sensor corruption $\{\mathbf{P}\mathbf{e}[k], k \in [0, n-1]\}$ using $\mathbf{y}[k] = \mathbf{C}\mathbf{x}[k] + \mathbf{P}\mathbf{e}[k]$. With this information, we can solve Problem 2 by finding all sensors $i \in \{1, 2, \dots, p\} \setminus P$ such that $\exists k \in [0, n-1]$ where $\mathbf{P}\mathbf{e}[k]_{\{i\}} \neq 0$.

Because of this, we are interested in solving Problem 2 under the assumption that the pair $(\mathbf{C}, \mathbf{A})$ is $2s$ -sparse observable with protection on $\mathbf{P}$ and $|K_{[0,n-1]}(\mathbf{P}\mathbf{e})| \leq s$. We want to find an algorithm that realizes the guarantees provided by Theorem 2.

The secure estimation literature studies precisely the problem of reconstructing the initial state $\mathbf{x}[0]$ in the presence of sensor faults. We will leverage a result from [21] to solve Problem 2.

3.6.1 Secure State-Reconstruction

The problem of recovering the initial state of a system in the presence of sensor faults is known as secure state-reconstruction. It is defined as follows:

Problem 3 ([21]). *Given a pair $(\mathbf{C}, \mathbf{A})$ and measurement vectors $\mathbf{y}[k] = \mathbf{C}\mathbf{A}^k\mathbf{x}[0] + \mathbf{e}[k], k \in [0, n-1]$, find the initial state $\mathbf{x}[0]$.*

In [21], the authors presented a solution to this problem as the solution to the following optimization problem:

$$\begin{aligned} & \arg \min && |M| \\ M \subseteq \{1, 2, \dots, p\}, \hat{\mathbf{x}}_0 \in \mathbb{R}^n &&& \\ & \text{subject to} && \text{supp}(\mathbf{y}[k] - \mathbf{C}\mathbf{A}^k\hat{\mathbf{x}}_0) \subseteq M, \quad \forall k \in [0, n-1] \end{aligned} \tag{3.6}$$

where $\text{supp}(\mathbf{x})$ denotes the support (i.e., the set of non-zero indices) of a vector $\mathbf{x}$.

Let $(M^*, \hat{\mathbf{x}}_0^*)$ be the solution to (3.6) for a pair $(\mathbf{C}, \mathbf{A})$ and measurements $\mathbf{y}[k] = \mathbf{C}\mathbf{A}^k\mathbf{x}[0] + \mathbf{e}[k], k \in [0, n-1]$. The authors presented an equivalent theorem to Theorem 1, and showed that in the presence of s sensor faults, if the initial state $\mathbf{x}[0]$ can be uniquely reconstructed, then $\hat{\mathbf{x}}_0^* = \mathbf{x}[0]$.

Incorporating sensor protection into Problem 3 is a straightforward replacement of $\mathbf{e}[k]$ with $\mathbf{P}\mathbf{e}[k]$ in the measurement vectors $\mathbf{y}[k]$.

Problem 4. *Given a pair $(\mathbf{C}, \mathbf{A})$, a protection matrix $\mathbf{P}$, and measurement vectors $\mathbf{y}[k] = \mathbf{C}\mathbf{A}^k\mathbf{x}[0] + \mathbf{P}\mathbf{e}[k], k \in [0, n-1]$, find the initial state $\mathbf{x}[0]$.*

With a slight modification to (3.6), by restricting the set of sensors that can be corrupted M to not contain protected sensors P , we can solve Problem 4 as well. Here's the updated optimization problem:

$$\begin{aligned} & \arg \min && |M| \\ & M \subseteq \{1, 2, \dots, p\} \setminus P, \hat{\mathbf{x}}_0 \in \mathbb{R}^n && \\ & \text{subject to} && \text{supp}(\mathbf{y}[k] - \mathbf{C}\mathbf{A}^k\hat{\mathbf{x}}_0) \subseteq M, \quad \forall k \in [0, n-1] \end{aligned} \quad (3.7)$$

Note that the only difference between (3.6) and (3.7) is the restriction on the set of corrupted sensors M .

Proposition 2. *Let $(M^*, \hat{\mathbf{x}}_0^*)$ be the solution to (3.7) for a pair $(\mathbf{C}, \mathbf{A})$, protection matrix $\mathbf{P}$, and measurements $\mathbf{y}[k] = \mathbf{C}\mathbf{A}^k\mathbf{x}[0] + \mathbf{P}\mathbf{e}[k], k \in [0, n-1]$. If the pair $(\mathbf{C}, \mathbf{A})$ is $2s$ -sparse observable with sensor protection on P and $K_{[0, n-1]}(\mathbf{P}\mathbf{e}) \leq s$, then $\hat{\mathbf{x}}_0^*$ is the unique solution to Problem 4.*

Proof. By the construction of (3.7), $\text{supp}(\mathbf{y}[k] - \mathbf{C}\mathbf{A}^k\hat{\mathbf{x}}_0^*) \subseteq M$ for all $k \in [0, n-1]$ and $M \cap P = \emptyset$. Thus $\exists \mathbf{e}$ such that $\mathbf{P}\mathbf{e}[k] = \mathbf{y}[k] - \mathbf{C}\mathbf{A}^k\hat{\mathbf{x}}_0^*$ for all $k \in [0, n-1]$. So $\hat{\mathbf{x}}_0^*$ is a solution to Problem 4. And since $K_{[0, n-1]}(\mathbf{P}\mathbf{e}) \leq s$, by Theorem 2, there cannot be an $\hat{\mathbf{x}}_0' \neq \hat{\mathbf{x}}_0^*$ such that $\exists \mathbf{P}\mathbf{e}'[k] = \mathbf{y}[k] - \mathbf{C}\mathbf{A}^k\hat{\mathbf{x}}_0'$ for all $k \in [0, n-1]$. Therefore, $\hat{\mathbf{x}}_0^*$ is the unique solution to Problem 4. $\blacksquare$

Note that the solution to (3.7) not only solves Problem 4, but also directly solves Problem 2. We can see this in two ways:

1. The M obtained from the solution to (3.7) represents the sensors that are corrupted in the measurement window, which is a solution to Problem 2.

2. Solving Problem 4 gives us the initial state $\mathbf{x}[0]$, which we can use to recover $\mathbf{P}\mathbf{e}[k]$ for all $k \in [0, n-1]$ using $\mathbf{y}[k] = \mathbf{C}\mathbf{A}^k\mathbf{x}[0] + \mathbf{P}\mathbf{e}[k]$. From this, the set of corrupted sensors can be obtained by finding all sensors $i \in \{1, 2, \dots, p\} \setminus P$ such that $\exists k \in [0, n-1]$ where $(\mathbf{P}\mathbf{e}[k])_{\{i\}} \neq 0$.

In summary, if the pair $(\mathbf{C}, \mathbf{A})$ is $2s$ -sparse observable with protection on $\mathbf{P}$ and the number of corrupted sensors in the measurement window is at most s , then Problem 2 can be solved by finding the solution to (3.7).

3.6.2 Handling Uncertainty

In practice, it is almost never possible to know the exact system matrices $\mathbf{A}$ and $\mathbf{C}$ due to modelling uncertainty. This uncertainty is particularly significant because the majority of real-world systems are inherently nonlinear, and their linearized representations depend on the chosen operating point, leading to approximation errors. Additionally, process and measurement noise may make it impossible to perfectly match the measurements to the system dynamics. To address these issues, we introduce a slack variable $\epsilon \in \mathbb{R}^p$ into the optimization problem (3.7) to account for modelling uncertainty and noise.

$$\begin{aligned} & \arg \min && |M| \\ & M \subseteq \{1, 2, \dots, p\} \setminus P, \hat{\mathbf{x}}_0 \in \mathbb{R}^n && \\ & \text{subject to} && \text{supp}_\epsilon(\mathbf{y}[k] - \mathbf{C}\mathbf{A}^k\hat{\mathbf{x}}_0) \subseteq M, \quad \forall k \in [0, n-1] \end{aligned} \quad (3.8)$$

where $\text{supp}_\epsilon(\mathbf{z})$ denotes the set of indices i where $\mathbf{z}_i > \epsilon_i$. Setting ϵ to a small value allows the optimization problem to be solved even when there is uncertainty in the system matrices, and setting it to zero recovers the original optimization problem (3.7).

3.6.3 Implementation

We can leverage (3.8) to facilitate real-time monitoring of sensor failures. Consider an autonomous system with a state estimator, as shown in Figure 3.1. The state estimator processes sensor measurements $\mathbf{y}[k]$ to estimate the state $\hat{\mathbf{x}}[k]$ at each time step k .

To detect sensor failures, we introduce a new module, which we will call the *detector*, to solve (3.8) to find the set of corrupted sensors $M[k]$ (see Algorithm 1).

Note that the detector is only useful when downstream modules can make use of the output $M[k]$. In the context of state estimation (as shown in Figure 3.1), this can often

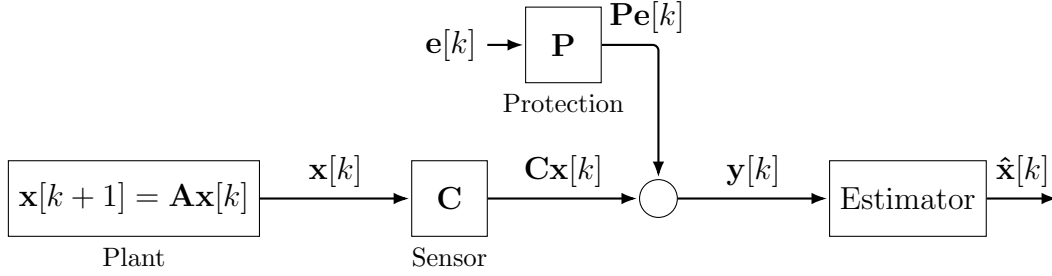


Figure 3.1: Example of an autonomous system with state estimation.

Algorithm 1 Detector Module

```

1: Parameters:  $\mathbf{C} \in \mathbb{R}^{p \times n}$ ,  $\mathbf{A} \in \mathbb{R}^{n \times n}$ ,  $\boldsymbol{\epsilon} \in \mathbb{R}^p$ 
2: Initialize:  $Y \leftarrow []$  ▷ An empty list of measurements
3: loop ▷ Triggered by clock at each time step  $k \geq 0$ 
4:   Receive measurement  $\mathbf{y}[k]$ 
5:    $Y.append(\mathbf{y}[k])$  ▷ Collect measurement for use later
6:   if  $|Y| = n$  then ▷  $Y = [\mathbf{y}[k - n + 1], \dots, \mathbf{y}[k]]$ 
7:     Note: List  $Y$  is used as  $\mathbf{y}$  in (3.8) in the following step.
8:     Solve (3.8) using  $\mathbf{C}$ ,  $\mathbf{A}$ ,  $\boldsymbol{\epsilon}$  and  $Y$  to obtain the set of corrupted sensors  $M[k]$ 
9:     Output:  $M[k]$ 
10:     $Y.popleft()$  ▷ Remove the oldest measurement
11:   end if
12: end loop

```

be achieved by applying minor tweaks to the estimator. We will show an example of this in Chapter 5.

With proper modifications, the detector can be integrated as shown in Figure 3.2.

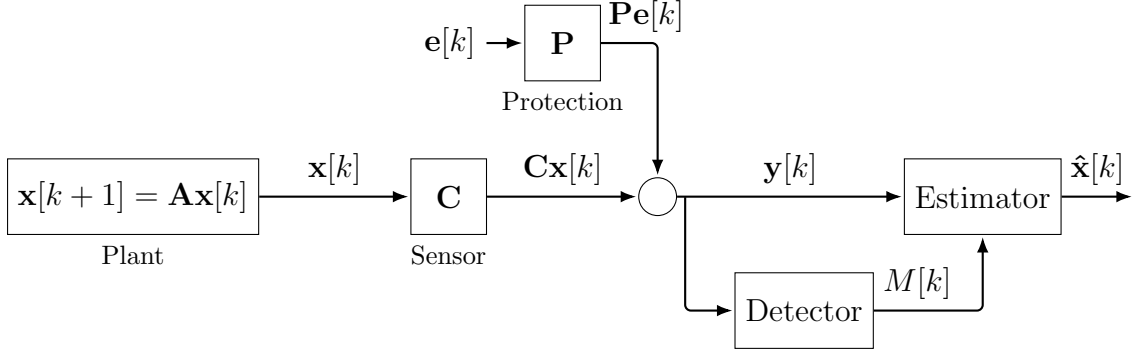


Figure 3.2: Example of an autonomous system with state estimation and sensor fault detection.

In Figure 3.2, the detector collects (potentially corrupted) measurements $\mathbf{y}[k]$, and outputs the set of corrupted sensors $M[k]$ based on the solution to (3.8). The estimator uses $M[k]$ in addition to the measurements to produce the estimated state $\hat{\mathbf{x}}[k]$. Compared to secure state reconstruction algorithms presented in previous works, the detector is modular and does not require replacing existing state estimation algorithms. Oftentimes, only minor tweaks are needed to take advantage of the detector's output in the estimator. This allows for easier integration into existing systems.

For non-autonomous systems, the detector can be integrated in a similar way (see Figure 3.3). To handle the input signal, we can make use of the method described in [49]. Consider the following system:

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{A}\mathbf{x}[k] + \mathbf{B}\mathbf{u}[k], & k \in \mathbb{N}_0, & \quad \mathbf{x}[0] = \mathbf{x}_0, \\ \mathbf{y}[k] &= \mathbf{C}\mathbf{x}[k] + \mathbf{P}\mathbf{e}[k], \end{aligned} \quad (3.9)$$

where $\mathbf{B}$ is the input matrix and $\mathbf{u}[k]$ is the input signal. The output signal $\mathbf{y}[k]$ expands to

$$\mathbf{y}[k] = \mathbf{C}\mathbf{x}[k] + \mathbf{P}\mathbf{e}[k] = \mathbf{C}\mathbf{A}^k \mathbf{x}[0] + \mathbf{C} \sum_{i=0}^{k-1} \mathbf{A}^i \mathbf{B} \mathbf{u}[k-i-1] + \mathbf{P}\mathbf{e}[k].$$

Note that the term $\mathbf{C} \sum_{i=0}^{k-1} \mathbf{A}^i \mathbf{B} \mathbf{u}[k-i-1]$ only depends on the input signal $\mathbf{u}[k]$ and the system matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. If we have access to $u[k]$, we can turn this non-autonomous system into an autonomous system by redefining the output signal as $\mathbf{y}'[k] =$

$\mathbf{y}[k] - \mathbf{C} \sum_{i=0}^{k-1} \mathbf{A}^i \mathbf{B} \mathbf{u}[k-i-1]$. Then the system becomes

$$\begin{aligned} \mathbf{x}'[k+1] &= \mathbf{A} \mathbf{x}'[k], \quad k \in \mathbb{N}_0, \quad \mathbf{x}'[0] = \mathbf{x}_0, \\ \mathbf{y}'[k] &= \mathbf{C} \mathbf{x}'[k] + \mathbf{P} \mathbf{e}[k]. \end{aligned} \quad (3.10)$$

Solving (3.8) with $\mathbf{C}$, $\mathbf{A}$ and $\mathbf{y}'[k]$, $k \in [0, n-1]$ gives $\mathbf{x}_0$ and $\mathbf{P} \mathbf{e}[k]$ for all $k \in [0, n-1]$, which is the same for (3.9) and (3.10). We can modify Algorithm 1 to handle the input signal $\mathbf{u}[k]$ and use the transformed output signal $\mathbf{y}'[k]$ to solve Problem 2 for non-autonomous systems (see Algorithm 2).

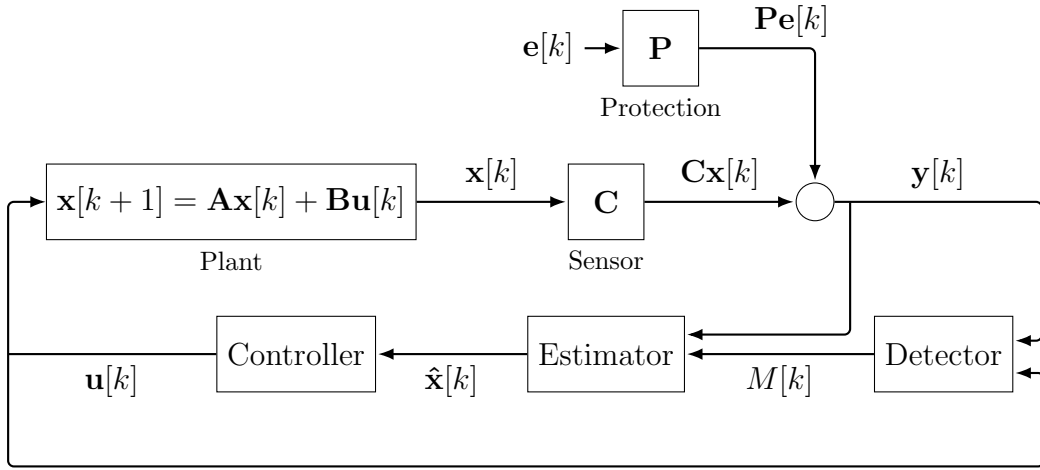


Figure 3.3: Example of a control loop with sensor fault detection.

Since $\mathbf{A}$, $\mathbf{C}$, and $\mathbf{P}$ remain the same during the transformation from (3.9) to (3.10), it follows that Theorem 2 holds for both autonomous systems (3.2) and non-autonomous systems (3.9). This will become useful when we perform sparse observability analysis on non-autonomous systems in Chapters 4 and 5.

3.6.4 Computational Complexity

Solving (3.8) is computationally difficult because it is a combinatorial optimization problem. The number of possible subsets of sensors is 2^p , which makes the problem intractable for large p . In fact, as shown in [21], the special case where $\epsilon = \mathbf{0}$, $\mathbf{A} \in \mathbb{R}^{1 \times 1}$ and $\mathbf{C} \in \mathbb{R}^{p \times 1}$ can be written as:

$$\arg \min_{\mathbf{x}[0] \in \mathbb{R}} \|\mathbf{y}[0] - \mathbf{C} \mathbf{x}[0]\|_{l_0}, \quad (3.11)$$

Algorithm 2 Detector Module for Non-Autonomous Systems

```

1: Parameters:  $\mathbf{C} \in \mathbb{R}^{p \times n}$ ,  $\mathbf{A} \in \mathbb{R}^{n \times n}$ ,  $\mathbf{B} \in \mathbb{R}^{n \times m}$ ,  $\boldsymbol{\epsilon} \in \mathbb{R}^p$ 
2: if  $n = 1$  then
3:   Note:  $n = 1$  implies an window size of 1. Inputs are not needed for this case.
4:   Run Algorithm 1 with  $\mathbf{C}, \mathbf{A}$ 
5: end if
6: Initialize:  $Y \leftarrow []$  ▷ An empty list of measurements
7: Initialize:  $U \leftarrow []$  ▷ An empty list of input signals
8: loop ▷ Triggered by clock at each time step  $k \geq 0$ 
9:   Receive measurement  $\mathbf{y}[k]$ 
10:   $Y.append(\mathbf{y}[k])$  ▷ Collect measurement for use later
11:  if  $k \geq 1$  then
12:    Receive input signal  $\mathbf{u}[k - 1]$  ▷ The latest input is from the previous time step
13:     $U.append(\mathbf{u}[k - 1])$  ▷ Collect input for use later
14:  end if
15:  if  $|Y| = n$  then
16:    Note:  $Y = [\mathbf{y}[k - n + 1], \dots, \mathbf{y}[k]]$ ,  $U = [\mathbf{u}[k - n + 1], \dots, \mathbf{u}[k - 1]]$ 
17:    Initialize:  $Y' \leftarrow []$ 
18:    for  $i = 1$  to  $n$  do
19:       $Y'.append(Y[i] - \mathbf{C} \sum_{j=0}^{i-1} \mathbf{A}^j \mathbf{B} U[i - j - 1])$ 
20:    end for
21:    Note: List  $Y'$  is used as  $\mathbf{y}$  in (3.8) in the following step.
22:    Solve (3.8) using  $\mathbf{C}, \mathbf{A}, \boldsymbol{\epsilon}$  and  $Y'$  to obtain the set of corrupted sensors  $M[k]$ 
23:    Output:  $M[k]$ 
24:     $Y.popleft()$  ▷ Remove the oldest measurement
25:     $U.popleft()$  ▷ Remove the oldest input signal
26:  end if
27: end loop

```

where $\|v\|_{l_0} = |\text{supp}(v)|$ is the l_0 norm (the number of non-zero entries) of a vector v . This problem is known to be NP-hard [23].

Authors of [21] proposed a relaxation of the l_0 -norm where the mixed l_1/l_r -norm is used instead. This relaxation makes the optimization problem convex and can be solved efficiently. However, the relaxation has fewer guarantees than the original problem. For example, in the presence of s sensor faults, solving (3.8) recovers the initial state $\mathbf{x}[0]$ if the pair $(\mathbf{C}, \mathbf{A})$ is $2s$ -sparse observable (by Theorem 1). The relaxation does not have this guarantee. Instead, it requires a stronger condition characterized by a more complex expression. On the other hand, if the stronger condition is satisfied and the relaxation is used to solve the problem, the solution is guaranteed to be the optimal solution to the original problem (3.8).

In practice, if sufficient computational resources are available or if the system is sufficiently small, the original problem (3.8) can be solved directly. In Chapter 4, we will show that the detector is real-time capable on a single core of a modern CPU for a small car-like system. On less powerful computers or for larger systems, it may also be acceptable to run the detector at set intervals instead of every time step. This delays the detection of sensor faults, but can be a reasonable trade-off. We demonstrate this in Chapter 5.

3.7 Time-Varying Systems

So far, we have focused on linear time-invariant systems. However, many real-world systems are nonlinear. In practice, a common way to deal with nonlinear systems is to linearize the model and then use the linear model as the basis for design. When this process is done at multiple operating points, the resulting approximation can be viewed as a time-varying linear system, where the system matrices $\mathbf{A}$ and $\mathbf{C}$ change with time. In this section, we extend our methodology to handle time-varying systems.

A key difference between time-invariant and time-varying systems is that the observability matrix is now parameterized by the time interval $[k_0, k_f] \subset \mathbb{N}_0$. In the time-invariant case, the Cayley-Hamilton theorem allows us to fix $k_f - k_0 = n$, where n is the dimension of the state space. However, for time-varying systems, the observability properties may change over time, which affects the guaranteed performance of the fault detection algorithm.

To address this, we first extend the concept of sparse observability (Definition 2) to time-varying systems:

Definition 6. Let $P \subseteq \{1, 2, \dots, p\}$ and let $\mathbf{A} : \mathbb{N}_0 \rightarrow \mathbb{R}^{n \times n}$, $\mathbf{C} : \mathbb{N}_0 \rightarrow \mathbb{R}^{p \times n}$. The **sparse observability index with sensor protection on P** for the pair $(\mathbf{C}[k], \mathbf{A}[k])$ over the time interval $[k_0, k_f] \subset \mathbb{N}_0$, denoted by $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P}, [k_0, k_f])$, is the largest s where for every set $M \subseteq \{1, 2, \dots, p\} \setminus P$ with $|M| \leq s$, the following matrix has full column rank:

$$\begin{bmatrix} \mathbf{C}_{\{1,2,\dots,p\} \setminus M}[k_0] \\ \mathbf{C}_{\{1,2,\dots,p\} \setminus M}[k_0 + 1]\Phi(k_0 + 1, k_0) \\ \mathbf{C}_{\{1,2,\dots,p\} \setminus M}[k_0 + 2]\Phi(k_0 + 2, k_0) \\ \vdots \\ \mathbf{C}_{\{1,2,\dots,p\} \setminus M}[k_f]\Phi(k_f, k_0) \end{bmatrix}$$

where $\Phi(k, k_0)$ is defined recursively by:

$$\Phi(k_0, k_0) = \mathbf{I}, \quad \Phi(k + 1, k_0) = \mathbf{A}[k]\Phi(k, k_0).$$

The matrix $\Phi(k, k_0)$ is known as the state transition matrix and represents how the state evolves from time k_0 to time k . It serves as the equivalent of $\mathbf{A}^k$ in the time-invariant case.

Since the sparse observability index may vary with time, we introduce the concept of uniform sparse observability to provide a framework for analyzing time-varying systems with stable sparse observability properties.

Definition 7. A pair of matrix-valued sequences $\mathbf{A} : \mathbb{N}_0 \rightarrow \mathbb{R}^{n \times n}$, $\mathbf{C} : \mathbb{N}_0 \rightarrow \mathbb{R}^{p \times n}$ has **uniform sparse observability with protection** if there exists an $N \geq n$ such that for every set of protected sensors $\mathbf{P}$, the sparse observability index $\sigma_o(\mathbf{C}, \mathbf{A}, \mathbf{P}, [k_0, k_0 + N - 1])$ is constant and maximal (increasing N does not increase the sparse observability index), for all $k_0 \in \mathbb{N}_0$.

If a time-varying system has uniform sparse observability with protection, then from the perspective of optimal sensor protection and fault detection, the system can effectively be treated as time-invariant. The only difference is that the measurement window now has length N , which, in general, is longer than the window length of n used in the time-invariant case.

In general, it is difficult to verify if a given time-varying system satisfies the criteria in Definition 7. However, for many practical systems, extensive numerical testing can provide strong evidence that the system possesses this property. This is particularly true for systems that are periodic or have slowly varying dynamics.

The secure state reconstruction problem (3.8) can be adapted to time-varying systems by replacing the time-invariant system evolution calculations with their time-varying counterparts:

$$\begin{aligned}
& \arg \min && |M| \\
M \subseteq \{1, 2, \dots, p\} \setminus P, \hat{\mathbf{x}}_0 \in \mathbb{R}^n &&& \\
\text{subject to} &&& \text{supp}_{\epsilon}(\mathbf{y}[k] - \mathbf{C}[k]\Phi(k, k_0)\hat{\mathbf{x}}_0) \subseteq M, \quad \forall k \in [k_0, k_f],
\end{aligned} \tag{3.12}$$

where $[k_0, k_f]$ is the time interval of interest ($k_f = k_0 + N - 1$), and $\Phi(k, k_0)$ is the state transition matrix defined in Definition 6.

The measurement corruption detection algorithm (Algorithm 2) can be similarly adapted to handle time-varying systems. The main changes involve:

1. Adjusting the measurement window size to N instead of n
2. Computing the state transition matrix $\Phi(k, k_0)$ instead of using $\mathbf{A}^k$

The resulting algorithm (Algorithm 3) maintains the same structure and guarantees as the time-invariant version, provided that the system has uniform sparse observability with protection.

Algorithm 3 Detector Module for Non-Autonomous Linear Time-Varying Systems

```

1: Parameter:  $N \in \mathbb{N}, \mathbf{C} : \mathbb{N}_0 \rightarrow \mathbb{R}^{p \times n}, \Phi : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{R}^{n \times n}, \mathbf{B} : \mathbb{N}_0 \rightarrow \mathbb{R}^{n \times m}, \boldsymbol{\epsilon} \in \mathbb{R}^p$ 
2: Initialize:  $Y \leftarrow []$  ▷ An empty list of measurements
3: Initialize:  $U \leftarrow []$  ▷ An empty list of input signals
4: loop ▷ Triggered by clock at each time step  $k \geq 0$ 
5:   Receive measurement  $\mathbf{y}[k]$ 
6:    $Y.append(\mathbf{y}[k])$  ▷ Collect measurement for use later
7:   Receive state estimate  $\hat{\mathbf{x}}[k]$ 
8:   if  $k \geq 1$  then
9:     Receive input signal  $\mathbf{u}[k-1]$  ▷ The latest input is from the previous time step
10:     $U.append(\mathbf{u}[k-1])$  ▷ Collect input for use later
11:  end if
12:  if  $|Y| = N$  then
13:    Note:  $Y = [\mathbf{y}[k-N+1], \dots, \mathbf{y}[k]], U = [\mathbf{u}[k-N+1], \dots, \mathbf{u}[k-1]]$ 
14:    Initialize:  $Y' \leftarrow []$ 
15:    for  $i = 1$  to  $n$  do
16:       $Y'.append(Y[i] - \mathbf{C}[i] \sum_{j=0}^{i-1} \Phi(i, j) \mathbf{B}[j] U[i-j-1])$ 
17:    end for
18:    Note: List  $Y'$  is used as  $\mathbf{y}$  in (3.12) in the following step.
19:    Solve (3.12) using  $\mathbf{C}, \Phi, \boldsymbol{\epsilon}$  and  $Y'$  to obtain the set of corrupted sensors  $M[k]$ 
20:    Output:  $M[k]$ 
21:     $Y.popleft()$  ▷ Remove the oldest measurement
22:     $U.popleft()$  ▷ Remove the oldest input signal
23:  end if
24: end loop

```

Chapter 4

Path-Following for a Car-Like Vehicle: Simulations and Case Study

In this chapter, we solve the Optimal Sensor Protection Problem (Section 3.4) and apply our measurement corruption detection scheme (Section 3.6) on a simulated nonlinear car-like vehicle following a predefined figure eight path. We show how the LTV adaptation of these methods (Section 3.7) can be applied to the nonlinear vehicle model through linearization. We also backtest the algorithm on the data collected during the [Indy Autonomous Challenge \(IAC\)](#) crash introduced in Chapter 1 to demonstrate the effectiveness of the fault detection algorithm in a real-world scenario.

4.1 Nominal Path Following for a Car-like Vehicle

In this section, we present a simulation of a car-like vehicle following a predefined figure eight path using a standard hierarchical controller. With this setup, we establish a baseline for the vehicle’s nominal behavior. This serves as a foundation for applying the tools developed in Chapter 3 to analyze the system’s tolerance to sensor corruptions and evaluate the fault detection algorithm’s effectiveness.

4.1.1 System Model

We use a kinematic bicycle model with a rear-axle reference in a 2D plane. The system states are: x (m) and y (m), denoting the position of the vehicle’s rear axle in a fixed

inertial frame; θ (rad), the heading angle relative to the x -axis of the inertial frame; v (m/s), the longitudinal velocity of the vehicle; and δ (rad), the steering angle of the front wheels. The control inputs are a (m/s²), representing the longitudinal acceleration, and ω_δ (rad/s), representing the rate of change of the steering angle. The model is given by the following:

$$\begin{aligned}\dot{x}(t) &= v(t) \cos(\theta(t)) \\ \dot{y}(t) &= v(t) \sin(\theta(t)) \\ \dot{\theta}(t) &= \frac{v(t)}{L} \tan(\delta(t)) \\ \dot{v}(t) &= a(t) \\ \dot{\delta}(t) &= \omega_\delta(t)\end{aligned}\tag{4.1}$$

where $L \geq 0$ is the wheelbase length of the vehicle. The model is subject to the following constraints:

$$0 \leq v(t) \leq v_{\max}, \quad a_{\min} \leq a(t) \leq a_{\max}, \quad |\delta(t)| \leq \delta_{\max},\tag{4.2}$$

where $v_{\max}$, $a_{\min}$, $a_{\max}$, and $\delta_{\max}$ are the maximum velocity, minimum and maximum acceleration, and maximum steering angle, respectively. The values of these parameters are given in Table 4.1.

Parameter	Value	Description
L	4.094 m	Wheelbase length of the vehicle ^a
$v_{\max}$	107.29 m/s	Maximum velocity of the vehicle ^b
$a_{\min}$	-5.2 g	Minimum acceleration of the vehicle ^c
$a_{\max}$	3.3 g	Maximum acceleration of the vehicle ^d
$\delta_{\max}$	0.59 rad	Maximum steering angle of the vehicle ^e

^a Modelled after the AV-21 racecar, which is the racecar used in the [IAC](#).

^b Based on the 240 mph (107.29 m/s) maximum velocity of a typical IndyCar racecar [\[27\]](#).

^c Based on deceleration data from a typical motorsport race [\[42\]](#).

^d Based on the maximum acceleration of an IndyCar racecar [\[10\]](#).

^e Based on the maximum steering angle (34 degrees, or 0.59 rad) of an IndyCar racecar at the [Indianapolis Motor Speedway \(IMS\)](#) oval [\[55\]](#).

Table 4.1: Parameters of the vehicle model in (4.1) and (4.2)

We place the following sensors on the vehicle: position sensors x_m and y_m for x and y , respectively, a heading sensor θ_m for θ , two velocity sensors $v_{m,1}$ and $v_{m,2}$ for v , and a

steering angle sensors δ_m for δ . These sensors are chosen to be consistent with the sensor setup in the case study in Section 4.6 and are subject to faults, which we will inject into the system in the simulations.

4.1.2 Control Objective

The control objective is to make the (x, y) position of the car follow a figure eight path with a pre-defined velocity profile. The figure eight path is defined in the inertial frame by the following parametric equations:

$$x_d(\lambda) = \frac{l}{2} \sin(\lambda), \quad y_d(\lambda) = \frac{w}{2} \sin(2\lambda), \quad \lambda \in [0, 2\pi), \quad (4.3)$$

where $l > 0$ and $w > 0$ are the extents of the path along the x and y directions of the inertial frame, respectively, and λ is the path parameter. We call $(x_d(0), y_d(0))$ the starting point of the path.

For this simulation, we set the length and width of the figure eight path such that the maximum path curvature is similar to the one found at the IMS oval (0.005 m^{-1}). The parameters used for the track are given in Table 4.2. The resulting path is shown in Figure 4.1.

Parameter	Value	Description
l	2000 m	extent of the path along the x axis
w	1000 m	extent of the path along the y axis

Table 4.2: Parameters of the figure eight path (4.3)

Closest Point on the Path

In order to describe the nominal control law (Section 4.1.4) and the fault detection algorithm (Section 4.4), we require obtaining the closest point on the path. For any given point (x, y) in the plane, we can find the closest point on the path by solving the following optimization problem:

$$\arg \min_{\lambda} \quad (x - x_d(\lambda))^2 + (y - y_d(\lambda))^2. \quad (4.4)$$

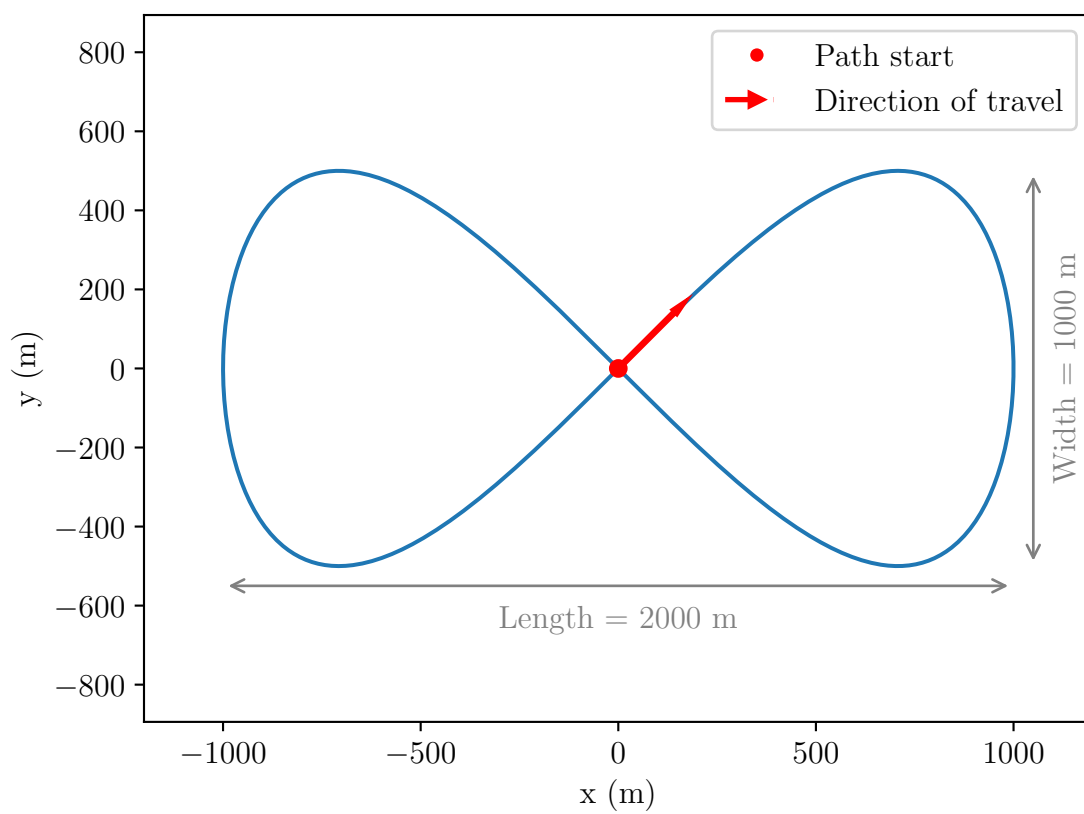


Figure 4.1: Figure eight path.

The solution λ^* to this optimization problem gives the closest point $(x_d(\lambda^*), y_d(\lambda^*))$ on the path to the given point (x, y) .

When solving (4.4) on a computer, the process is performed in discrete time steps. If there are multiple solutions for a given (x, y) , we resolve this ambiguity as follows:

1. At initialization, when no previous closest point is available, we select the smallest λ among the solutions.
2. Subsequently, once the closest point from a previous time step, λ^{*-} , is available, we select the solution that is closest to λ^{*-} . This ensures continuity and avoids abrupt changes in the computed closest point.

This heuristic is used to ensure that the vehicle follows the path in a predictable and well-defined manner.

Distance Along the Path

The distance along the path from the starting point to a given point $(x_d(\lambda), y_d(\lambda))$ is given by the arc length:

$$s(\lambda) = \int_0^\lambda \sqrt{\left(\frac{dx_d}{d\mu}\right)^2 + \left(\frac{dy_d}{d\mu}\right)^2} d\mu, \quad (4.5)$$

where μ is a dummy variable of integration. The derivatives of x_d and y_d with respect to λ are given by:

$$\frac{dx_d}{d\lambda} = \frac{l}{2} \cos(\lambda), \quad \frac{dy_d}{d\lambda} = w \cos(2\lambda). \quad (4.6)$$

Substituting these into the arc length formula gives:

$$s(\lambda) = \int_0^\lambda \sqrt{\left(\frac{l}{2} \cos(\mu)\right)^2 + (w \cos(2\mu))^2} d\mu. \quad (4.7)$$

The distance along the path with respect to the path parameter λ is shown in Figure 4.2. It is clear that there's a bijection between s and λ . Given s , we can always recover λ . We denote the inverse of the function $s(\lambda)$ as $\lambda(s)$.

In subsequent sections, we will plot the curvature and velocity profile (both of which are functions of λ) with respect to the distance along the path, $s(\lambda)$, for better interpretability.

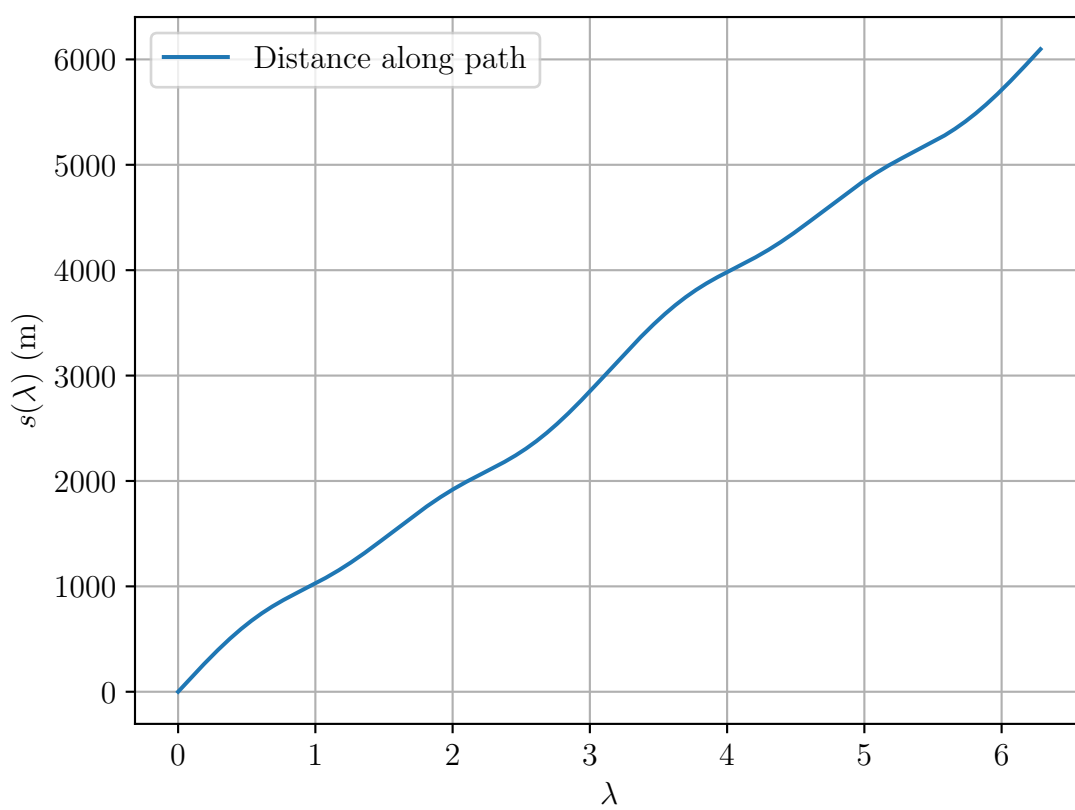


Figure 4.2: Distance along the figure eight path with respect to the path parameter λ .

Curvature Profile

The curvature at a point $(x_d(\lambda), y_d(\lambda))$ on the path is given by:

$$\kappa(\lambda) = \frac{x'_d(\lambda)y''_d(\lambda) - y'_d(\lambda)x''_d(\lambda)}{(x'_d(\lambda)^2 + y'_d(\lambda)^2)^{3/2}}, \quad (4.8)$$

where the prime denotes differentiation with respect to λ [56]. The curvature profile of the path with respect to the distance along the path is shown in Figure 4.3.

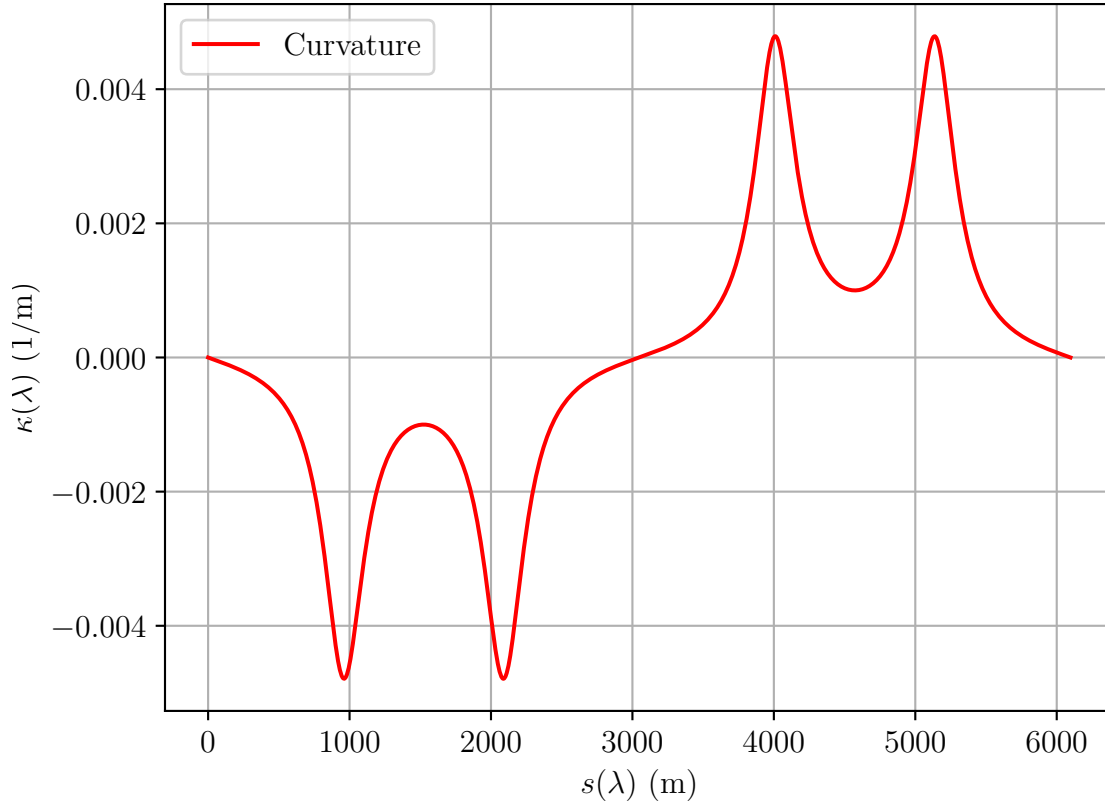


Figure 4.3: Curvature profile of the figure eight path.

Velocity Profile

In a typical IndyCar racing scenario, the maximum lateral acceleration experienced by the vehicle, denoted by $a_{\text{lat,max}}$, is around 3.3 g [43]. To ensure that the vehicle does not exceed this limit, we design a velocity profile based on the curvature of the path:

$$v(\lambda) = \min \left(v_{\text{max}}, \sqrt{\frac{a_{\text{lat,max}}}{|\kappa(\lambda)|}} \right). \quad (4.9)$$

The resulting velocity profile is shown in fig. 4.4.

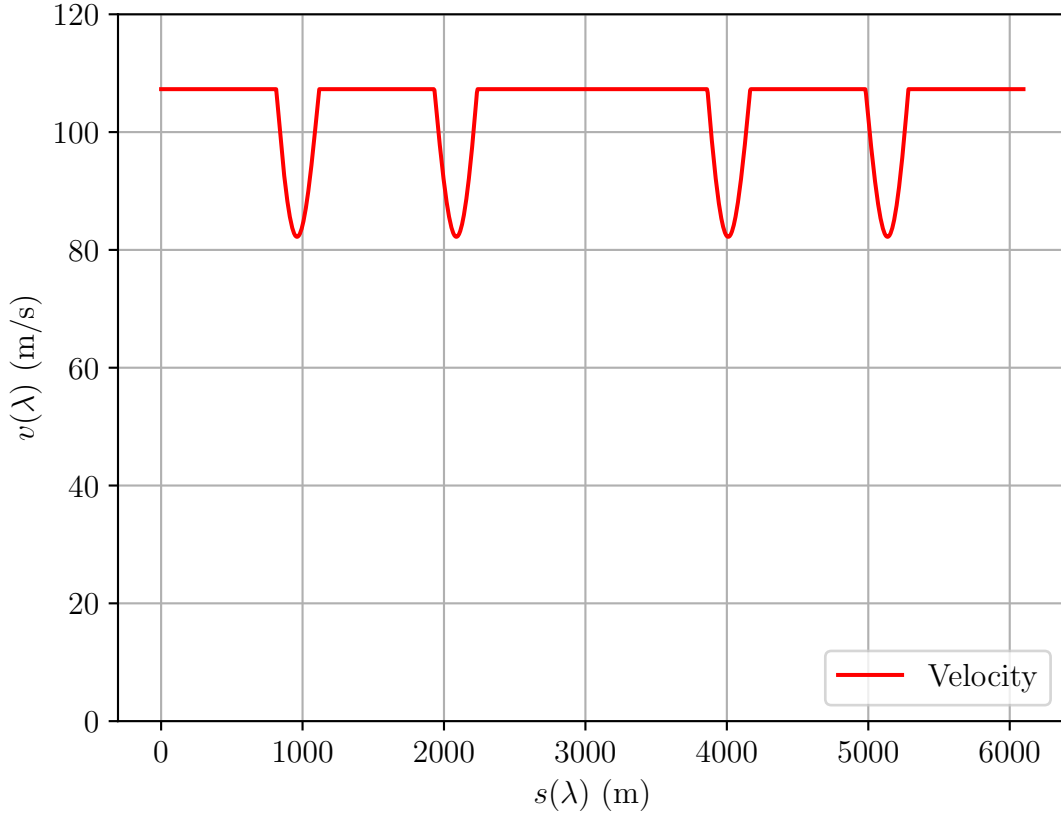


Figure 4.4: Velocity profile of the vehicle following the figure eight path.

4.1.3 State Estimation

We use an Unscented Kalman Filter (UKF) to estimate the states of the vehicle. The UKF is a nonlinear state estimator that uses the unscented transform to approximate the posterior distribution of the states through a set of sigma points [29]. These sigma points capture the mean and covariance of the state distribution, allowing the UKF to handle nonlinearities effectively. Compared to methods like the Extended Kalman Filter, the UKF is easier to implement, as it avoids the need for explicit computation of Jacobian matrices. The UKF is implemented using the `filterpy` library [34].

The UKF takes as input measurements from the sensors on the vehicle and produces estimates of the states of the vehicle, $(\hat{x}, \hat{y}, \hat{\theta}, \hat{v}, \hat{\delta})$. For the purposes of this simulation, we treat the UKF as a black box and do not delve into the details of its internals (see Figure 4.5). This UKF will play the role of the estimator as described in Chapter 3 (Figures 3.2 and 3.3).

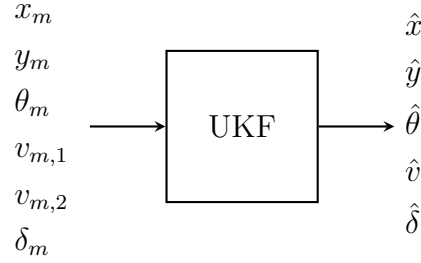


Figure 4.5: Block diagram of the Unscented Kalman Filter (UKF) denoting the inputs (sensor measurements) and outputs (state estimates).

4.1.4 Control Design

The controller takes in the desired path (4.3), velocity profile (4.9), and the state estimates from the UKF (Section 4.1.3), and generates control inputs for the vehicle. We employ a hierarchical control architecture (also known as an inner-loop/outer-loop controller) to achieve this goal. The controller consists of a high-level controller and a low-level controller.

Lookahead Point

A lookahead point is chosen on the path to guide the vehicle. The lookahead point is defined as the point on the path that is at a distance $d_{\text{lookahead}}$ ahead of the vehicle. The lookahead distance varies based on the vehicle's velocity and is tuned to improve tracking performance. In this simulation, we set $d_{\text{lookahead}}$ to $0.25 \cdot \hat{v}$, where $\hat{v}$ is the estimated velocity of the vehicle.

To find the lookahead point, we first solve (4.4) using the state estimate $(\hat{x}, \hat{y})$ to find the path parameter λ^* corresponding to the closest point on the path to the vehicle's current position estimate. Then, the lookahead point can be found by incrementing λ^* by $d_{\text{lookahead}}$:

$$\begin{aligned}\lambda_{\text{lookahead}} &= \lambda((s(\lambda^*) + d_{\text{lookahead}}) \bmod s(2\pi)), \\ x_{\text{lookahead}} &= x_d(\lambda_{\text{lookahead}}), \\ y_{\text{lookahead}} &= y_d(\lambda_{\text{lookahead}}),\end{aligned}\tag{4.10}$$

where $\lambda_{\text{lookahead}}$ is the path parameter corresponding to the lookahead point. A graphical representation of the lookahead point is shown in Figure 4.6.

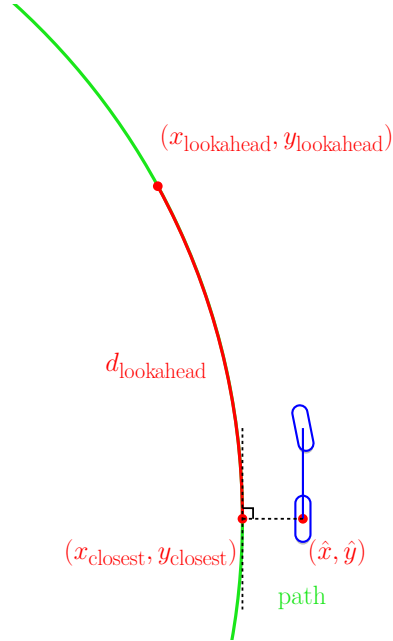


Figure 4.6: Lookahead point on the path. Derived from [22].

High-Level Controller

The high-level (outer-loop) controller generates the desired steering angle δ^* and velocity v^* based on the vehicle's current state estimate and the lookahead point on the path. The desired steering angle δ^* is calculated as:

$$\begin{aligned}\alpha &= \tan^{-1} \left(\frac{y_{\text{lookahead}} - \hat{y}}{x_{\text{lookahead}} - \hat{x}} \right), \\ \delta^* &= \tan^{-1} \left(\frac{2L \sin(\alpha)}{d_{\text{lookahead}}} \right),\end{aligned}\tag{4.11}$$

where α is the angle to the target point relative to the vehicle's heading (see Figure 4.7 for an illustration). This is a standard pure pursuit algorithm used in path-following controllers for kinematic bicycle models [22]. The desired velocity v^* is the target velocity set in the velocity profile (4.9) at the lookahead point:

$$v^* = v(\lambda_{\text{lookahead}}).\tag{4.12}$$

Low-Level Controller

The low-level (inner-loop) controller generates steering rate ω_δ^* and acceleration a^* commands that try to track the desired steering angle δ^* and velocity v^* . The control inputs are given by:

$$\omega_\delta^* = k_\delta(\delta^* - \hat{\delta}), \quad a^* = k_v(v^* - \hat{v}),\tag{4.13}$$

where k_δ and k_v are the gains of the proportional controllers. For this simulation, they are set to $k_\delta = 30$ and $k_v = 10$.

This is a simple controller commonly used in practice. The performance of the controller is adequate as long as the desired steering angle and velocity vary slowly. In this thesis, the specific choice of controller is not critical as the focus is on the fault detection algorithm. A nominal simulation of the vehicle following the path is shown in fig. 4.8.

4.2 Linearization and Discretization

In order to apply the tools developed in Chapter 3 to the vehicle model, we linearize the system around a nominal trajectory. Calculating the Jacobians of (4.1) with respect to

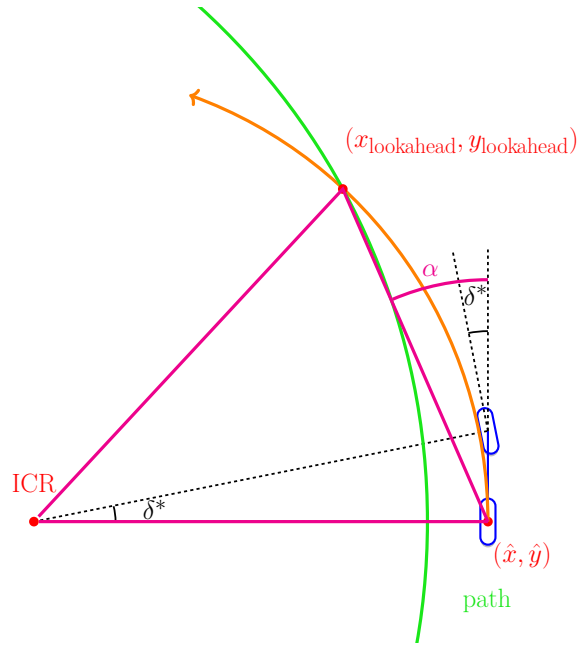


Figure 4.7: Illustration of the pure pursuit algorithm showing the lookahead point, the vehicle's position, the instantaneous center of rotation (ICR), and the desired steering angle. The orange line represents the expected trajectory of the (x, y) position of the vehicle when following the desired steering angle. Derived from [22].

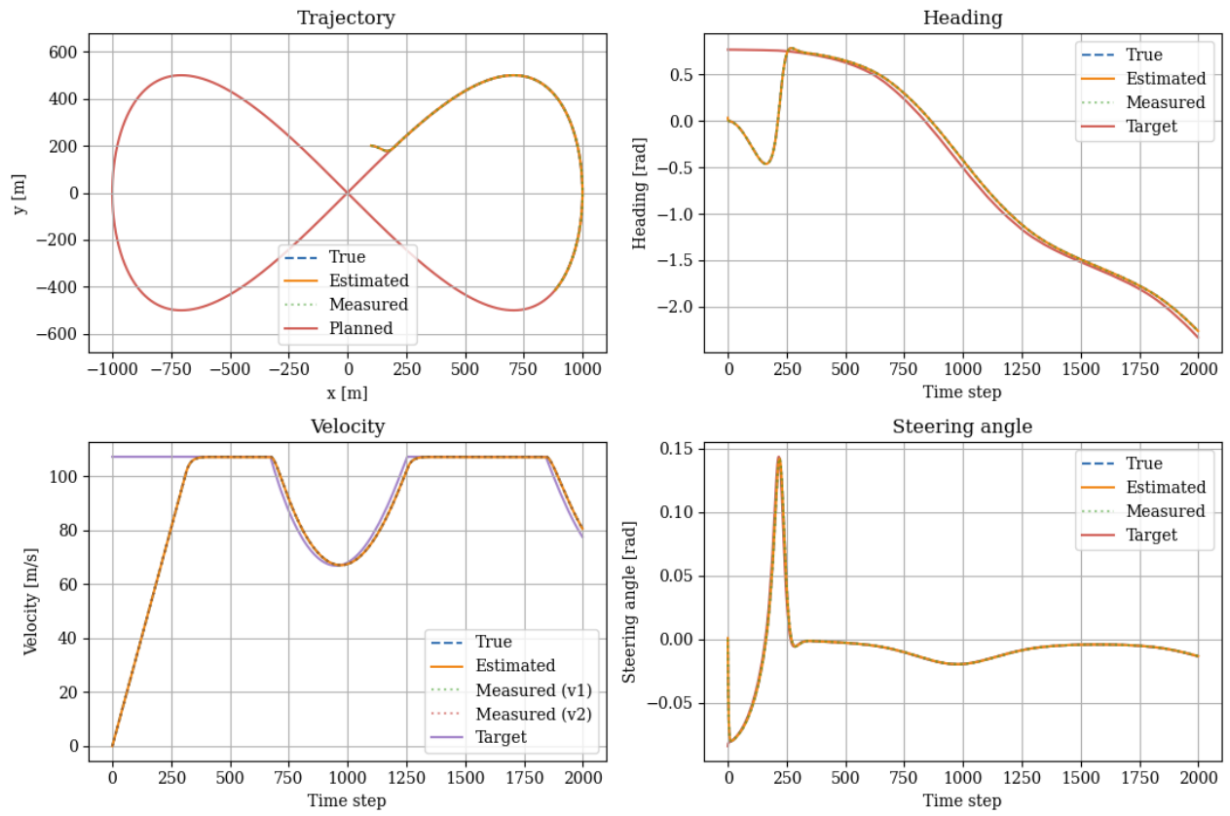


Figure 4.8: Nominal simulation of the vehicle following the figure eight path.

the states and inputs gives the linearized system:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t).\end{aligned}$$

where matrices $\mathbf{A}(t)$, $\mathbf{B}$, and $\mathbf{C}$ are as follows:

$$\mathbf{A}(t) = \begin{bmatrix} 0 & 0 & -v_0(t) \sin(\theta_0(t)) & \cos(\theta_0(t)) & 0 \\ 0 & 0 & v_0(t) \cos(\theta_0(t)) & \sin(\theta_0(t)) & 0 \\ 0 & 0 & 0 & \frac{1}{L} \tan(\delta_0(t)) & \frac{v_0(t)}{L} \sec^2(\delta_0(t)) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Matrix $\mathbf{C}$ is time-invariant, as the sensors (introduced in Section 4.1.1) directly measure the states. The matrix $\mathbf{A}(t)$ varies in time due to the dependence on $v_0(t)$, $\theta_0(t)$, and $\delta_0(t)$, which are the nominal values of the velocity and direction and steering angle, respectively. Meanwhile, the matrix $\mathbf{B}$ is time-invariant. In this simulation, we derive these values based on the closest point on the path to the vehicle's current position estimate. This is achieved by solving (4.4) using the state estimate $(\hat{x}, \hat{y})$ to find the path parameter λ^* , then using λ^* to find the nominal values:

$$\begin{aligned}v_0(t) &= v(\lambda^*), \\ \theta_0(t) &= \text{atan2}(y'_d(\lambda^*), x'_d(\lambda^*)), \\ \delta_0(t) &= 0,\end{aligned}$$

where $v(\lambda)$ is the velocity profile (4.9), $\text{atan2}(y, x)$ is the four-quadrant arctan function, and $x'_d(\lambda)$ and $y'_d(\lambda)$ are the derivatives of the path with respect to the path parameter λ . The nominal steering angle $\delta_0(t)$ is set to 0 for simplicity. This implies a piecewise linear approximation of the path, which is sufficient for the purposes of this simulation.

We discretize the linearized system using the step-invariant transformation with a sampling time of $T \geq 0$. The discretized system is:

$$\begin{aligned}\mathbf{x}[k+1] &= \mathbf{A}_d[k]\mathbf{x}[k] + \mathbf{B}_d[k]\mathbf{u}[k], \\ \mathbf{y}[k] &= \mathbf{C}\mathbf{x}[k],\end{aligned}\tag{4.14}$$

where $\mathbf{C}$ is the same as before, and $\mathbf{A}_d[k]$ and $\mathbf{B}_d[k]$ are the discretized system matrices:

$$\begin{aligned}\mathbf{A}_d[k] &= e^{\mathbf{A}(kT)T} \\ \mathbf{B}_d[k] &= \int_0^T e^{\mathbf{A}(kT)\tau} \mathbf{B}(kT) d\tau.\end{aligned}$$

In this simulation, we set $T = 0.01$ s.

4.3 Optimal Sensor Protection

In Section 3.7, we discussed how the concept of sparse observability extends naturally from time-invariant to time-varying systems, given that the system satisfies some uniform observability properties (Definition 7). In this section, we make use of this concept to determine the optimal set of sensors to protect for the vehicle model (4.14).

In general, it is difficult to verify if a given time-varying system satisfies Definition 7. However, we find through extensive randomized testing that the vehicle model (4.14) satisfies Definition 7 with $N = 5$. Thus, we make the following running assumption.

Assumption 1. *The linearized and discretized kinematic bicycle model (4.14) has uniform sparse observability with protection with $N = 5$.*

Under Assumption 1, we can now apply the tools developed in Section 3.4 to determine the optimal set of sensors to protect for this time-varying system. Say take our cost function to be the number of sensors protected, $c(\mathbf{P}) = \text{tr}(\mathbf{I} - \mathbf{P}) = |P|$, and a budget to protect at most 2 sensors, $b = 2$. Solving Problem 1 with these parameters gives the optimal sensor protection set $P = \{1, 2\}$, which corresponds to protecting the position sensors. This is intuitive, as we can derive other states like velocity and heading from the position sensors, but not the other way around.

If we increase the budget to 3, the solution is no longer unique. In this case, protecting sensors 1 and 2, along with any one of the remaining sensors, is optimal. Conversely, if we

reduce the budget to 1, the solution remains non-unique, as protecting any single sensor yields the same safety factor. This occurs because protecting only one sensor does not enhance the guaranteed detection of sensor faults.

To determine the optimal set of sensors to protect, we evaluated the safety factor function for all possible combinations of sensors up to the budget. With a cost function that is the number of sensors protected, $p = 6$ sensors, and a budget of $b = 2$, we evaluated $\binom{6}{0} + \binom{6}{1} + \binom{6}{2} = 22$ combinations. In general, for the same cost function, p sensors, and a budget of b , we need to evaluate the safety factor function $\sum_{i=0}^b \binom{p}{i}$ times. When the budget is small relative to the total number of sensors, the number of evaluations is much smaller than the exponential upper bound of 2^p .

Theorem 2 is also applicable to the time-varying case (with slight modification to replace the calculation of the system outputs). Using this protection scheme $P = \{1, 2\}$, we find that the sparse observability index is 4. This means that when there are at most 2 sensor faults, the measurement corruption detection algorithm can detect all of them. We now implement the measurement corruption detection algorithm in the simulation.

4.4 Measurement Corruption Detection

In Section 3.7, we introduced Algorithm 3 as a method to detect measurement corruptions in time-varying systems. In this section, we make use of this algorithm for the simulation.

Under Assumption 1, the main challenge in implementing Algorithm 3 for linearized discretized vehicle model (4.14) is determining the appropriate $\mathbf{A}[k]$ to use for each time step. This is because the system matrices $\mathbf{A}[k]$ are not directly a function of the time step k , but rather depend on the closest points on the desired path to the vehicle's state estimates and the associated target velocities. There are a few ways to approach this. We consider the following options:

1. Use the state estimates at each time step to determine the closest points on the path and the target velocities.
2. Use the latest state estimate to determine the closest point on the path, then “walk back” along the path using the desired motion model (target velocity) to determine the nominal states at each time step.
3. Use the oldest state estimate in the measurement window (of size N) to determine the closest point on the path, then “walk forward” along the path using the desired motion model to determine the nominal states at each time step.

In this simulation, we choose option 3 because the alternatives are more prone to contamination by sensor faults (i.e. older state estimates are less likely to be corrupted by sensor faults than newer ones because newer faults may not have been detected yet).

4.5 Simulation

We now present the simulation results of the measurement corruption detection algorithm (Section 4.4) on the vehicle model (4.1). All code presented in this section is available in the GitHub repository [68].

4.5.1 Initial Conditions

We set the initial conditions of the vehicle to be $(x_0, y_0, \theta_0, v_0, \delta_0) = (100, 200, 0, 0, 0)$, which corresponds to the vehicle starting near the path at rest with the steering angle set to 0. The vehicle is then controlled to follow the path and velocity profile using the control algorithm described in Section 4.1.4.

4.5.2 Fault Injection

We consider four types of sensor fault: bias, spike, drift, and noise. Examples of these faults are shown in Figure 4.9. We inject these faults with varying configurations and severity into the system. A summary of the parameters used for the generation of faults is given in Table 4.3.

During each simulation, we inject faults into the sensors at a random time point. Each fault is uniformly sampled from the ranges given in Table 4.3 for each sensor and fault type. Because the sparse observability analyses from Section 4.3 tell us that the system can detect up to 2 unprotected sensors with faults, faults are injected into up to 2 sensors at a time.

4.5.3 Detector Configuration

We configure the measurement detection algorithm (Algorithm 3) with the following parameters:

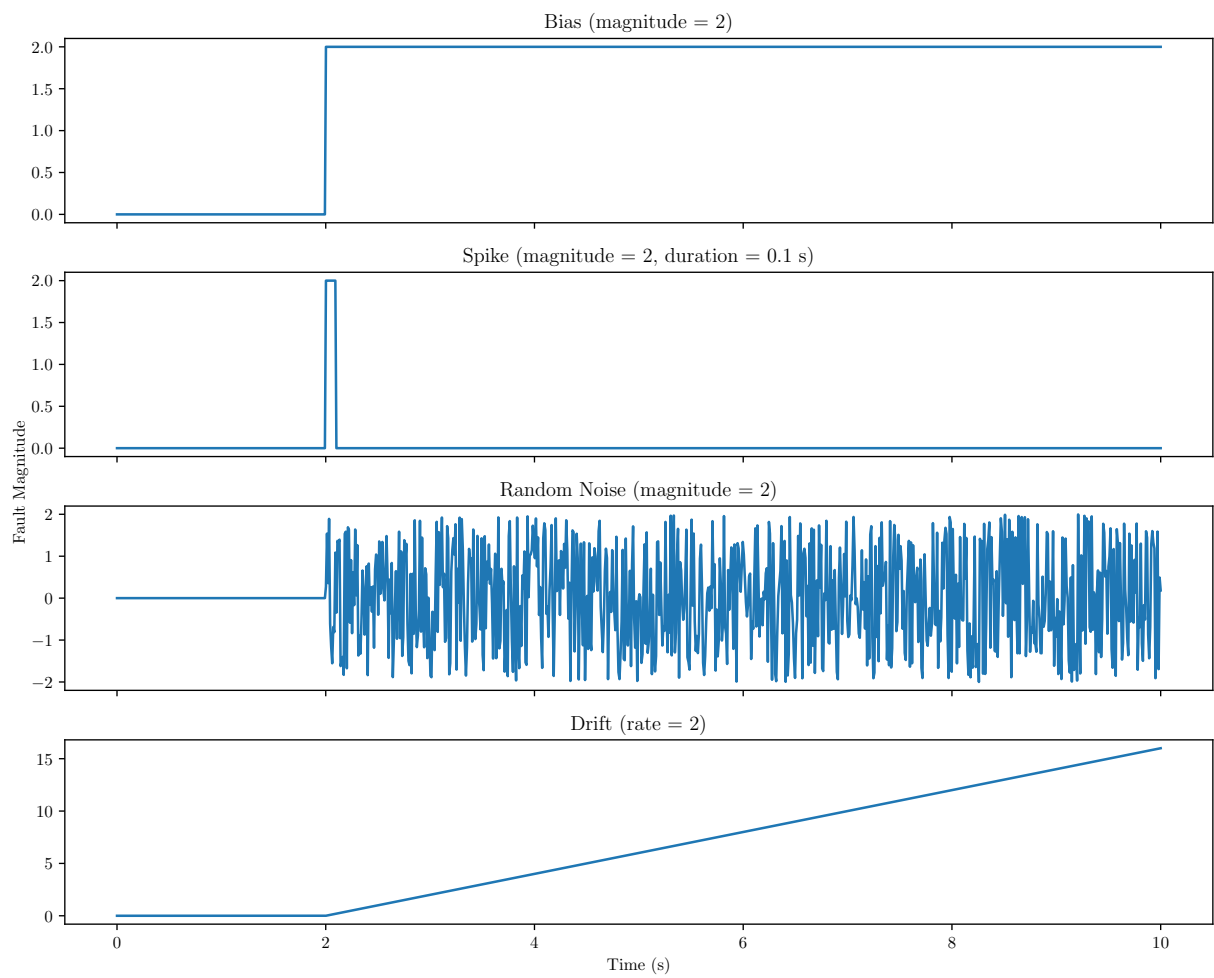


Figure 4.9: Examples of sensor faults

Table 4.3: Fault Parameter Ranges for Sensor Fault Injection

Fault Type	Sensor	Parameter Range
Bias	Heading	$-\pi/2$ to $\pi/2$ rad
	Velocity 1	-10 to 10 m/s
	Velocity 2	-10 to 10 m/s
	Steering Angle	$-\pi/4$ to $\pi/4$ rad
Spike	Heading	$-\pi/2$ to $\pi/2$ rad
	Velocity 1	-10 to 10 m/s
	Velocity 2	-10 to 10 m/s
	Steering Angle	$-\pi/4$ to $\pi/4$ rad
Noise	Heading	0 to $\pi/2$ rad
	Velocity 1	0 to 10 m/s
	Velocity 2	0 to 10 m/s
	Steering Angle	0 to $\pi/4$ rad
Drift	Heading	$-\pi/2$ to $\pi/2$ rad/s
	Velocity 1	-10 to 10 m/s ²
	Velocity 2	-10 to 10 m/s ²
	Steering Angle	$-\pi/4$ to $\pi/4$ rad/s

- $N = 5$: we set the window size to 5, the number of states of the system. Based on Assumption 1, this window size allows us to detect up to 2 sensor faults at any time point.
- ϵ is a tunable parameter. It can be set to allow for some uncertainty in the system matrices. In this simulation, we apply a scaler from a uniform distribution, $\lambda_\epsilon \sim \mathcal{U}(0, 10)$ to a vector preselected based on intuition of the underlying system, such that:

$$\epsilon = \lambda_\epsilon \begin{bmatrix} 1.5 & 1.5 & 0.3 & 1.5 & 1.5 & 0.3 \end{bmatrix}. \quad (4.15)$$

This gives us a simple tuning knob to control the trade-off between false positives and false negatives. We will see the effects of this parameter in Section 4.5.5.

- $\mathbf{C}$, Φ , $\mathbf{B}$ are computed using state estimates in the measurement window, as described in Section 4.4.

4.5.4 Simulation Setup

As discussed in Section 4.3, the simulated vehicle can detect up to 2 sensor faults. For each simulation, we inject randomly selected faults into up to 2 randomly selected unprotected sensors. We then run the measurement corruption detection algorithm (Algorithm 3) to detect the faults. We repeat this process for 5,000 simulations.

4.5.5 Results

To evaluate the detection performance, we use two primary metrics: **precision** and **recall**, which are computed as follows:

- **Precision** measures the proportion of correctly detected faulty sensors out of all detected sensors:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (4.16)$$

where TP (true positives) represents correctly detected faulty sensors, and FP (false positives) represents incorrectly flagged sensors.

- **Recall** measures the proportion of faulty sensors that are correctly detected:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (4.17)$$

where FN (false negatives) represents missed faulty sensors.

In our simulations, we compare these metrics across different types of injected faults to assess the robustness of the detection algorithm. For each simulation, the number of faulty sensors is known from the ground truth metadata, and the detected faults are obtained by checking sensor validity outputs from the detector.

The selection of the ϵ parameter is crucial for the performance of the detection algorithm. Using the one-parameter heuristic (λ_ϵ) as described in Section 4.5.3, we show that an appropriate choice of ϵ can lead to a good detection performance. In this simulation, we performed a preliminary simulation run and found that setting λ_ϵ to around 1 gives us the best precision and recall rates overall. We present the precision and recall values for different λ_ϵ values in Figure 4.10. Note that the absolute values of precision and recall are not important, as they are heavily influenced by the specific fault parameters (Table 4.3). Instead, we focus on the relative performance across different λ_ϵ values.

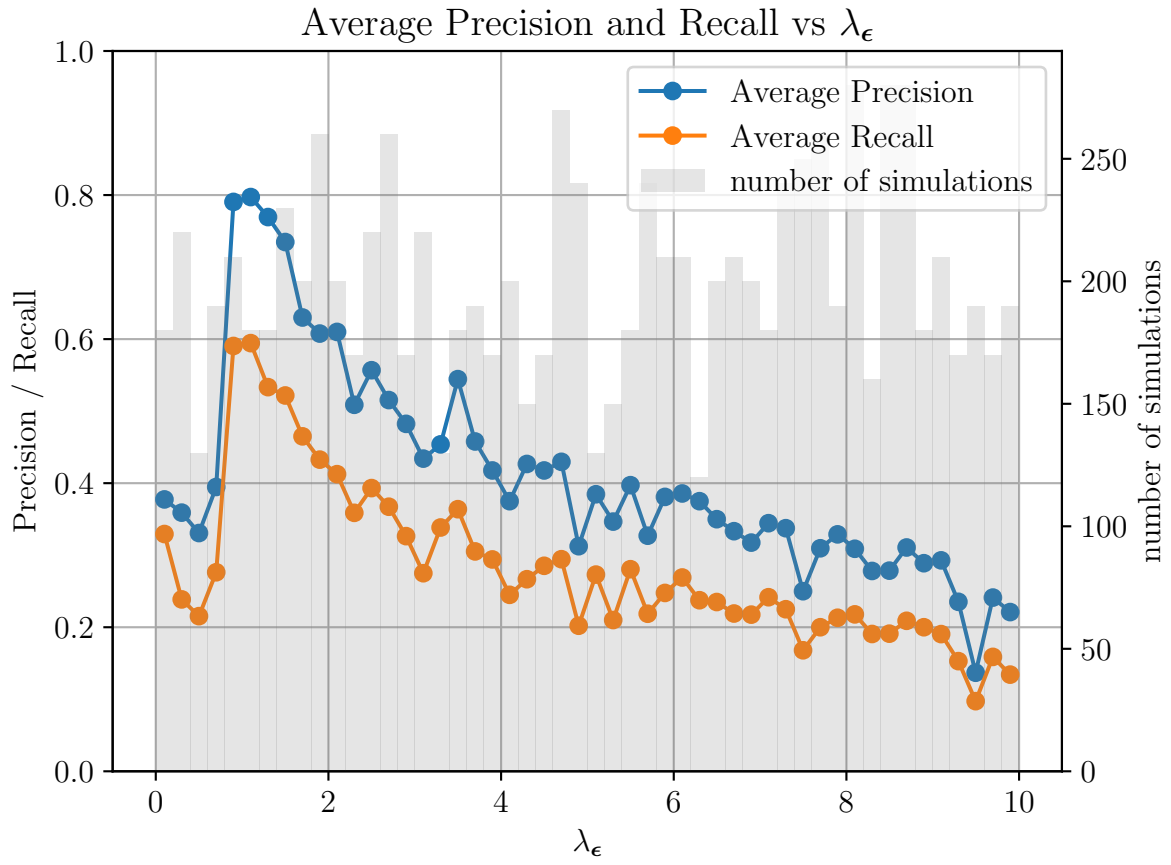


Figure 4.10: Average precision and recall for different λ_ϵ values

Type	Sen	Thresh	Below Threshold				Above Threshold	
			#	Det Rate	Det Delay	#	Det Rate	Det Delay
bias	3	0.90	82	0.07 ± 0.26	0.01 ± 0.00	73	1.00 ± 0.00	0.01 ± 0.00
	4	3.50	63	0.14 ± 0.35	0.01 ± 0.00	122	1.00 ± 0.00	0.01 ± 0.00
	5	3.00	41	0.29 ± 0.46	3.54 ± 4.14	118	1.00 ± 0.00	0.01 ± 0.00
	6	0.70	145	0.39 ± 0.49	0.26 ± 0.28	27	1.00 ± 0.00	0.01 ± 0.00
drift	3	0.80	83	0.25 ± 0.44	0.24 ± 0.21	78	1.00 ± 0.00	0.02 ± 0.00
	4	0.80	11	0.18 ± 0.40	0.05 ± 0.00	138	1.00 ± 0.00	0.02 ± 0.01
	5	0.00	0	N/A	N/A	157	1.00 ± 0.00	0.07 ± 0.31
	6	0.01	5	0.20 ± 0.45	0.60 ± 0.00	161	1.00 ± 0.00	0.06 ± 0.08
noise	3	1.50	126	0.69 ± 0.46	0.23 ± 0.95	7	1.00 ± 0.00	0.02 ± 0.01
	4	1.50	25	0.20 ± 0.41	3.07 ± 1.94	118	1.00 ± 0.00	0.03 ± 0.07
	5	1.00	9	0.00 ± 0.00	N/A	174	1.00 ± 0.00	0.26 ± 1.63
	6	0.31	64	0.03 ± 0.18	5.04 ± 3.39	83	1.00 ± 0.00	0.10 ± 0.24
spike	3	0.80	76	0.01 ± 0.11	0.02 ± 0.00	71	1.00 ± 0.00	0.01 ± 0.00
	4	3.00	57	0.11 ± 0.31	0.01 ± 0.00	109	1.00 ± 0.00	0.01 ± 0.00
	5	3.00	53	0.02 ± 0.14	0.01 ± 0.00	96	1.00 ± 0.00	0.01 ± 0.00
	6	0.65	125	0.10 ± 0.30	0.02 ± 0.02	25	1.00 ± 0.00	0.01 ± 0.00

Table 4.4: Summary of detection rate and detection delay for single-sensor faults, separated by threshold values for fault magnitude or drift rate. Below-threshold and above-threshold statistics are shown for each fault type and sensor.

With the chosen ϵ parameter (by setting $\lambda_\epsilon = 1$ in (4.15)), we analyze the distribution of precision, recall, and detection delay across 5,000 simulation runs to assess the effectiveness of the fault detection algorithm under various conditions. A consistent pattern is observed across all fault types: for each sensor, there is a threshold of fault magnitude (i.e., magnitude for bias, noise, and spike faults, and drift rate for drift faults) above which fault detection is nearly perfect, and below which performance drops sharply. The detection performance data is summarized in Table 4.4.¹ As an example of this threshold effect, Figure 4.11 shows the detection rate for bias faults on sensor 3.

Using the same thresholds as shown in Table 4.4, we evaluate detector performance for two-sensor faults. Similar to the one-sensor fault case, the detector achieves near-perfect

¹Note that the precision and recall metrics are identical for single-sensor faults, so we combine them into a single column labeled “Detection Rate”.

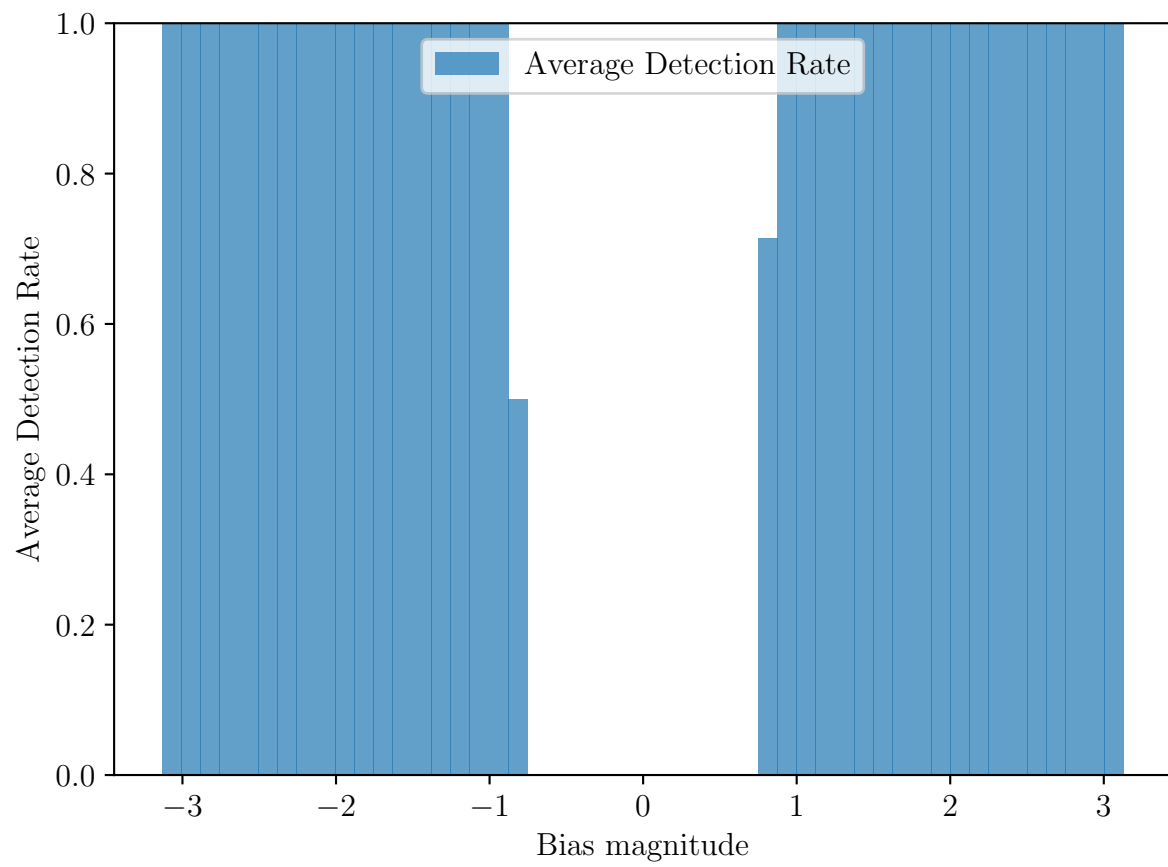


Figure 4.11: Detection rate for bias faults in sensor 3 (heading sensor θ_m). Notice that the detection rate experiences a sharp change when the fault magnitude exceeds a threshold.

precision and recall for above-threshold faults. For below-threshold faults, performance degrades but does not exhibit the sharp drop seen in the one-sensor case. The results for two-sensor faults are summarized in Table 4.5 (above-threshold) and Table 4.6 (below-threshold).

Sensors	3	4	5	6
3	-	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
4	1.00 ± 0.00	-	1.00 ± 0.00	0.99 ± 0.06
5	1.00 ± 0.00	1.00 ± 0.00	-	1.00 ± 0.00
6	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	-

Sensors	3	4	5	6
3	-	0.70 ± 0.25	0.76 ± 0.25	0.55 ± 0.15
4	0.70 ± 0.25	-	0.66 ± 0.24	0.59 ± 0.19
5	0.68 ± 0.24	0.68 ± 0.24	-	0.58 ± 0.19
6	0.60 ± 0.21	0.60 ± 0.20	0.60 ± 0.20	-

Table 4.5: Detection precision and recall (mean $\pm$ standard deviation) for two-sensor faults. Above threshold. Top: Precision, Bottom: Recall.

4.5.6 Timing Analysis

We evaluate the algorithm’s running time by measuring the time taken by the detector to complete under different conditions. The algorithm is run on a single core of an Intel i5-7600 CPU, and faults are injected into the simulation at the 30-second mark. The time taken by the detector to complete is recorded for each condition. As shown in Figure 4.12, the detector runs at approximately 0.003 seconds per tick under nominal (no-fault) conditions, 0.005 seconds per tick with one fault, and 0.008 seconds per tick with two faults. The fast execution under nominal conditions is due to the need to evaluate only the no-fault case in the optimization problem (3.12). When a fault is present, the algorithm evaluates increasingly complex fault hypotheses—starting with the no-fault case, then one-fault scenarios, and so on—short-circuiting once a valid solution is found.

Because the simulation operates at 100 Hz (0.01 seconds per tick) and the maximum number of faults considered is two, the detector can run in real time at each tick. For

Sensors	3	4	5	6
3	-	0.83 ± 0.36	0.84 ± 0.33	0.90 ± 0.30
4	0.78 ± 0.39	-	0.97 ± 0.16	0.83 ± 0.37
5	0.87 ± 0.33	0.98 ± 0.12	-	0.89 ± 0.31
6	0.88 ± 0.32	0.90 ± 0.29	0.85 ± 0.36	-

Sensors	3	4	5	6
3	-	0.47 ± 0.23	0.48 ± 0.21	0.46 ± 0.17
4	0.41 ± 0.20	-	0.51 ± 0.13	0.44 ± 0.22
5	0.45 ± 0.18	0.52 ± 0.12	-	0.48 ± 0.21
6	0.45 ± 0.18	0.48 ± 0.19	0.46 ± 0.23	-

Table 4.6: Detection precision and recall (mean $\pm$ standard deviation) for two-sensor faults. Below threshold. Top: Precision, Bottom: Recall.

larger systems, it may be necessary to run the detector at a lower frequency, which would increase detection delay. Alternatively, as demonstrated in [20], a relaxed version of the detection algorithm can be used to further reduce running time, albeit with weaker theoretical guarantees.

4.6 Case Study

The [IAC](#) is a competition that showcases the capabilities of autonomous racing technology. In October 2021, during a time-trial event at the [IMS](#), one of the competing vehicles experienced a sensor failure that led to a crash. This incident provides a valuable real-world case study for evaluating our measurement corruption detection algorithm. By analyzing telemetry data from the crash, we seek to understand its root cause and assess whether our algorithm could have preemptively identified the sensor fault. We begin with an overview of the vehicle’s configuration and race conditions, then reconstruct the failure sequence. Finally, we use the optimal sensor protection problem from Section 3.4 to identify critical sensors before applying our detection algorithm from Section 3.6 to determine if the crash could have been avoided.

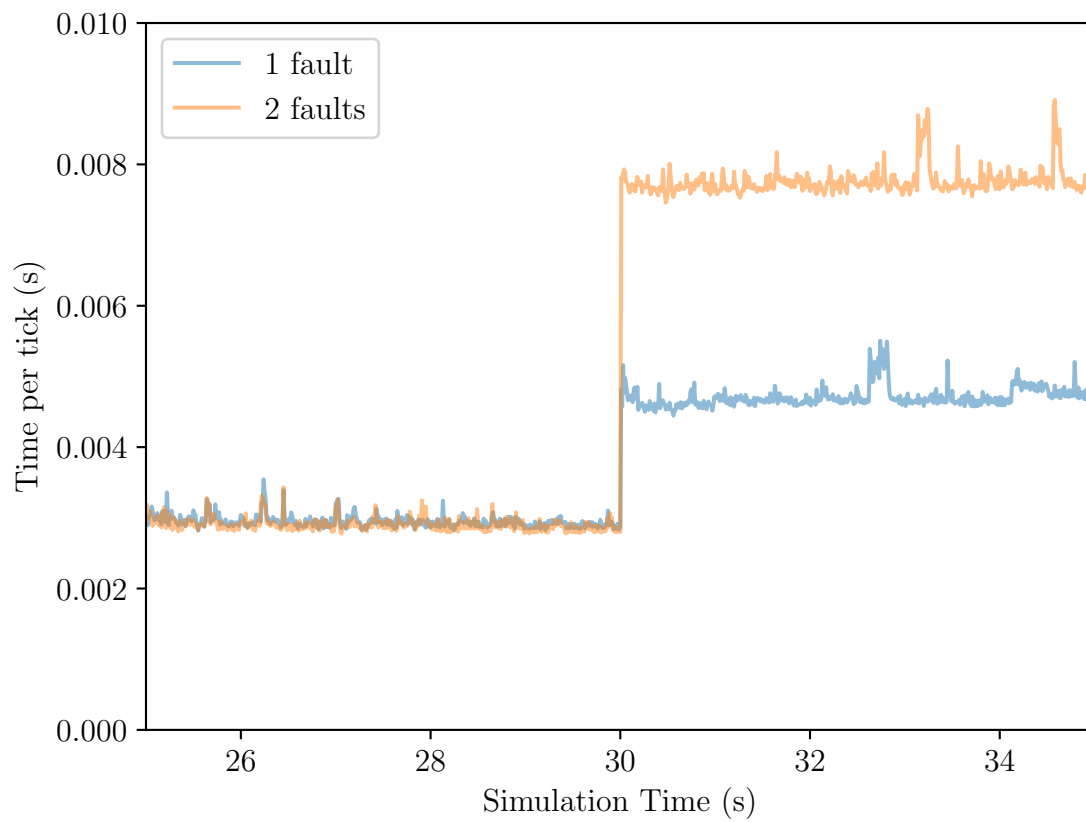


Figure 4.12: Detector running time. Faults are injected at the 30-second mark.

4.6.1 The IAC Racing Incident

The [IAC](#) vehicle, named AV-21, is a racecar equipped with the following sensors:

- 2 Inertial Navigation Systems (INSs) that each bundles a Global Navigation Satellite System (GNSS) receiver and an Inertial Measurement Unit (IMU) to provide position, velocity, and orientation information, mounted near the center of mass on the chassis of the vehicle,
- 4 wheel encoders that provide wheel speed information,
- 1 steering angle sensor,
- Lidar, radar, and camera sensors that provide 360-degree perception around the vehicle.

The vehicle is controlled by acceleration and steering angle commands generated by the onboard computer. These commands are converted into throttle, braking, gear, and steering torque commands by a lower level controller, which are then applied to the vehicle actuators.

The race was a time-trial event where autonomous vehicles competed to complete a fixed number of laps around a closed circuit in the shortest time possible. The vehicle was programmed to follow an optimized racing line with a corresponding velocity profile to minimize lap time. The racing line, shown in [Figure 4.13](#), was designed offline and did not change during the race.

During the event, the vehicle operated with a reduced sensor suite due to technical limitations. Only one of the two [INSs](#) was functional, providing position, velocity and orientation data. The front wheel encoders were averaged to produce a single speed measurement, and the steering angle sensor provided steering feedback. The perception sensors (Lidar, radar, and cameras) were disabled since they were not needed for the time-trial format. The locations of the active sensors used during the event are shown in [Figure 4.14](#).

The sensors and actuators operated at different frequencies, as shown in [Table 4.7](#).

During the warm-up lap, the vehicle was following a constant velocity of 30 m/s along the racing line. As the vehicle exited the last turn, a sudden fault occurred in the heading and velocity measurements coming from the [INS](#) package. The state estimator initially maintained a heading estimate close to the true values due to prior information and filter dynamics. However, as the faulty measurements persisted, the estimate gradually drifted

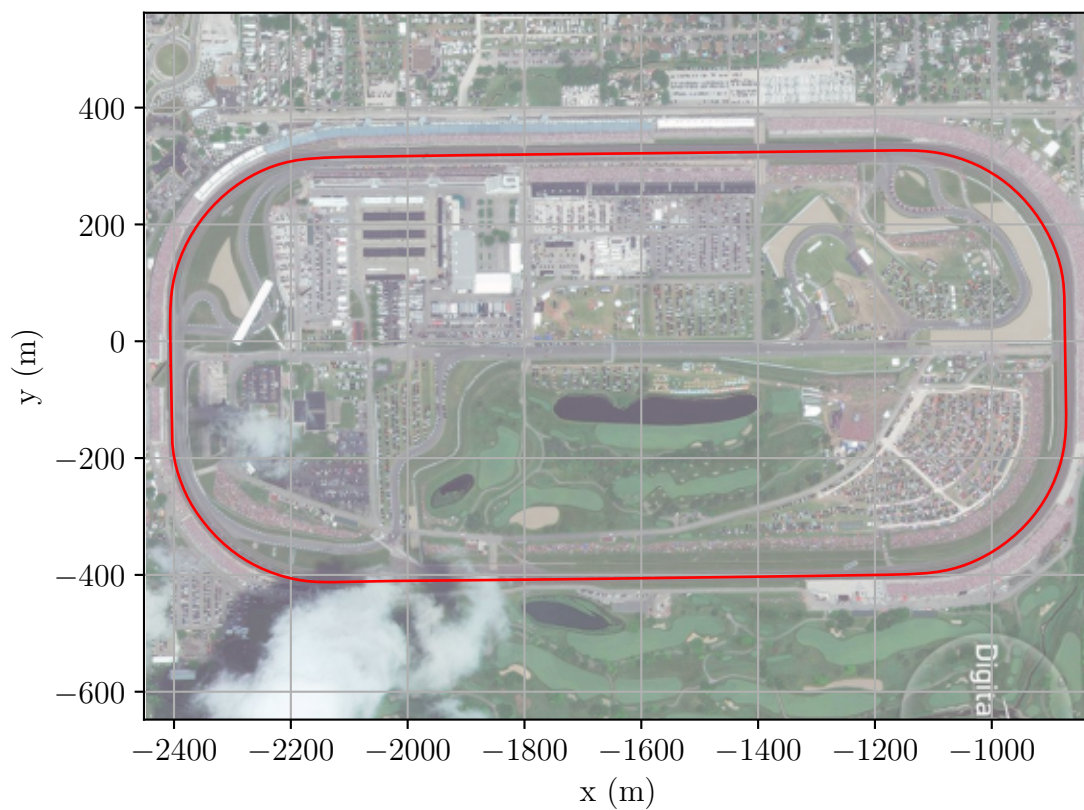


Figure 4.13: Racing line for the time-trial event. The vehicle starts in the pit area at the top straight and travels counterclockwise. Coordinates are in the Local Tangent Frame. Background image from [19].

Component	Frequency (Hz)
INS (position)	20
INS (heading, velocity)	100
Wheel Odometry	100
Steering Actuation	25
Throttle/Brake	100

Table 4.7: Sensor and actuator frequencies for AV-21.



Figure 4.14: Sensor setup for AV-21 during the time-trial. 1. GNSS antennas, 2. steering angle sensor (inside the cabin), 3. wheel encoder (near the axle), 4. INS (inside the cabin).

away and ultimately converged to an incorrect heading. The velocity estimate remained largely unaffected by the fault, as the estimator was designed to prioritize wheel odometry measurements. This choice was based on previous experience indicating that wheel odometry is more reliable than the INS velocity measurement. The controller, relying on the faulty heading estimate, responded accordingly and commanded incorrect steering inputs, ultimately driving the vehicle into a wall. Figure 4.15 shows a bird’s-eye view of the vehicle trajectory, highlighting the points where the vehicle experienced the fault and collided with the wall. Figure 4.16 shows heading and velocity data from the INS package, as well as the estimated states. There is a known bias in the velocity data from the INS package, which is corrected in the estimator by only taking into account the derivative of the velocity signal.

Our goal is to evaluate whether our measurement corruption detection algorithm could have identified the sensor fault early enough to prevent the crash.

4.6.2 Vehicle Model

We employ a simplified vehicle model to represent the race car’s dynamics. While this model does not fully capture the dynamics during a race, it is adequate for the warm-up lap scenario, where the vehicle’s velocity changes slowly. The equations of motion are as follows:

$$\begin{aligned}\dot{x}(t) &= v(t) \cos(\theta(t)) \\ \dot{y}(t) &= v(t) \sin(\theta(t)) \\ \dot{\theta}(t) &= \frac{v(t)}{L} \tan(\delta(t)) \\ \dot{v}(t) &= 0\end{aligned}\tag{4.18}$$

where x, y, θ, v are the position and heading of the vehicle, $\dot{v}(0) = v_0$ is a constant velocity, L is the wheelbase length, and δ is the steering angle input. As described in Section 4.6.1, the measured variables are given by $x_m, y_m, \theta_m, v_{m,1}, v_{m,2}$.

The nonlinear equations of motion (4.18) are linearized around the reference trajectory $(x_0(t), y_0(t), \theta_0(t), v_0, \delta_0(t))$. The resulting linearized system is given by:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t)\end{aligned}\tag{4.19}$$

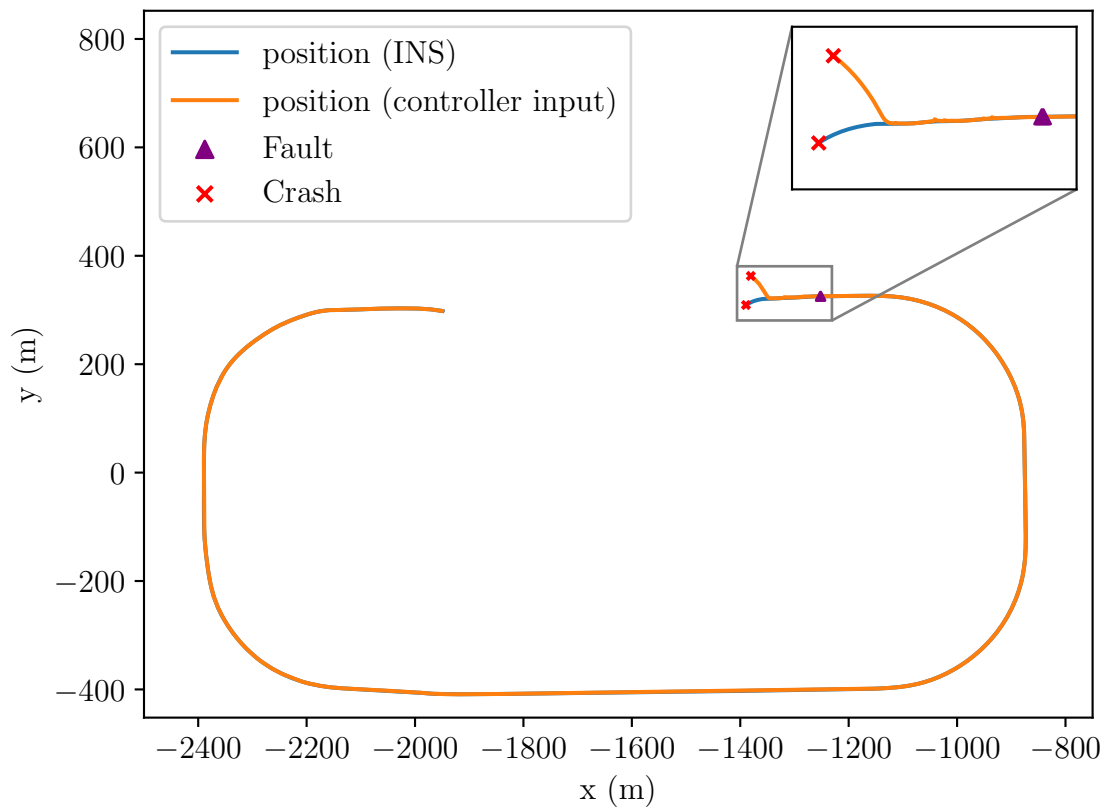


Figure 4.15: Bird's-eye view of the vehicle trajectory, highlighting the points where the vehicle experienced the fault and collided with the wall.

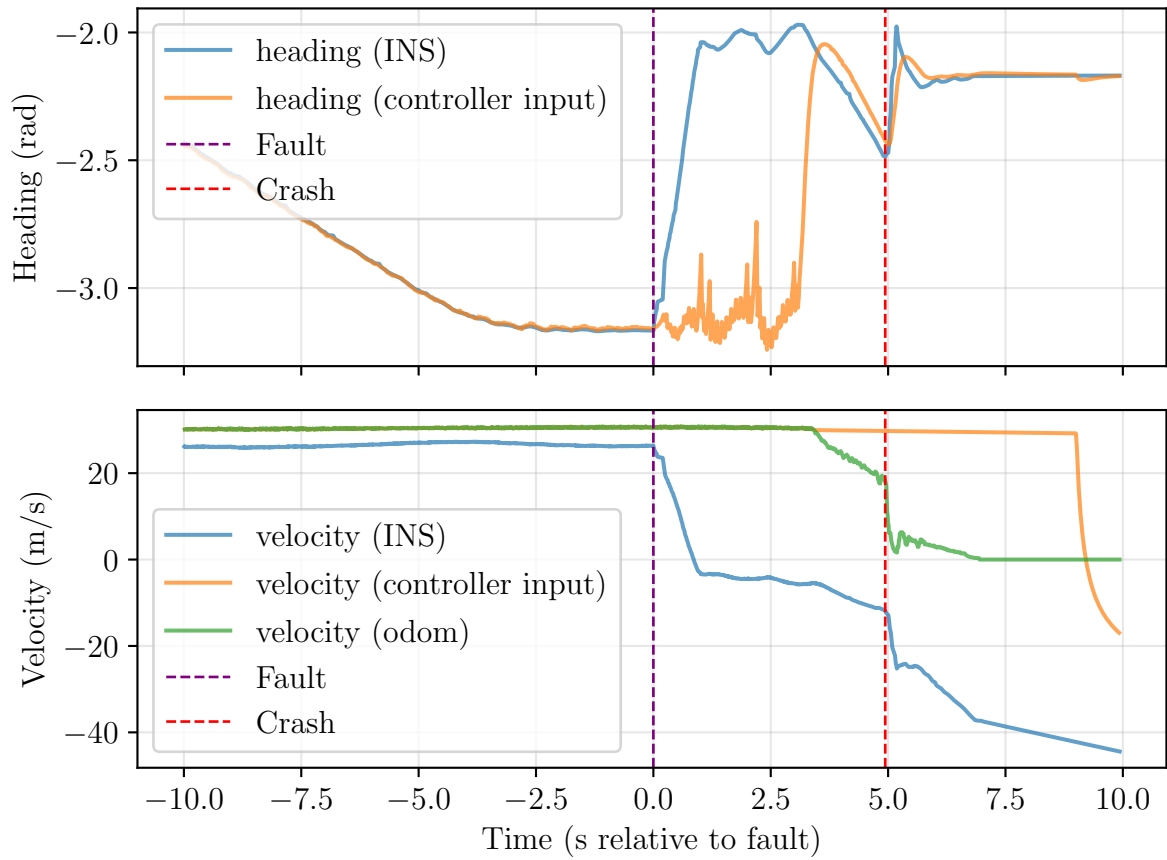


Figure 4.16: Heading and velocity data, highlighting the fault and crash times.

where $\mathbf{x}(t) = [x(t), y(t), \theta(t), v(t)]^T - [x_0(t), y_0(t), \theta_0(t), v_0]^T$ is the state vector, $\mathbf{u}(t) = \delta(t) - \delta_0(t)$ is the input, and the system matrix $\mathbf{A}(t)$, input matrix $\mathbf{B}(t)$, and output matrix $\mathbf{C}$ are:

$$\mathbf{A}(t) = \begin{bmatrix} 0 & 0 & -v_0 \sin(\theta_0(t)) & \cos(\theta_0(t)) \\ 0 & 0 & v_0 \cos(\theta_0(t)) & \sin(\theta_0(t)) \\ 0 & 0 & 0 & \frac{\tan(\delta_0(t))}{L} \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.20)$$

$$\mathbf{B}(t) = \begin{bmatrix} 0 \\ 0 \\ \frac{v_0}{L \cos^2(\delta_0(t))} \\ 0 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We discretize the system in the same way as in Section 4.2, using the step-invariant transformation with a sampling time of $T \geq 0$. The discretized system is given by:

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{A}_d[k] \mathbf{x}[k] + \mathbf{B}_d[k] \mathbf{u}[k], \\ \mathbf{y}[k] &= \mathbf{C} \mathbf{x}[k], \end{aligned} \quad (4.21)$$

where $\mathbf{C}$ is the same as before, and $\mathbf{A}_d[k]$ and $\mathbf{B}_d[k]$ are the discretized system matrices:

$$\begin{aligned} \mathbf{A}_d[k] &= e^{\mathbf{A}(kT)T} \\ \mathbf{B}_d[k] &= \int_0^T e^{\mathbf{A}(kT)\tau} \mathbf{B}(kT) d\tau. \end{aligned}$$

In this scenario, we set $T = 0.05\text{s}$ (20 Hz) to match the frequency of the slowest sensor (see Table 4.7). For data at higher frequencies, we downsample them to 20 Hz. In principle, it might be more accurate to sample at the greatest common divisor (GCD) of the sensor frequencies (5 Hz), thereby ensuring all measurements are perfectly aligned. However, using 5 Hz reduces the total number of data points, potentially increasing modeling error. Moreover, jitter in the system may still cause some misalignment even at the slower sampling rate.

4.6.3 Optimal Sensor Protection and Measurement Corruption Detection

Similar to Section 4.3, through extensive randomized testing, we find that the vehicle model (4.21) satisfies Definition 7 with $N = 4$. Thus, we make the following assumption.

Assumption 2. *The linearized and discretized vehicle model (4.21) has uniform sparse observability with protection with $N = 4$.*

With Assumption 2, we can apply the optimal sensor protection algorithm to determine the best set of sensors to protect. Using the same cost function, $c(P) = |P|$, and budget $b = 2$ as in Section 4.3, the optimal set of sensors to protect is $P = \{1, 2\}$. This result aligns with intuition, as in Section 4.3: position measurements allow us to infer heading and velocity, but not vice versa. This also explains why the car has two INS packages and why operating with only one is risky. In fact, in subsequent competitions, the vehicle was outfitted with an additional INS package from a different manufacturer, to allow for a variety of fault-tolerant configurations.

We apply the measurement corruption detection algorithm (Algorithm 3) to the crash data to detect the faults. The parameter ϵ is set to $[2.5, 2.5, 0.55, 5, 1]$. This choice is based on empirical characteristics of the sensors (such as noise levels and biases) and iterative testing on nominal operation data to eliminate false positives. Developing a more principled approach—such as optimization-based tuning or statistical analysis—remains an important direction for future research.

The algorithm successfully detected both faults, with a delay of 0.10 seconds for the velocity fault and 0.35 seconds for the combined faults. Figure 4.17 illustrates the detection times. This leaves over 4 seconds before the crash, which is likely enough time to take corrective action.

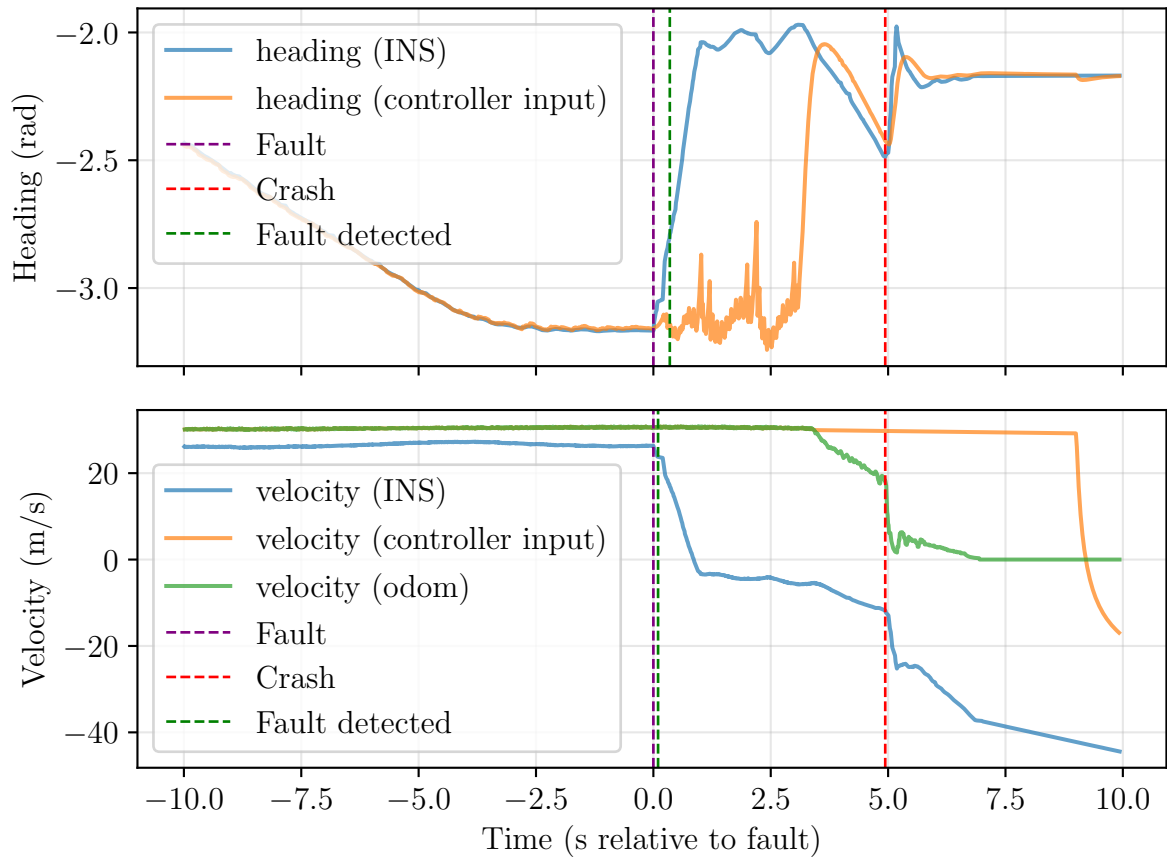


Figure 4.17: Detection times for the velocity and heading faults.

Chapter 5

Deployment to a Skid-Steer Robot

In this chapter, we take the same approach as in Chapter 4 to find the optimal sensor protection strategy for a physical robot. We deploy the fault detection algorithm and introduce a simple fault mitigation heuristic. We show that the fault detection algorithm can be easily integrated into a typical control pipeline to provide an additional layer of robustness to the robot. It provides a valuable real-world example of integrating the fault detection algorithm into an existing control pipeline.

5.1 Hardware and Sensor Setup

In this study, we use a Clearpath Jackal robot, a compact skid-steer mobile platform. It features four fixed wheels arranged symmetrically, with two on each side. Steering and angular motion are controlled by varying the rotational velocity between the left and right wheels, enabling precise maneuverability without the need for dedicated steering actuators. A schematic showing the side and front views of the Jackal is presented in Figure 5.1. A photograph of the Jackal platform used in our experiments is shown in Figure 5.2.

We utilize the Jackal’s standard sensor package, which includes wheel encoders for linear and rotational velocity measurements (v, ω), running at 50 Hz, and an inertial measurement unit (IMU) for rotational velocity measurements (ω), running at 20 Hz. Additionally, we integrate a VICON motion capture system to simulate global odometry in a controlled lab environment, providing absolute measurements of position (x, y) and orientation (θ) in the map frame. VICON produces data at 60 Hz, and we downsample it to 1 Hz to emulate the frequency of the onboard GPS. A summary of the sensor suite and their measurement rates is shown in Table 5.1.

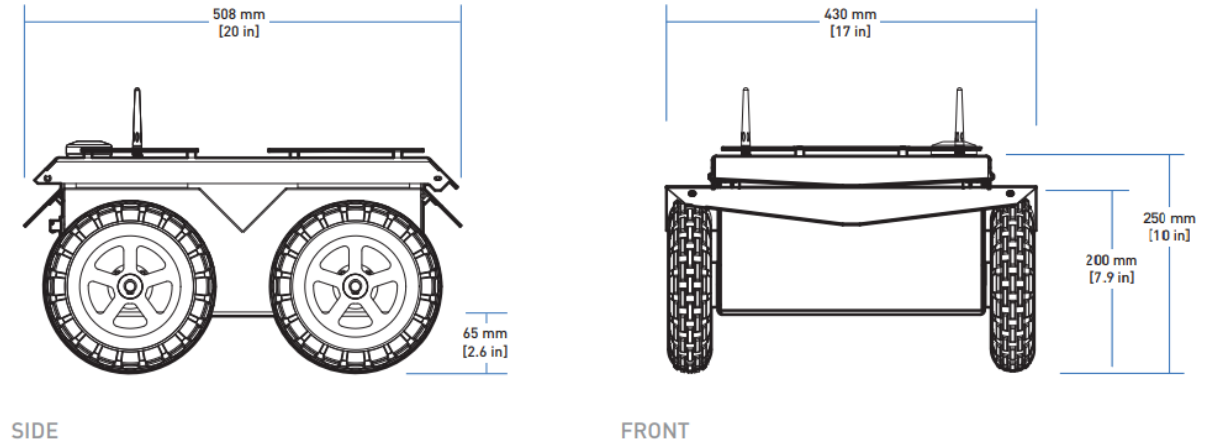


Figure 5.1: A schematic showing the side and front views of the Jackal [15].

Sensor	x	y	θ	v	ω	rate (Hz)
GPS (VICON)	✓	✓	✓	—	—	1
Wheel Encoders	—	—	—	✓	✓	50
IMU	—	—	—	—	✓	20

Table 5.1: Summary of sensors and their measurement rates.

To control the vehicle, we provide linear and angular acceleration commands (a, α) , which are converted to motor voltages by a built-in low-level controller.

The onboard computer is equipped with an Intel i5-4570TE processor and 8 GiB of RAM, running Ubuntu 18.04. It communicates with the VICON motion capture system via a 60 GHz low-latency, high-bandwidth wireless connection using a pair of MikroTik wAP60G "Wireless Wire" devices. Our setup runs the [Robot Operating System \(ROS\)](#) Noetic stack within Docker containers on the onboard computer. The sensor measurements come from the provided sensor drivers [57] for the Jackal and the VICON ROS bridge [1].

5.2 Mathematical Model

We model the robot using a simple dynamic unicycle model. The states are position (x, y) , heading (θ) , linear velocity (v) , and angular velocity (ω) . Inputs to the system are linear acceleration (a) and angular acceleration (α) . The equations of motion are given by:



Figure 5.2: A photograph of the Jackal platform used in the study.

$$\begin{aligned}
\dot{x}(t) &= v(t) \cos \theta(t) \\
\dot{y}(t) &= v(t) \sin \theta(t) \\
\dot{\theta}(t) &= \omega(t) \\
\dot{v}(t) &= a(t) \\
\dot{\omega}(t) &= \alpha(t)
\end{aligned} \tag{5.1}$$

We denote the measured variables corresponding to sensors shown in Table 5.1 with $x_m, y_m, \theta_m, v_m, \omega_{m,1}, \omega_{m,2}$.

5.2.1 Control Objective

The control objective is to track a circular path with radius $R = 2$ meters. The reference path is given by:

$$x_d(\lambda) = R \cos(\lambda), \quad y_d(\lambda) = R \sin(\lambda) \tag{5.2}$$

where x_d and y_d are the desired coordinates, and $\lambda \in [0, 2\pi)$ is the angular position along the circle. We call λ the *path parameter* to be consistent with the notation used in Chapter 4. The reference path is shown in Figure 5.3.

To keep the system implementation general, the reference path is discretized into a piecewise-linear approximation with $N_d = 100$ points evenly distributed along the circle:

$$\lambda_i = \frac{2\pi i}{N_d}, \quad i \in \{0, 1, \dots, N_d - 1\}. \tag{5.3}$$

The path discretization component also computes and provides the reference heading, curvature, and velocity at each point, which are given by:

$$\begin{aligned}
\theta_d(\lambda_i) &= \text{atan2}(y'_d(\lambda_i), x'_d(\lambda_i)) \\
\kappa_d(\lambda_i) &= \frac{1}{R} \\
v_d(\lambda_i) &= v_0
\end{aligned} \tag{5.4}$$

where $\text{atan2} : \mathbb{R}^2 \setminus \{0\} \rightarrow (-\pi, \pi]$ is the four quadrant arctangent function, and x'_d and y'_d are the derivatives of x_d and y_d with respect to λ . In this study, we use a constant velocity of $v_0 = 0.5$ m/s.

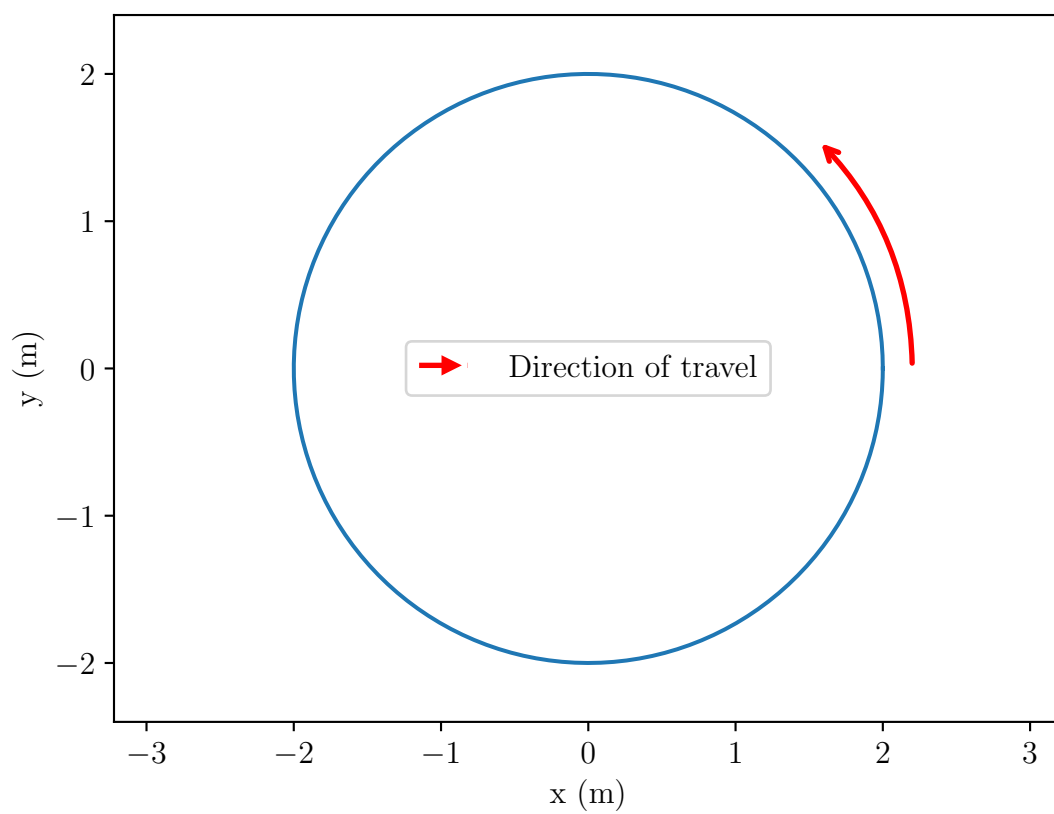


Figure 5.3: The reference path, a circle with radius $R = 2$ meters.

Similar to Section 4.1, we obtain the path parameter λ as a function of time t by taking the closest point on the path to the vehicle's position estimate $(\hat{x}(t), \hat{y}(t))$ at time t :

$$\arg \min_{\lambda} (\hat{x}(t) - x_d(\lambda))^2 + (\hat{y}(t) - y_d(\lambda))^2. \quad (5.5)$$

The solution $\lambda^*(t)$ to this optimization problem gives the closest point on the path to the vehicle's position estimate at time t .

This discretized representation allows downstream components to implement control algorithms without considering the original mathematical form of the path and enables modifications to the path as needed.

5.2.2 Model Linearization and Discretization

The nonlinear equations of motion (5.1) are linearized around the reference trajectory. Calculating the Jacobians of the system with respect to the states and inputs yields:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t). \end{aligned}$$

where matrices $\mathbf{A}(t)$, $\mathbf{B}$, and $\mathbf{C}(t)$ are as follows:

$$\mathbf{A}(t) = \begin{bmatrix} 0 & 0 & -v_d(t) \sin(\theta_d(t)) & \cos(\theta_d(t)) & 0 \\ 0 & 0 & v_d(t) \cos(\theta_d(t)) & \sin(\theta_d(t)) & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that the measurement matrix $\mathbf{C}$ is a nominal measurement matrix, which assumes that all sensors are available. When we discretize the system, we will introduce a time-varying measurement matrix to account for the different sampling rates of the sensors.

To discretize the system, we approximate the discrete-time state-space equations as:

$$\begin{aligned}\mathbf{x}[k+1] &= \mathbf{A}_d[k]\mathbf{x}[k] + \mathbf{B}_d\mathbf{u}[k] \\ \mathbf{y}[k] &= \mathbf{C}_d[k]\mathbf{x}[k]\end{aligned}\tag{5.6}$$

where $\mathbf{A}_d[k]$ and $\mathbf{B}_d$ are the discrete-time system matrices, obtained using a forward Euler discretization with a time step of $T = 0.05$ seconds:

$$\mathbf{A}_d[k] = \mathbf{I} + T\mathbf{A}(kT), \quad \mathbf{B}_d = T\mathbf{B}.$$

Unlike the step-invariant transformation used in Chapter 4, this earlier experiment employs a simpler first-order Euler discretization. While the step-invariant transformation provides improved numerical accuracy and stability, the choice of first-order Euler was sufficient for this particular deployment, as the system operates at relatively low speeds.

The measurement matrix $\mathbf{C}_d[k]$ is time-varying depending on the availability of sensor measurements at each time step. It is given by:

$$\mathbf{C}_d[k] = \begin{cases} \mathbf{C}, & \text{if position sensor data is available at time step } k, \\ \mathbf{C}_{\{3,4,5,6\}}, & \text{otherwise.} \end{cases}$$

where $\mathbf{C}_{\{3,4,5,6\}}$ is the submatrix of $\mathbf{C}$ that excludes the first two rows, corresponding to the position sensors. This time-varying measurement matrix is within the specifications of the time-varying system model described in Section 3.7.

5.3 Optimal Sensor Protection

We apply the optimal sensor protection analysis from Section 3.4 to the model derived in Section 5.2.2. We find through extensive numerical testing that the vehicle model (5.6) satisfies Definition 7 with $N = 60$. Note that unlike the simulation and the case study in Chapter 4, the measurement window N is larger than the number of states $n = 5$. This is because of the time-varying nature of the measurement matrix $\mathbf{C}_d[k]$. Under ideal conditions, GPS measurements are available once every second. Given our sampling rate of 20 Hz (0.05 seconds per tick), we expect a GPS measurement every 20 ticks. Thus, choosing $N = 60$ ensures exactly 3 GPS measurements within our measurement window. This is intuitive: 2 position measurements would allow inference of average velocity and heading but would be insufficient to determine angular velocity. 3 position measurements allows us to have 2 average heading estimates, which is sufficient to determine the average

angular velocity. In other words, retaining only the position measurements allows us to infer all other states, given sufficient time. Since the position measurements constitute 2 out of the 6 sensors, we achieve the maximum sparse observability of 4.

Based on the numerical testing, we make the following running assumption:

Assumption 3. *The linearized and discretized unicycle model (5.6) has uniform sparse observability with protection with $N = 60$.*

Under Assumption 3, we can now apply the tools developed in Section 3.4 to determine the optimal set of sensors to protect for this time-varying system. Consistent with the simulation and case study in Chapter 4, we take our cost function to be the number of sensors protected, $c(\mathbf{P}) = \text{tr}(\mathbf{I} - \mathbf{P}) = |P|$, and a budget to protect at most 2 sensors, $b = 2$. Solving Problem 1 with these parameters gives the optimal sensor protection set $P = \{1, 2\}$, which corresponds to protecting the position sensors. This finding is consistent with the previous results from Chapter 4, where we found that the position sensors are the most critical for the kinematic bicycle model as well.

Similar to Section 4.3, to determine the optimal sensor protection set, we evaluated the safety factor function for all possible subsets of sensors up to the budget, which resulted in a total of $\sum_{i=0}^b \binom{p}{i} = \binom{6}{0} + \binom{6}{1} + \binom{6}{2} = 22$ evaluations. Since the number of sensors is small, this can be done quickly.

5.4 Fault Detection and Mitigation

A standard control pipeline for the Jackal is shown in Figure 5.4. We use the Extended Kalman Filter (EKF) implementation in the `robot_localization` package [41] to estimate the state of the vehicle, a simple pure-pursuit controller to track the reference path based on the waypoints from Section 5.2.1 and the path parameter λ computed from the vehicle's position estimate using Equation (5.5), and a PID controller to track the reference velocity.

We integrate the fault detection algorithm from Section 3.7 into the control pipeline. Other than a ROS wrapper, the implementation is identical to the one described in Chapter 4 (though an earlier version of the algorithm was used). Due to jitter in the sensor measurements, we use a conservative measurement window of $N = 80$ ticks (4 seconds). This allows us to have 4 position measurements within the measurement window assuming no jitter, or 3 position measurements with some jitter, which by the discussion in Section 5.3 and Theorem 2, is sufficient to detect up to 2 sensor faults that occur in non-position measurements.

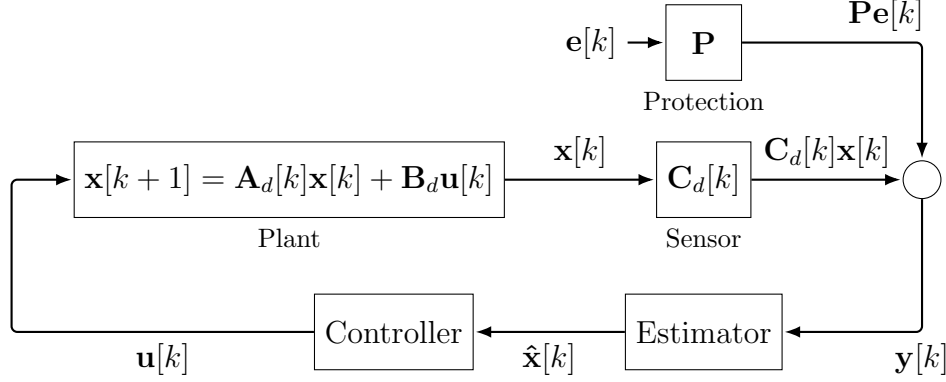


Figure 5.4: A nominal control pipeline for the Jackal.

The ϵ used in Algorithm 3 is obtained in the same way as Chapter 4, by finding the minimum value that cause no false positives during nominal operation. For this deployment, we set $\epsilon = [0.08 \ 0.08 \ 0.17 \ 0.16 \ 0.25 \ 0.25]$.

As in Section 4.4, we can choose among three linearization strategies for the fault detection algorithm: (i) using the position estimate at each time step, (ii) walking backward along the reference trajectory using the latest position estimate, or (iii) walking forward using the earliest position estimate. In this study, we adopt the “walking backward” approach, as it was the only option under consideration at this stage of deployment.

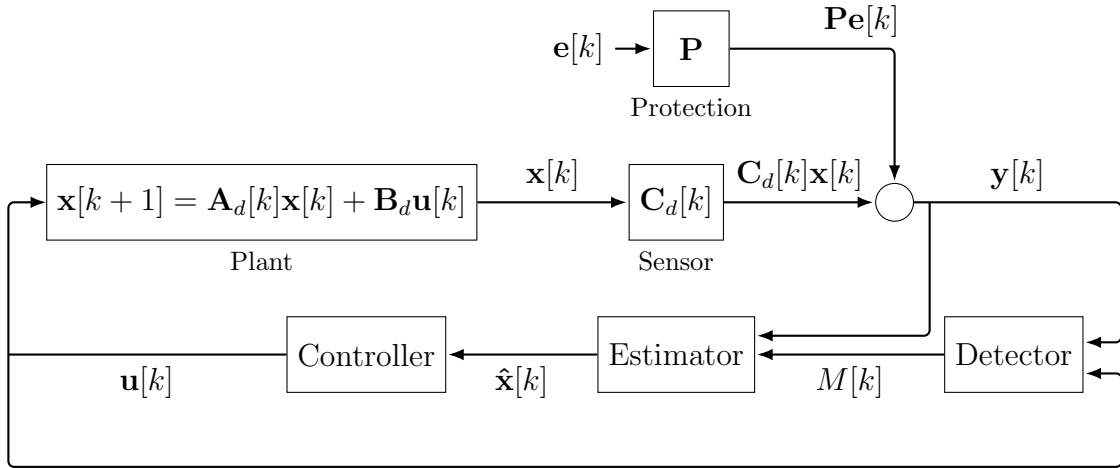


Figure 5.5: A control pipeline for the Jackal with fault detection and mitigation.

The fault mitigation algorithm is implemented as a patch to the `robot_localization`

package. The existing EKF in the package can already handle missing measurements through partial updates. The introduced patch adds functionality to revert (roll back) the EKF's state to an earlier saved instance, enabling selective replay of only valid measurements upon detecting a fault. An illustration of the rollback mechanism is shown in Figure 5.6. The patch is available on GitHub [66].

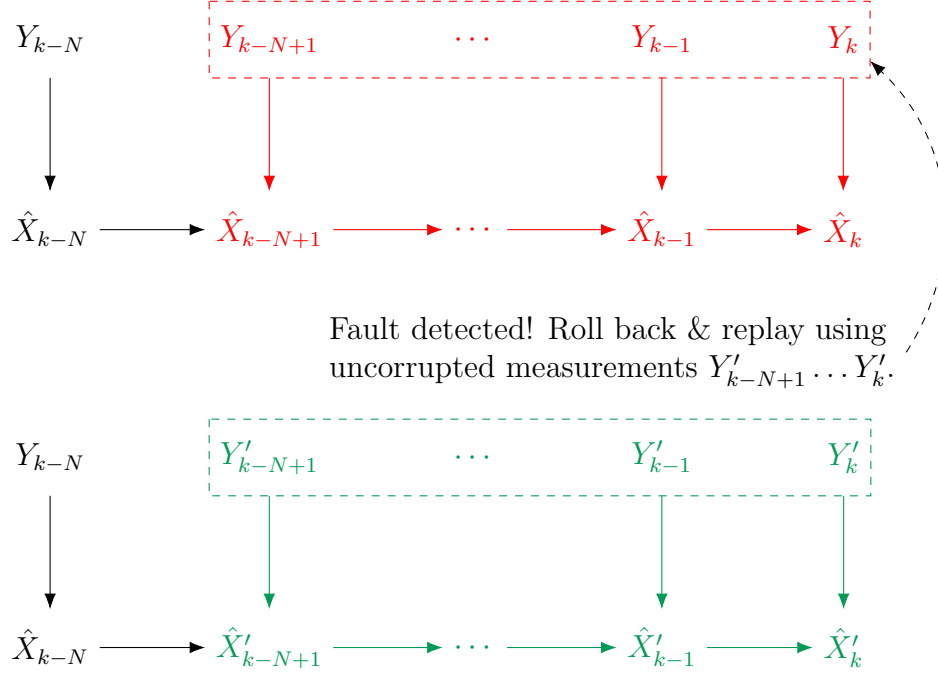


Figure 5.6: Illustration of the rollback mechanism. The red dashed rectangle indicates the measurement window in which a fault is detected. Within this window, the detector identifies the set of faulty sensors. These identified faults allow us to reconstruct modified measurements, $Y'_{k-N+1}, \dots, Y'_k$, obtained by excluding the faulty sensors from the original measurements. The estimator is then rewound to $\hat{X}_{k-N}$, the last estimate confirmed to be fault-free. From there, the modified measurements are replayed to produce new state estimates, $\hat{X}'_{k-N+1}, \dots, \hat{X}'_k$.

5.5 Fault Injection and Results

We inject a bias fault in the heading measurement of the Jackal as it follows the circular path. The fault is injected through software, by adding a constant 1 radian offset to the

heading measurement produced by the sensor. This offset is chosen to be similar to the fault in the IAC incident as analyzed in Section 4.6.

Figure 5.7 shows the results of the fault injection and detection. It compares the trajectory and heading of the vehicle under nominal conditions, with a fault injected, and with the fault detection and mitigation algorithm active. Without the fault detection and mitigation algorithm, the vehicle deviated significantly from the reference trajectory. With the fault detection and mitigation algorithm, the vehicle detected the fault around 0.3 seconds after the injected fault appeared in the first measurement. The vehicle then corrected the heading estimate and continued tracking the reference trajectory.

5.6 Discussions

In this chapter, we applied the optimal sensor protection and fault detection algorithms developed in Chapter 3 to a physical robot, Jackal. We also developed a simple fault mitigation strategy on top of the existing EKF implementation in `robot_localization` to allow the vehicle to continue tracking the reference trajectory even when a fault is detected.

The implementation of the fault algorithm in this chapter represents an earlier version, developed prior to the improvements made in Chapter 4. Consequently, there are several differences in implementation between the two chapters. First, this study uses a first-order Euler method to discretize the vehicle model, whereas Chapter 4 employs a more accurate step-invariant transformation. Second, the linearization strategy differs: this study uses a “walking backward” approach, while Chapter 4 uses the “walking forward” approach. The latter was found to be more robust during further development, as it relies less on the most recent data, which is more prone to undetected faults. Additionally, a software bug in the linearization routine in this chapter may have adversely affected detection performance, though its precise impact was not quantified. Finally, the optimization algorithm in Chapter 4 runs significantly faster due to an improved software architecture that leverages optimization library features to move part of the computation to initialization time.

Two additional factors may have affected this study. First, the communication setup introduced timing irregularities. Initial experiments used standard Wi-Fi within the building, but heavy interference led to unreliable data transmission. To address this, we switched to a Wireless Wire system, which uses directional antenna. However, this hardware was likely not intended for mounting on a mobile robot, and its use introduced periodic delays in the data stream—likely due to misalignment during motion—resulting in inconsistent sensor timing, as can be seen in the larger gaps between measurements in Figure 5.7.

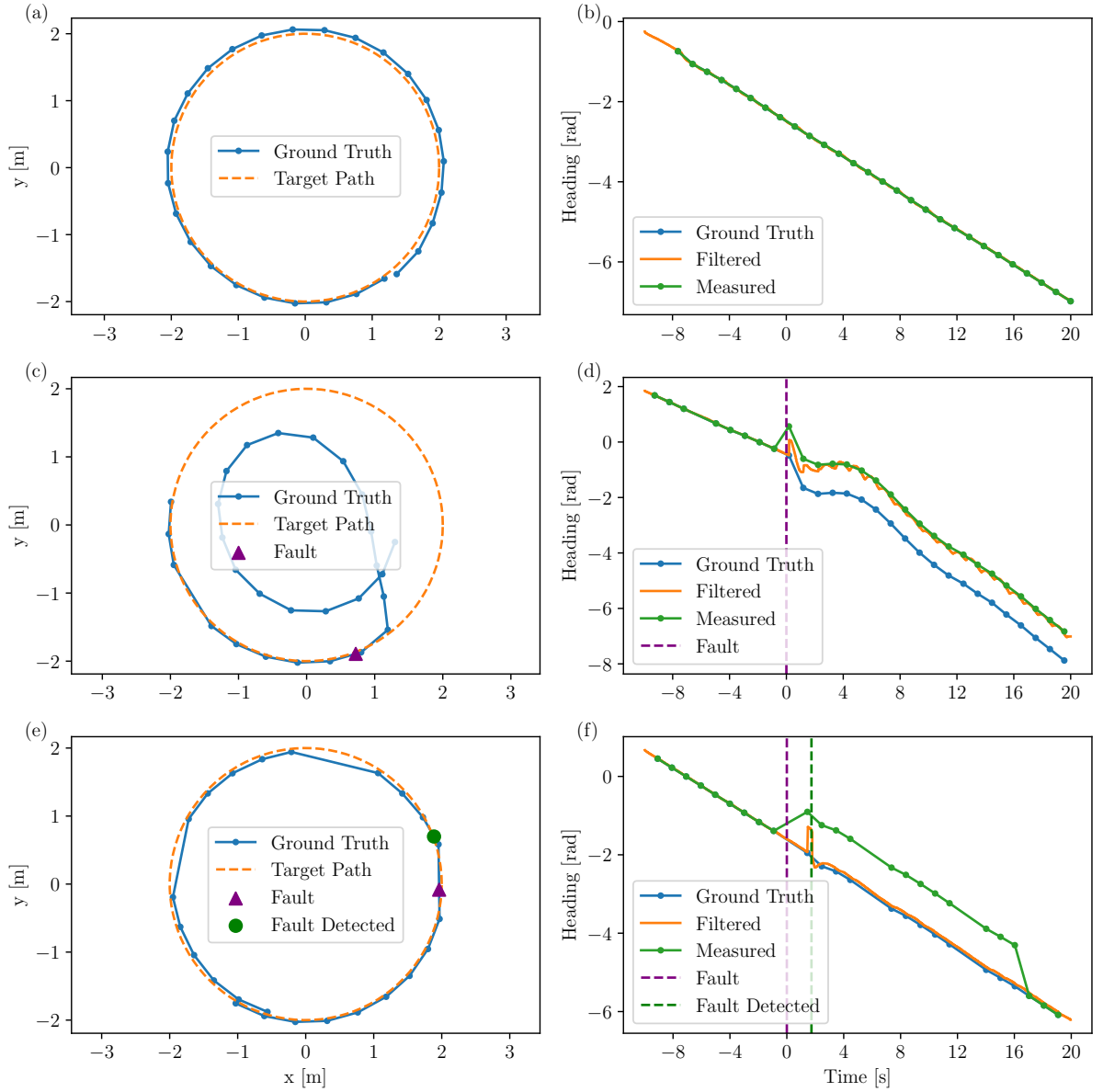


Figure 5.7: Comparison of trajectory and heading under nominal and fault conditions. (a–b) System under nominal conditions, without any fault injected. (c–d) System during a sensor fault: fault detection and mitigation module is inactive. (e–f) System during a sensor fault: fault detection and mitigation module is active. Fault injection times are marked in purple, and detection times are shown in green.

Second, the fault magnitude presented in this study—1 radian—is relatively large. While this is consistent with the type of fault observed in the IAC incident, it was close to the smallest magnitude that the earlier version of the detection algorithm could reliably detect. We conjecture that the improved linearization strategy in Chapter 4, and more generally, a better tuned motion controller, would result in trajectories more consistent with the linearized model, thereby improving model accuracy and enhancing fault detection performance. The exact relationship between model mismatch and detection accuracy remains a topic for future research.

All implementation details are available in the GitHub repository [\[67\]](#).

Chapter 6

Conclusions

Motivated by the need for robust fault detection and mitigation in autonomous systems, this thesis developed a mathematical framework for analyzing sensor importance based on their impact on a system’s ability to tolerate sensor faults. We further extended a secure estimation algorithm—originally designed for linear time-invariant systems—to operate on linearized models of nonlinear systems, enabling fault detection in a broader class of systems. The proposed algorithm was validated across three main scenarios: a simulation study, a case study using data from an IAC racing incident, and a real-world deployment on a physical robot. In the final scenario, we additionally demonstrated a simple, nearly plug-and-play fault mitigation strategy that integrates seamlessly with an existing control pipeline.

6.1 Key Contributions

6.1.1 Sparse Observability with Protection

We began with the concept of s -sparse observability (Definition 1), which provides a worst-case assessment of how sensor faults affect system observability. In Section 3.2, we introduced the notion of sensor protection to relax this framework by modeling a subset of sensors as immune to faults—a property that can be realized through redundancy or fault-hardened design. This binary model of protection allows for a structured way to improve fault tolerance. We showed that a foundational result in secure estimation, Theorem 1, remains valid under this protection model. Furthermore, we extended this framework to time-varying systems in Section 3.7, which is essential for most real-world applications.

6.1.2 Optimal Sensor Protection

Building on the protection model, we addressed the problem of selecting the most critical sensors to protect. In Section 3.4, we formulated this as an optimization problem aimed at maximizing system resilience within a fixed protection budget. Although the problem is combinatorial, it is solved offline, and we found that for systems with a modest number of sensors or limited budget constraints, the approach is easily tractable.

6.1.3 Fault Detection in Nonlinear Systems

Using our extension of the theorem that connects sparse observability with fault tolerance (Theorem 2), we generalized a secure estimation algorithm from linear time-invariant to linear time-varying systems. This generalization enabled us to apply the method to linearized models of nonlinear systems. In Chapter 4, we validated this algorithm in simulation and through a case study based on an IAC racing incident. In Chapter 5, we demonstrated a real-world deployment of the algorithm on a physical robot, alongside a simple mitigation strategy that enabled continued safe operation in the presence of sensor faults.

6.2 Summary of Findings

This research has produced several insights grounded in theoretical analysis, simulation studies, and real-world deployment. Building on existing results, we used the sparse observability index as a foundation for analyzing the importance of sensors in a system. We further showed that the optimal sensor protection problem, though combinatorial in nature, is tractable for systems with a relatively small number of sensors or limited protection budgets. Additionally, we found that extending fault detection algorithms to time-varying systems introduces the need for additional considerations—particularly in the choice of linearization strategy and the handling of model inaccuracies—which can significantly influence performance.

In simulation, we evaluated the algorithm’s sensitivity to a range of fault types—bias, drift, noise, and spikes—and observed consistent detection behavior. Specifically, for each fault type, there is a minimum fault magnitude above which the algorithm reliably detects faults. This suggests that the secure estimation approach behaves similarly to a thresholding mechanism, a useful insight for tuning detection parameters.

The deployment of the algorithm on a physical robot offered practical validation of its effectiveness. The fault detection algorithm operated in real-time and correctly identified sensor faults, while the mitigation strategy enabled the system to maintain operational stability during fault conditions. Integration into the existing control pipeline was achieved with minimal changes, underscoring the practicality of the proposed approach. However, the deployment also highlighted the importance of communication infrastructure, which play a critical role in real-world performance.

The case study involving the IAC racing incident further illustrated the practical implications of fault detection. Retrospective analysis indicated that the algorithm could have detected the relevant sensor fault approximately 4 seconds before the crash, potentially preventing the incident. This finding supported our theoretical analysis on the importance of position sensors and reinforced the value of robust fault detection in high-performance autonomous systems.

6.3 Future Directions

While this work takes initial steps toward improving the robustness of autonomous vehicles to sensor faults, several promising directions remain for future research. From a theoretical standpoint, the s -sparse observability with protection framework is a simple and effective way to characterize sensor importance, but it is binary in 2 key respects. First, the protection model is binary, considering only whether a sensor is fully protected or unprotected. A probabilistic model of protection, where sensor resilience is characterized by a distribution or likelihood of failure, may provide a more nuanced understanding of sensor importance. Second, s -sparse observability itself is a binary notion—it determines whether a system is observable under a certain number of missing sensors, but does not quantify the degree of observability. Recent work on eigenvalue observability [39] may offer additional insights into sensor importance beyond binary metrics.

As we have seen in simulation (Chapter 4), given a choice of noise threshold setting in the optimization problem, there is a threshold for the magnitude of each fault type, above which the detection performance is almost perfect. Further investigation into the interplay between noise thresholds and fault detectability could help optimize algorithm performance. Some work has been done in [48] to explicitly account for noise in the optimization problem, and this could be a fruitful area of research.

Extending our analyses to vehicle models in local frames, such as curvilinear coordinate models, would make our approach applicable to a wider range of autonomous vehicles, particularly those operating in environments where global positioning is unreliable or

unavailable. The structural and observability properties of such models may differ significantly from global-frame models, and this could provide new insights into the importance of sensors in a system.

Another promising direction for future work involves handling multi-rate sensor systems within the deployment algorithm. In Chapter 5, we briefly discussed how varying sensor update rates influence the choice of measurement window size. To enable deployment, we relied on extensive numerical testing to justify a “uniform observability” assumption, allowing us to apply the algorithms developed in earlier chapters. However, a more rigorous treatment of multi-rate systems—potentially through dedicated observability analyses or modified estimation techniques similar to [37]—could enhance the robustness and applicability of the overall approach. This would be especially relevant in real-world scenarios where heterogeneous sensor update rates arise due to hardware limitations or communication constraints.

Finally, broader real-world deployments would provide a deeper understanding of how these algorithms perform in diverse operational contexts. As shown in Chapter 4, the algorithm can run in real-time on small systems using a single CPU core, even without the l_1 relaxation. We also showed in Chapter 5 that the algorithm can be integrated into an existing control pipeline with minimal effort. These findings suggest that the method is computationally feasible and practically deployable for low-dimensional systems. However, scaling to higher-dimensional systems or more complex platforms will require further evaluation. Larger-scale deployments could also surface additional implementation challenges—such as communication overhead or timing constraints—which could be addressed through further algorithmic refinements or infrastructure improvements.

6.4 Final Remarks

In summary, this thesis has laid a foundation for analyzing and improving fault resilience in autonomous systems through a combination of theoretical contributions, algorithmic development, and empirical validation. By extending sparse observability theory and applying fault detection algorithms to practical scenarios, this work offers a step toward safer and more reliable autonomous operation. While the results are promising, they represent only an initial step into a broad and complex design space. Future work in more nuanced fault models, noise-aware algorithms, and real-world integration will be essential for advancing the safety and robustness of next-generation autonomous systems.

References

- [1] Markus Achtelik. vicon_bridge: A ROS driver for VICON motion capture systems. https://github.com/ethz-asl/vicon_bridge, 2025. Accessed: May 16, 2025. 76
- [2] Safoan Alhalali, Christopher Nielsen, and Ramadan El-Shatshat. Mitigation of cyber-physical attacks in multi-area automatic generation control. *International Journal of Electrical Power & Energy Systems*, 112:362–369, 2019. 11, 12
- [3] Liwei An and Guang-Hong Yang. Secure state estimation against sparse sensor attacks with adaptive switching mechanism. *IEEE Transactions on Automatic Control*, 63(8):2596–2603, 2018. 8
- [4] Liwei An and Guang-Hong Yang. State estimation under sparse sensor attacks: A constrained set partitioning approach. *IEEE Transactions on Automatic Control*, 64(9):3861–3868, 2018. 8
- [5] Liwei An and Guang-Hong Yang. Distributed secure state estimation for cyber-physical systems under sensor attacks. *Automatica*, 107:526–538, 2019. 9
- [6] Richard Vernon Beard. *Failure accommodation in linear systems through self-reorganization*. PhD thesis, Massachusetts Institute of Technology, 1971. 7
- [7] Mogens Blanke, Michel Kinnaert, Jan Lunze, and Marcel Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer Science & Business Media, 2006. 7
- [8] National Transportation Safety Board. Briefing points on mid-air collision near DCA, February 2025. Accessed: May 16, 2025. 1
- [9] Joseph S Brinker and Kevin A Wise. Flight testing of reconfigurable control law on the X-36 tailless aircraft. *Journal of Guidance, Control, and Dynamics*, 24(5):903–909, 2001. 11, 12

- [10] Car&Bike Team. Top speeds of F1, IndyCar, MotoGP and more. <https://www.carandbike.com/news/how-fast-2727217>, January 2022. Accessed: May 16, 2025. 41
- [11] Jie Chen and Ron J. Patton. *Robust model-based fault diagnosis for dynamic systems*. Springer New York, NY, 1999. 7
- [12] Michelle S. Chong, Henrik Sandberg, and Joao P. Hespanha. A secure state estimation algorithm for nonlinear systems under sensor attacks. In *IEEE Conference on Decision and Control (CDC)*, pages 5743–5748, 2020. 9
- [13] Michelle S. Chong, Masashi Wakaiki, and João P. Hespanha. Observability of linear systems under adversarial attacks. In *American Control Conference (ACC)*, pages 2439–2444, 2015. 7, 9, 10, 15, 16, 19
- [14] Edward Y. Chow and Alan S. Willsky. Analytical redundancy and the design of robust failure detection systems. *IEEE Transactions on automatic control*, 29(7):603–614, 1984. 7
- [15] Clearpath Robotics. Jackal user manual. https://docs.clearpathrobotics.com/docs/robots/outdoor_robots/jackal/user_manual_jackal/, February 2023. Accessed: May 16, 2025. xi, 76
- [16] James C. Deckert, Mukund N. Desai, John J. Deyst, and Alan S. Willsky. F-8 DFBW sensor failure identification using analytic redundancy. *IEEE Transactions on Automatic Control*, 22(5):795–803, 1977. 10, 12
- [17] Sidney Dekker. B-737 MAX and the crash of the regulatory system. *Safety Science*, 120:862–873, 2019. 1
- [18] Mukund N. Desai, James C. Deckert, John J. Deyst, Alan S. Willsky, and Edward Y. Chow. Dual redundant sensor FDI techniques applied to the NASA F8C DFBW aircraft. In *AIAA Guidance and Control Conference*, page 1976, August 1976. 10, 12
- [19] European Space Agency (ESA). GeoEye-1 mission overview. <https://earth.esa.int/eogateway/missions/geoeye-1>, 2025. Accessed: May 16, 2025. xi, 67
- [20] Hamza Fawzi, Paulo Tabuada, and Suhas Diggavi. Secure state-estimation for dynamical systems under active adversaries. In *Allerton Conference on Communication, Control, and Computing*, pages 337–344, 2011. 7, 8, 10, 11, 12, 64

- [21] Hamza Fawzi, Paulo Tabuada, and Suhas Diggavi. Secure estimation and control for cyber-physical systems under adversarial attacks. *IEEE Transactions on Automatic Control*, 59(6):1454–1467, June 2014. 7, 8, 10, 15, 16, 29, 34, 36
- [22] Thomas Fermi. Algorithms for automated driving: Pure pursuit. <https://thomasfermi.github.io/Algorithms-for-Automated-Driving/Control/PurePursuit.html>, 2022. Accessed: May 16, 2025. x, 49, 50, 51
- [23] Venkatesan Guruswami, James R Lee, and Avi Wigderson. Euclidean sections of with sublinear randomness and error-correction over the reals. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 444–454. Springer, 2008. 36
- [24] Xingkang He, Ehsan Hashemi, and Karl H Johansson. Distributed control under compromised measurements: Resilient estimation, attack detection, and vehicle platooning. *Automatica*, 134:109953, 2021. 11, 12
- [25] Mashor Housh and Ziv Ohar. Model-based approach for cyber-physical attack detection in water distribution systems. *Water research*, 139:132–143, 2018. 10
- [26] Indy Autonomous Challenge. Live stream - Indy Autonomous Challenge. <https://www.indyautonomuschallenge.com/stream-oct-23>. Backup link: https://youtu.be/gHvUf_J4rkw. Accessed: May 16, 2025. x, 2
- [27] IndyCar.com. What is IndyCar? <https://www.indycar.com/2024/Future-Dev-IC101/Where-Does-INDYCAR-Race/Schedule-Map>, n.d. Accessed: May 16, 2025. 41
- [28] Rolf Isermann, Ralf Schwarz, and Stefan Stolz. Fault-tolerant drive-by-wire systems. *IEEE Control Systems Magazine*, 22(5):64–81, 2002. 11, 12
- [29] Simon J. Julier and Jeffrey K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. 48
- [30] Junsoo Kim, Chanhwa Lee, Hyungbo Shim, Yongsoon Eun, and Jin H Seo. Detection of sensor attack and resilient state estimation for uniformly observable nonlinear systems having redundant sensors. *IEEE Transactions on Automatic Control*, 64(3):1162–1169, 2018. 9
- [31] Junsoo Kim, Jin Gyu Lee, Chanhwa Lee, Hyungbo Shim, and Jin H. Seo. Local identification of sensor attack and distributed resilient state estimation for linear systems. In *IEEE Conference on Decision and Control (CDC)*, pages 2056–2061, 2018. 9

- [32] Junsoo Kim, Jin Gyu Lee, Henrik Sandberg, and Karl H. Johansson. Complexity reduction for resilient state estimation of uniformly observable nonlinear systems. *IEEE Transactions on Automatic Control*, 70(2):1267–1272, 2025. 9
- [33] Andreas Krause and Daniel Golovin. Submodular function maximization. In Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli, editors, *Tractability*, pages 71–104. Cambridge University Press, 1 edition, February 2014. 25
- [34] Roger Labbe. FilterPy (0.0.15). <https://github.com/rlabbe/filterpy>, 2015. Accessed: May 16, 2025. 48
- [35] Chanhwa Lee, Hyungbo Shim, and Yongsoon Eun. On redundant observability: From security index to attack detection and resilient state estimation. *IEEE Transactions on Automatic Control*, 64(2):775–782, 2019. 8
- [36] Jinghao Li and Guang-Hong Yang. Disturbance decoupled secure state estimation: An orthogonal projection-based method. *Automatica*, 147:110740, 2023. 12
- [37] Zishuo Li, Anh Tung Nguyen, André MH Teixeira, Yilin Mo, and Karl H Johansson. Secure state estimation with asynchronous measurements against malicious measurement-data and time-stamp manipulation. In *IEEE Conference on Decision and Control (CDC)*, pages 7073–7080, 2023. 9, 11, 91
- [38] Xusheng Luo, Miroslav Pajic, and Michael M. Zavlanos. An optimal graph-search method for secure state estimation. *Automatica*, 123:109323, 2021. 8
- [39] Yanwen Mao, Aritra Mitra, Shreyas Sundaram, and Paulo Tabuada. On the computational complexity of the secure state-reconstruction problem. *Automatica*, 136:110083, 2022. 8, 14, 17, 90
- [40] Hussein Mazeh, Majd Saied, Hassan Shraim, and Clovis Francis. Fault-tolerant control of an hexarotor unmanned aerial vehicle applying outdoor tests and experiments. *IFAC-PapersOnLine*, 51(22):312–317, 2018. 11, 12
- [41] Thomas Moore and Daniel Stouch. A generalized extended kalman filter implementation for the robot operating system. In *International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014. 82
- [42] Motorsport.com. Braking power: What’s the biggest stop of the Formula 1 season? <https://www.motorsport.com/f1/news/braking/3221934/>, December 2017. Accessed: May 16, 2025. 41

- [43] MotorTrend. Motor trend indy car - performance trend. <https://www.motortrend.com/reviews/motor-trend-indy-car/>, December 2003. Accessed: May 16, 2025. 47
- [44] Jeffrey L. Musgrave, Ten-Huei Guo, Edmond Wong, and Ahmet Duyar. Real-time accommodation of actuator faults on a reusable rocket engine. *IEEE Transactions on Control Systems Technology*, 5(1):100–109, 1997. 11, 12
- [45] Yorie Nakahira and Yilin Mo. Attack-resilient $\mathcal{H}_2$, $\mathcal{H}_\infty$, and ℓ_1 state estimator. *IEEE Transactions on Automatic Control*, 63(12):4353–4360, 2018. 8
- [46] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, December 1978. 25
- [47] Hassan Noura, Didier Theilliol, Jean-Christophe Ponsart, and Abbas Chamseddine. *Fault-tolerant control systems: Design and practical applications*. Springer Science & Business Media, 2009. 6, 7
- [48] Miroslav Pajic, Insup Lee, and George J. Pappas. Attack-resilient state estimation for noisy dynamical systems. *IEEE Transactions on Control of Network Systems*, 4(1):82–92, 2017. 8, 10, 11, 12, 16, 90
- [49] Miroslav Pajic, James Weimer, Nicola Bezzo, Oleg Sokolsky, George J Pappas, and Insup Lee. Design and implementation of attack-resilient cyberphysical systems: With a focus on attack-resilient state estimators. *IEEE Control Systems Magazine*, 37(2):66–81, 2017. 33
- [50] Fabio Pasqualetti, Florian Dörfler, and Francesco Bullo. Attack detection and identification in cyber-physical systems. *IEEE Transactions on Automatic Control*, 58(11):2715–2729, 2013. 11, 12
- [51] Fabio Pasqualetti, Florian Dorfler, and Francesco Bullo. Control-theoretic methods for cyberphysical security: Geometric principles for optimal cross-layer resilient control systems. *IEEE Control Systems Magazine*, 35(1):110–127, 2015. 11, 12
- [52] Fabio Pasqualetti, Florian Dörfler, and Francesco Bullo. A divide-and-conquer approach to distributed attack identification. In *IEEE Conference on Decision and Control (CDC)*, pages 5801–5807, 2015. 9
- [53] Ron J. Patton, Paul M. Frank, and Robert N. Clark. *Fault Diagnosis in Dynamic Systems: Theory and Application*. Prentice-Hall, Inc., 1989. 7

- [54] Ron J. Patton, Paul M. Frank, and Robert N. Clark. *Issues of Fault Diagnosis for Dynamic Systems*. Springer Science & Business Media, 2000. 7
- [55] Popular Mechanics. How to take turn one at the Indy 500. <https://www.popularmechanics.com/cars/a6352/>, May 2010. Accessed: May 16, 2025. 41
- [56] Andrew Pressley. *Elementary Differential Geometry*. Springer, New York, U.S.A., 2001. 46
- [57] Clearpath Robotics. jackal: Common packages for Jackal, including messages, robot description, and controllers. <https://github.com/jackal/jackal/tree/31fdf4efc366d4d7ad9e1ff4f308138bd9b5c6fe>, 2025. Accessed: May 16, 2025. 76
- [58] Takumi Shinohara and Toru Namerikawa. Optimal security investment problem for secure state estimation on cyber-physical systems. *IEEE Transactions on Automatic Control*, 70(2):1244–1251, 2025. 10, 16
- [59] Martin L. Shooman. *Reliability of computer systems and networks: fault tolerance, analysis, and design*. John Wiley & Sons, 2003. 6
- [60] Yasser Shoukry, Michelle Chong, Masashi Wakaiki, Pierluigi Nuzzo, Alberto Sangiovanni-Vincentelli, Sanjit A Seshia, Joao P Hespanha, and Paulo Tabuada. SMT-based observer design for cyber-physical systems under sensor attacks. *ACM Transactions on Cyber-Physical Systems*, 2(1):1–27, 2018. 8, 11, 12
- [61] Yasser Shoukry and Paulo Tabuada. Event-triggered state observers for sparse sensor noise/attacks. *IEEE Transactions on Automatic Control*, 61(8):2079–2091, 2016. 7, 11, 12, 14, 15, 16
- [62] Xiulan Song, Xiaoxin Lou, Junwei Zhu, and Defeng He. Secure state estimation for motion monitoring of intelligent connected vehicle systems. *Sensors*, 20:1253, February 2020. 11, 12
- [63] James F. Stewart and Thomas L. Shuck. Flight-testing of the self-repairing flight control system using the F-15 highly integrated digital electronic control flight research facility. In *AIAA/SFTE/DGLR/SETP 5th Biannual Flight Test Conference*, 1990. 11, 12
- [64] Riccardo Taormina, Stefano Galelli, Nils Ole Tippenhauer, Elad Salomons, Avi Ostfeld, Demetrios G. Eliades, Mohsen Aghashahi, Raanju Sundararajan, Mohsen Pourahmadi, M. Katherine Banks, B. M. Brentan, M. Herrera, Amin Rasekh, Enrique

- Campbell, I. Montalvo, G. Lima, J. Izquierdo, Kelsey Haddad, Nikolaos Gatsis, Ahmad Taha, Saravanakumar Lakshmanan Somasundaram, D. Ayala-Cabrera, Sarin E. Chandy, Bruce Campbell, Pratim Biswas, Cynthia S. Lo, D. Manzi, E. Luvizotto, Jr, Zachary A. Barker, Marcio Giacomoni, M. Fayzul K. Pasha, M. Ehsan Shafiee, Ahmed A. Abokifa, Mashor Housh, Bijay Kc, and Ziv Ohar. The battle of the attack detection algorithms: Disclosing cyber attacks on water distribution networks. *Journal of Water Resources Planning and Management*, 144(8):04018048, August 2018. 9
- [65] Alan S. Willsky. A survey of design methods for failure detection in dynamic systems. *Automatica*, 12(6):601–611, 1976. 7
- [66] Chuan Tian Zhang. Add service to clear topic data (commit f8029cd) in *robot_localization*. https://github.com/ben-z/robot_localization/commit/f8029cd322a2f19f15b734b0ad6b186f5af2c2fa, 2023. Accessed: May 16, 2025. 84
- [67] Chuan Tian Zhang. Measurement corruption detection for the Jackal robot. <https://github.com/ben-z/measurement-corruption-detection-jackal>, 2025. Accessed: May 16, 2025. 87
- [68] Chuan Tian Zhang. Measurement corruption detection simulations. <https://github.com/ben-z/measurement-corruption-detection-sim>, 2025. Accessed: May 16, 2025. 56
- [69] Chuan Tian Zhang, Derek Rayside, and Christopher Nielsen. Sensor protection and fault detection: Application to autonomous vehicles. In *IEEE Conference on Control Technology and Applications (CCTA)*, 2025. To appear. 4
- [70] Youmin Zhang and Jin Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2):229–252, 2008. 11
- [71] Kemin Zhou and John Comstock Doyle. *Essentials of robust control*, volume 104. Prentice hall Upper Saddle River, NJ, 1998. 7