5-14-2023

Real-time Adaptive Detection and Recovery against Sensor Attacks in Cyber-physical Systems

Lin Zhang
Syracuse University

ABSTRACT

Cyber-physical systems (CPSs) utilize computation to control physical objects in real-world environments, and an increasing number of CPS-based applications have been designed for life-critical purposes. Sensor attacks, which manipulate sensor readings to deceive CPSs into performing dangerous actions, can result in severe consequences. This urgent need has motivated significant research into reactive defense. In this dissertation, we present an adaptive detection method capable of identifying sensor attacks before the system reaches unsafe states. Once the attacks are detected, a recovery approach that we propose can guide the physical plant to a desired safe state before a safety deadline.

Existing detection approaches tend to minimize detection delay and false alarms simultaneously, despite a clear trade-off between these two metrics. We argue that attack detection should dynamically balance these metrics according to the physical system's current state. In line with this argument, we propose an adaptive sensor attack detection system comprising three components: an adaptive detector, a detection deadline estimator, and a data logger. This system can adapt the detection delay and thus false alarms in real-time to meet a varying detection deadline, thereby improving usability. We implement our detection system and validate it using multiple CPS simulators and a reduced-scale autonomous vehicle testbed.

After identifying sensor attacks, it is essential to extend the benefits of attack detection. In this dissertation, we investigate how to eliminate the impact of these attacks and propose novel real-time recovery methods for securing CPSs. Initially, we target sensor attack recovery in linear CPSs. By employing formal methods, we are able to reconstruct state estimates and calculate a conservative safety deadline. With these constraints, we formulate the recovery problem as either a linear programming or a quadratic program-

ming problem. By solving this problem, we obtain a recovery control sequence that can smoothly steer a physical system back to a target state set before a safe deadline and maintain the system state within the set once reached. Subsequently, to make recovery practical for complex CPSs, we adapt our recovery method for nonlinear systems and explore the use of uncorrupted sensors to alleviate uncertainty accumulation. Ultimately, we implement our approach and showcase its effectiveness and efficiency through an extensive set of experiments. For linear CPSs, we evaluate the approach using 5 CPS simulators and 3 types of sensor attacks. For nonlinear CPSs, we assess our method on 3 nonlinear benchmarks.

REAL-TIME ADAPTIVE DETECTION AND RECOVERY AGAINST SENSOR ATTACKS IN CYBER-PHYSICAL SYSTEMS

by

Lin Zhang

B.E., Dalian University of Technology, 2015
M.S., Syracuse University, 2022

Dissertation
Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer and Information Science and Engineering.

Syracuse University
May 2023

ACKNOWLEDGEMENT

First, I would like to express my deepest gratitude to my advisor, Prof. Fanxin Kong, for his unwavering support, constructive feedback, and invaluable insights throughout my research journey. He has guided me in developing new ideas, honing my academic tastes, and cultivating the skills required to conduct high-quality research. I am deeply appreciative of the countless times he worked overnight to revise my papers and for providing me with opportunities to collaborate with top researchers in my field. I consider myself truly fortunate to have had the opportunity to learn from his expertise and wisdom and to have grown under his supervision.

I would like to extend my heartfelt thanks to my dissertation committee members, Prof. Amit K. Sanyal, Prof. Qinru Qiu, and Prof. M. Cenk Gursoy, for dedicating their time to my dissertation, offering valuable comments, constructive suggestions, and engaging in thoughtful discussions that have significantly enhanced my research.

I am grateful to my research collaborators and lab members, whose contributions have been invaluable to my research and personal development. Prof. Insup Lee and Oleg Sokolsky provided crucial suggestions and insights for several research papers, while Prof. Xin Chen offered extensive support in formal methods. Prof. Álvaro Cárdenas generously shared his valuable experience within the security community. I also appreciate the assistance of Pengyuan Lu, Kaustubh Sridhar, and Luis Burbano in writing and conducting experiments for several research papers. My lab members, Mengyu Liu, Zifan Wang, Shixiong Jiang, and Weizhe Xu, have also contributed significantly to my growth.

I would like to acknowledge the financial support from Prof. Kong's startup funding, Syracuse University fellowship, and the National Science Foundation, which have provided me

with the necessary financial resources and allowed me to focus on my research and present my work at various conferences.

Special thanks to the department chair, Dr. Jae C. Oh, and the administrative and technical personnel of the Department of Electrical Engineering and Computer Science, especially Mrs. Cynthia Salanger, Mrs. Rebecca Noble, and Mrs. Cynthia Bromka-Skafidas, for their assistance in addressing technical, procedural, and logistical matters.

I would like to express my deepest appreciation to my family, especially my parents, who have respected my decision to study abroad and provided immense emotional support. Their selfless love, constant encouragement, and unwavering support have been the driving forces behind my success in my PhD journey.

This dissertation stands as a testament to the support, guidance, and encouragement I have received from all these remarkable individuals in my life. My sincerest thanks to each and every one of you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER 1

Introduction

Cyber-physical systems (CPSs) integrate computational resources and physical processes to form a cohesive whole with sensing and actuation components. These systems have become increasingly prevalent in various domains, including transportation, energy, healthcare, supply chain, industrial manufacturing, and agriculture. As a result, they now underpin critical infrastructure and pervade everyday life. For instance, tall buildings employ structural vibration control systems to counteract wind-induced vibrations [1], while Amazon is on the verge of using drones for package delivery within one hour [2]. Many CPS applications are safety-critical, and their failure can lead to significant consequences, ranging from economic losses and societal disruption to personal injury.

The safety-critical nature of CPSs necessitates a thorough understanding of their security vulnerabilities. The tight integration of computational and physical components exposes CPSs to various types of attacks. One significant security risk involves sensor attacks, which manipulate sensor data to affect the physical system adversely. When a controller receives malicious sensor data, it generates corrupted state estimates that may push the system into unsafe physical states. Sensor attacks can originate from cyber attack surfaces, such as compromised control software or communication networks between sensors and the controller [3], [4]. However, an emerging threat known as transduction attacks poses a distinct challenge. These attacks non-invasively manipulate sensor readings by altering physical properties, allowing the injection of malicious signals [5], [6]. For example, attackers can generate ultrasonic waves to corrupt IMU sensor data [7], spoof GPS signals to misdirect autonomous vehicles [8], or remotely tamper with LiDAR sensors to make vehicles perceive nonexistent objects [9]. Since sensor attacks directly impact physical states, rely-

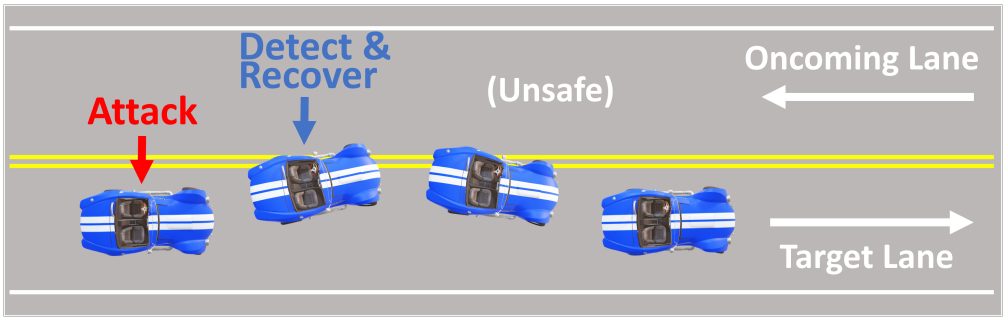


Figure 1: Attack Detection and Recovery Demonstration on a Vehicle

ing solely on cybersecurity measures is insufficient to protect CPSs against these threats [4], [5], [10]. Moreover, as CPSs become more autonomous, the effectiveness of such attacks is likely to increase [11]–[13].

The pressing need to counter sensor attacks has led to the development of both proactive and reactive defense strategies [14]. Proactive defenses involve security-hardening designs implemented within the CPS prior to any attack. One such example is sensor fusion, which combines data from multiple sensors to reduce the influence of uncertain readings and provide security guarantees [8]. However, when proactive defenses are breached, reactive defenses become essential to protect the CPS. Reactive defenses are activated upon detecting an attack, functioning online to minimize its impact [15]. This paper concentrates on reactive defense mechanisms, particularly sensor attack detection and recovery strategies. As an example, Figure 1 demonstrates a detection and recovery process for an autonomous vehicle subjected to a GPS spoofing attack. The attack causes the vehicle to deviate from the center of its lane and even veer into the oncoming lane. Reactive defenses must identify the attack before the car enters the oncoming lane and steer the vehicle back to its own lane, as depicted by the green recovery trajectory. This highlights the importance of timely and effective reactive defense mechanisms in ensuring the safety of CPSs and their applications.

1.1. Real-time Adaptive Detection

Sensor attack detection aims to identify attacks by discerning differences between observed sensor measurements and predicted values [5], [16]–[18]. Existing works attempt to minimize detection delay while maximizing usability. Detection delay refers to the time between an attack's onset and its detection, while usability corresponds to the false alarm rate; a lower rate indicates better usability. Achieving both goals simultaneously is challenging due to the inherent trade-off between these two metrics, as greater usability often results in longer detection delays [19], [20]. We argue that attack detection should prioritize different metrics depending on the CPS's current state. For instance, when a physical system's state is near the unsafe region, lowering the detection delay takes precedence over reducing false alarms, and vice versa.

Implementing this adaptability in attack detection presents several challenges. First, in safety-critical CPSs, timing is crucial. Detection after consequences occur is equally damaging, as illustrated by the futility of detecting an attack post-car accident. This time constraint, known as the detection deadline, must be met for effective detection. Second, online calculating a detection deadline is not trivial, which varies as the physical system state evolves. The overhead must be low, or the calculated deadline may become outdated. Additionally, a detector with fixed or unpredictable detection delay is incompatible with the varying deadline. Third, although the detection delay should not exceed the deadline, a shorter detection delay is not always advantageous. For example, a detector that raises an alert every control period can discover attacks immediately, offering the shortest detection delay. However, this results in an unmanageable number of false alarms and therefore unacceptable usability. Conversely, a detector that waits for multiple control periods before raising an alert increases the detection delay.

To address these challenges, we propose a real-time adaptive attack detection system capable of dynamically adjusting detection delay and false alarms based on the system's varying state. Our detection system elaborates in Chapter 3 in detail. It exhibits shorter detection delays and more false alarms when the detection deadline is stringent, and vice versa. The system comprises three major components (as shown in Fig. 5), with the technical contributions for each component as follows:

• *Detection Deadline Estimator:* We develop a reachability-based technique for conservatively estimating the detection deadline in real-time. This method assesses system vulnerability by computing the reachable set of future potential system behaviors.

• *Adaptive Detector:* We create a window-based detection algorithm that dynamically adapts its detection delay according to the deadline, ensuring no data points are missed during the adaptation process.

• *Data Logger:* We design a sliding-window based data logging protocol, maintaining sufficient trustworthy data for deadline estimation and attack detection even when the detection delay varies

We implement our detection system and evaluate its performance using multiple CPS simulators and a reduced-scale autonomous vehicle testbed. The results demonstrate the efficiency and effectiveness of our proposed detection system.

1.2. Real-time Attack Recovery

Despite the extensive volume of research on attack detection, a critical question remains largely unaddressed: what should be done after detecting an attack? A recent survey summarizing 32 CPS security papers also raises this question, emphasizing the need for future work on attack response [21]. It is crucial to capitalize on the benefits of attack detection, halt the ongoing deviation, and eliminate the negative impacts caused by the attack on

the system [20], [22]. Several other surveys on CPS attack detection, such as [20], [23], also highlight the lack of research on attack response strategies. Furthermore, detection delays exist before attacks are identified, during which the physical system may have already significantly deviated from the desired state. The deviations resulting from sensor attacks pose severe consequences for CPSs. Therefore, attack recovery approaches are needed to mitigate these negative impacts and correct the deviations in physical states [20], [22].

Existing approaches are insufficient to address the real-time recovery problem. First, a common method for responding to detected attacks involves isolating compromised sensors, deriving state estimates from virtual sensors, and continuing to use the original controller to manage the system [4], [24]–[28]. However, this method faces two significant issues. One is that the original controller implements a mild policy, which may not be fast enough to restore the system before the safety deadline. For instance, a car's physical state must be recovered before colliding with an obstacle, and a UAV must be stabilized before crashing into the ground. The other issue is that the original controller may not be sufficiently robust to avoid unsafe states during recovery. Second, in the control domain, robust control techniques primarily focus on tolerating bounded disturbances or errors [29]. As such, they are ill-equipped to handle sensor attacks, where sensor readings may be arbitrarily manipulated by attackers [22], [30]. Moreover, control methods with static policies are unsuitable for the real-time recovery problem due to the need to accommodate varying safety deadlines.

1.2.1. Real-time Sensor-attack Recovery in Linear CPSs

To address the limitations of previous studies, we propose several real-time recovery methods against sensor attacks in linear CPSs. Our recovery system can safely and smoothly recover a CPS before the recovery deadline and maintain the system in a target state set once it is reached. More detailed designs can be found in Chapter 4. The attack recovery

system consists of:

• *Recovery Controller:* The core component of the recovery controller is a control calculator based on optimization problems with safety and timing constraints. The optimization can be a linear-programming problem (LP) or a quadratic-programing problem based on a Linear-Quadratic Regulator (LQR). The time horizon of optimization is set as a safety deadline plus a conservatively estimated time that the system is maintained in the target set (named maintainable time) when under attack. Once the optimization problem is solved, we obtain the recovery control sequence.

• *Supporting Components:* The recovery controller also includes a checkpointer, a state reconstructor, and a deadline estimator. First, we propose a sliding-window-based checkpointing protocol that considers a varying detection delay, removes false data, and retains sufficient trustworthy data for attack recovery computations. Second, as the sensor information is no longer trustworthy during an attack, we present a state reconstruction approach that considers computational overhead and uses the checkpointed data to estimate the system state when the recovery sequence begins to be applied. Third, to determine an appropriate length for the recovery control sequence, we develop a reachability-based approach to calculate a safety deadline and an approach to conservatively estimate the maintainable time.

• We implement our framework and evaluate its performance using five CPS simulators under three sensor attack scenarios. The results demonstrate the efficiency and efficacy of our design and techniques.

1.2.2. Real-time Sensor-attack Recovery in Complex CPSs

Existing attack recovery methods for cyber-physical systems (CPSs) face significant challenges when applied to real-world scenarios, since real CPSs are often complex and nonlin-

ear. Nonlinear differential equations often lack analytical solutions, so exact reachability computation for nonlinear systems can be both challenging and computationally intensive [31]. Although some methods simplify the problem by working on linear models obtained through system linearization or identification, the linear approximation is only accurate within a small range around the equilibrium point. Outside this range, large modeling errors occur, rendering these methods incapable of finding an effective recovery solution. Moreover, uncertainties such as noise and external disturbances accumulate over time due to the lack of feedback from physical sensors after an attack is detected. Existing methods assume all sensors are compromised, ceasing to receive any feedback, which prevents the rejection of accumulating uncertainties and hinders the effectiveness of recovery. To address these challenges, we propose a novel real-time recovery method against sensor attacks specifically tailored for nonlinear CPSs, shown in Chapter 5. Our contributions can be summarized as follows:

• For nonlinear CPSs, we propose an attack recovery system consisting of four components. The state predictor conducts nonlinear reachability analyses, enabling the reconstruction of the initial state set for recovery. Subsequently, the time oracle calculates an online safety deadline, beyond which the system states may become unsafe under current control inputs. During recovery, the model adaptor continually approximates the nonlinear system as linear discrete-time models, facilitating the generator to efficiently obtain a recovery control sequence to guide the CPS toward a target state before the safety deadline.

• Our proposed method optimizes the use of uncompromised sensor data to mitigate uncertainty accumulation. By incorporating accurate sensor measurements as feedback at each activation, the recovery control sequence generator prevents uncertainty growth in uncorrupted dimensions. Furthermore, leveraging reliable sensor measurements during state reconstruction further alleviates the impact of uncertainties.

• The proposed method boasts low computational overhead. First, the state predictor utilizes Flow*, which employs Taylor models as over-approximate representations for nonlinear ODE solutions, thus, significantly enhancing the efficiency of reachability analysis. Second, the model adaptor transforms nonlinear and even nonconvex dynamics into linear discrete-time models. As a result, the recovery controller solves optimization problems for linear systems rather than nonlinear ones, reducing computational overhead. Additionally, the linear approximation is continuously updated with current state estimates and control inputs during recovery, ensuring accuracy within a small range.

• The proposed method checks the system safety before implementing the recovery control sequence. The state predictor performs the safety checking through reachability analysis for the nonlinear system to guarantee a safe recovery. There is a small probability that the recovery control fails to pass the check; then a fail-safe method takes over.

• Our method verifies system safety before implementing the recovery control sequence. The state predictor performs safety checking via reachability analysis for the nonlinear system, guaranteeing safe recovery.

1.3. Organization of Dissertation

The remainder of the dissertation is structured as follows: Chapter 2 provides an overview of the background knowledge related to cyber-physical systems, sensor attacks, and the existing work on attack detection and recovery. Chapter 3 introduces an adaptive sensor attack detection system, which can adjust detection delay and false alarms in real-time to meet varying detection deadlines and improve usability. Chapter 4 presents a formal method-based approach for online computation of a recovery control sequence that guides a linear system under an ongoing sensor attack from its current state to a target state while ensuring no unsafe state is reachable along the way. Chapter 5 proposes a real-time

recovery system for nonlinear CPSs and explores the utilization of uncompromised sensor data to enhance recovery. Finally, Chapter 6 concludes the dissertation and highlights promising directions for future research.

1.4. Previous Publications

This dissertation incorporates a collection of my prior work published in peer-reviewed conferences. Chapter 3's adaptive sensor attack detection work has been published in the 59th ACM/IEEE Design Automation Conference (DAC 2022)[32]. The real-time sensor attack recovery for linear CPSs in Chapter4 consists of work published in the proceedings of the 2020 IEEE Real-Time Systems Symposium (RTSS 2020)[15] and the 2021 ACM SIGBED International Conference on Embedded Software (EMSOFT 2021)[33]. The real-time sensor attack recovery for complex CPSs in Chapter 5 is based on work published in the proceedings of the 29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2023). In addition, the simulation and security toolbox for CPSs in the appendix are demonstrated in RTAS 2023 Brief Presentation. Any views or opinions expressed in the reused material are those of the authors and do not necessarily reflect those of IEEE or ACM.

CHAPTER 2

Related Work and Preliminaries

This chapter begins by summarizing the related work and then proceeds to introduce the background and corresponding preliminaries.

2.1. CPS Security and Current Research Focus

Before security emerged as a concern, control systems focused on addressing faults. *Fault Detection, Isolation, and Reconfiguration* (FDIR) [34] is an area of control that detects anomalies using either a model-based detection system or a purely data-driven system, a process also referred to as *Bad Data Detection*. Isolation involves identifying the device responsible for the anomaly, while reconfiguration entails recovering from the fault. While FDIR systems excel at detecting and eliminating faults caused by natural occurrences or accidents, they fall short in providing security against faults deliberately created by a strategic adversary. This is mainly due to these protection systems generally assuming independent, non-malicious failures. In the realm of security, incorrect model assumptions are the easiest way for an adversary to bypass protections [35]. For instance, FDIR systems have been circumvented in the power grid [36] and chemical processes [22]. Consequently, overcoming these limitations requires the development of more robust defense methods against CPS attacks.

This need has inspired numerous research efforts on defending against sensor attacks. These works generally fall into two categories: proactive and reactive defenses. In the first category, a significant area of focus is tolerating sensor attacks through sensor redundancy, preventing corrupted sensors from impacting the physical system. Sensor redundancy involves both homogeneous sensors (e.g., multiple encoders measuring vehicle speed) and

heterogeneous sensors capable of measuring the same physical parameter (e.g., encoders and GPS sensors, where speed can also be calculated using GPS information). The value input into the controller is produced by fusing readings from redundant sensors. Existing works typically assume that there is at least triple modular redundancy and that less than half of the redundant sensors can be compromised [10], [37], [38]. Provided these assumptions hold true, the corrupted sensors can be tolerated, and the fused sensor data is considered trustworthy. In the second category, the goal is to detect sensor attacks before the system becomes unsafe and recover the physical state to a desired state. These approaches will be detailed further in the following two sections.

2.2. Sensor Attack Detection

One area that has attracted significant attention is the utilization of physical invariants to detect sensor attacks. A physical invariant is defined as an invariant governed by specific physical laws. There are two types of physical invariants commonly employed in the literature. The first type relies on a system model to capture the physical system's dynamics [5], [16], [20]. For instance, a set of differential equations [39] or a machine learning model can be used to describe the motion of a quadcopter. The second type of physical invariant involves sensor correlation, where multiple (heterogeneous) sensors respond correlatively to the same physical aspect simultaneously [17], [18], [40]. For example, pressing the accelerator will increase engine RPM and vehicle speed, as well as affect GPS readings. Attack detection usually consists of two phases. The offline learning phase aims to extract the physical invariant of a system. The online detection phase involves monitoring sensor (and actuation) values from physical observations and identifying anomalies between these values and the expected ones provided by the physical invariant.

2.3. Sensor Attack Recovery

Researchers place great expectations on attack recovery techniques to build upon the benefits of attack detection. CPS recovery is an emerging field that specifically investigates how to correct a CPS's behavior in the face of adversarial attacks.

One potential response to a detected attack is to restart the CPS. However, a physical system may require a significant amount of time to shut down and reboot [3], [15], [41]. During this time, the CPS is taken offline, and serious consequences may occur. Therefore, a better method should respond to detected attacks in an online manner. For instance, when dealing with sensor attacks, it may be more appropriate to restart only the affected sensors instead of the entire CPS. During the sensor reboot time, a recovery method is needed to prevent the system from further drifting and guide it towards a safe state. Moreover, as mentioned, the online recovery needs to meet timing constraints. In other words, this paper pursues real-time attack recovery that brings a CPS back to a safe state before the safety deadline. In the following, we will discuss two major groups of related work from two different research communities, respectively, and explain why they are inapplicable or inadequate to address this recovery problem.

The first group of related work comprises studies from the cybersecurity community. As mentioned earlier, a commonly used attack response approach is to obtain state estimates for the corrupted sensors and continue to use the original controller to restore the physical system [4], [24]–[28]. In reality, this approach merely performs state prediction using a preknown or learned system model, which is only one component of the recovery framework in Chapter 4. Notably, several essential aspects of recovering a CPS under attack are overlooked in these existing works. First, they do not consider the timing constraint, i.e., the safety deadline, and thus cannot guarantee a timely recovery. Second, they still rely on the

original controller, which does not ensure that no unsafe states will be encountered during the recovery.

The second group of related work includes studies from the control community. First, conventional robust control approaches are adept at tolerating disturbances and modeling errors that can be bounded [29], [42]. However, they are inadequate to defend against sensor attacks because the impacts caused by such attacks are difficult to bound, as attackers may arbitrarily manipulate sensor readings [22], [30], [35], [36]. Similarly, Kalman filter-based approaches are also insufficient to handle these malicious sensor attacks [43], [44]. Furthermore, the timing constraint, i.e., the safety deadline, is not considered by these approaches. Second, concerning the hierarchical control architecture, also called the multi-mode architecture at times, the control policy for each mode is usually static. Using static control policies is not suitable for the recovery problem here because, as mentioned earlier, the safety deadline varies at runtime, and it is infeasible to determine the deadline beforehand. To handle the varying deadline, we need a dynamic control policy that can generate recovery control to adapt to different deadlines on the fly. Lastly, on a conceptual level, our recovery problem differs from event-driven control. Although the event of attack detection triggers the recovery controller, the recovery control is time-driven because the generated control actions are still applied periodically [45], [46].

2.4. Overview of Cyber-physical Systems

This section shows the system model. Table 1 summarizes the notations and symbols used in this paper. In general, a typical CPS architecture is shown in Figure 2, including a physical process, controller, sensors, and actuators. The controller controls the physical process to maintain the reference (also known as target or desired) states in a periodic and close-loop manner. At each t^{th} control step, sensors measure the state of the physical system and send them to the controller. Based on the sensor measurements, the controller

obtains the state estimate $\boldsymbol{x}(t)$ of the physical system and generates control inputs $\boldsymbol{u}(t)$ according to its control algorithm. The control inputs are then sent to actuators who apply $\boldsymbol{u}(t)$ to supervise the physical system at a desired (or reference) state. The state of a physical system or the system state $\boldsymbol{x}(t) \in \mathcal{R}^n$ is a vector of size n that represents the number of dimensions of the system state (e.g., velocity, electric current, pressure, etc.). The control inputs $\boldsymbol{u}(t) \in \mathcal{U}$ is a vector of size m that represents the number of dimensions of the control input (e.g. steering angle, applied voltage, etc.). $\mathcal{U}$ is the control input range that is usually limited by the actuator's capability or physical properties, and for instance, the maximum voltage applied on a DC motor is limited by the capacity of the power source. Example 2.4.1 demonstrates the execution of a CPS, DC motor. Moreover, the symbol $\boldsymbol{x}(t)$ denotes the state estimate, while $\bar{\boldsymbol{x}}(t)$ is the real (true) state of the plant. The symbol $\boldsymbol{u}(t)$ denotes the control input computed by the controller, while $\bar{\boldsymbol{u}}(t)$ is the real input to the plant. For easy presentation of equations, we sometimes use x_t or u_t to denote $x(t)$ or $u(t)$, respectively.

It is important to note that feedback control systems are related concepts to CPS. Feedback control systems use a closed-loop design to regulate the behavior of a process or system. However, CPS have a broader scope, as they integrate computational, physical, and networking components. The implementation of each component in real CPS may be complex. For example, there may exist a supervisory and/or configuration device communicating with the controller to monitor the system or change the controller settings. Also, the communication channel between these components can be wireless and separate from each other geographically. For conciseness, this thesis abstracts the main components of CPSs in Figure 2 to avoid splitting hairs, which is adequate to understand how they work.

The dynamics of a physical system obey physics laws and can be modeled by a set of differential or difference equations. The linear time-invariant systems can be modeled as a

Table 1: Notations and symbols used in this dissertation.

Notation	Description
δ	length of one control step / control interval
$\boldsymbol{x}_t$	state estimate at time t
$\boldsymbol{u}_t$	control input to be implemented at time t
X_t	overapproximation of $\boldsymbol{x}_t$
$\mathcal{T}$	target state set $\subset \mathbb{R}^n$
$\mathcal{F}$	unsafe state set $\subset \mathbb{R}^n$, $\mathcal{F} \cap \mathcal{T} = \emptyset$
$\bar{\boldsymbol{x}}_t$	real/true/actual system state at time t
t_w	the time when the last trustworthy state is cached
t_a	unknown time when the sensor attack starts
t_f	the time when the sensor attack is detected
t_r	the time when the first recovery control is implemented
t_d	safety deadline, by which the system is in target set
t_m	maintainable time, such that the system is maintained in target set in $[t_d, t_m]$
D	recovery length, $D = t_d - t_r$
M	maintenance length, $M = t_m - t_d$
N	total recovery control length, $N = t_m - t_r$
T	number of control steps within which one optimization computation is guaranteed to finish
$\boldsymbol{v}_t$	the uncertainty at time t
v_{max}	the maximum value of the uncertainty
n	the dimension of system state vector
m	the dimension of control input vector
J	the objective function of optimization problem
Q	the state cost
Q_f	the final state cost
R	the control input cost
$\oplus$	Minkowski sum, i.e., $X \oplus Y = \{x + y \mid x \in X, y \in Y\}$
$\ominus$	Minkowski difference, i.e., $X \ominus Y = \bigcap_{y \in Y} \{x - y \mid x \in X\}$

state-space form of Equation (2.1),

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{v}_t, \tag{2.1}$$

where $\boldsymbol{x}_t \in \mathbb{R}^n$ denotes the plant's state vector at time t, $\boldsymbol{u}_t \in \mathbb{R}^m$ is the control input vector, $\boldsymbol{v}_t \in \mathbb{R}^n$ is the uncertainty vector, and A, B have suitable dimensions, indicating how future states evolve from the current state and control input. We assume that the uncertainty $\boldsymbol{v}_t$ at any time is constrained by a bounded range V, and use v_{max} to denote its magnitude $\sup_{\boldsymbol{v}\in V} \|\boldsymbol{v}\|$ where $\|\cdot\|$ is the Euclidean norm. The LTI model has been widely used in both control and security communities [4], [15], [16], [42]. Given the current system state and control input, we can predict next system state from the system model, which represents the system's behavior.

Similarly, we can model a continuous nonlinear system using an ordinary differential equation in the form of Equation (2.2).

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{d} \tag{2.2}$$

where $\boldsymbol{d}$ is the disturbance term due to uncertainties.

Observability refers to the ability to estimate the internal state of a system using its control inputs and sensor measurements over time [47]. For concise presentation, we assume that the system states are fully observable, i.e., they can be directly determined from sensor measurements, or $\boldsymbol{x} = \boldsymbol{y}$. However, in more general cases, the state estimates $\boldsymbol{x}$ need to be calculated from sensor measurements $\boldsymbol{y}$ by an observer, given an output equation $\boldsymbol{y} = h(\boldsymbol{x}) + \boldsymbol{v}$, where $\boldsymbol{v}$ is the sensor noise.

Example 2.4.1. *DC motors are extensively used as actuators in electric vehicles and*

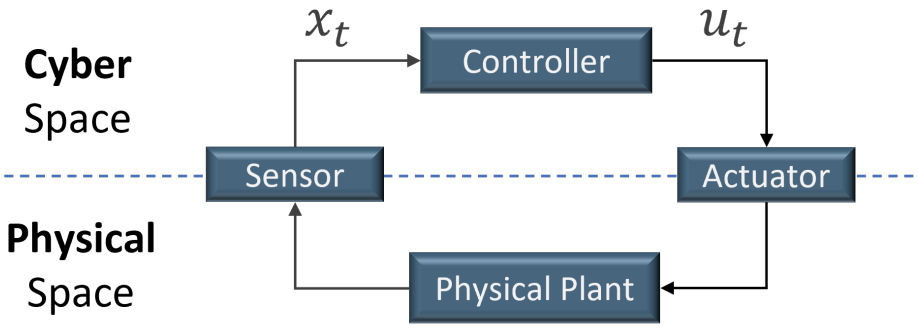


Figure 2: Abstract of CPS Architecture

prototype autonomous cars. A DC motor is equipped with torque converter, transmission, shaft, and wheels, and provides rotary motion. The following ODE models the behavior of a DC motor:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\frac{b}{J} & \frac{K_T}{J} \\ -\frac{K_e}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} u$$

such that x_1 denotes the angular velocity of the motor and x_2 denotes the armature current. The control input is denoted by u which is the voltage applied to the motor, it is updated every 0.02 seconds based on the current value of x_1 by a proportional–integral–derivative (PID) control scheme whose goal is to maintain the angular velocity along a specific (reference) value which could be different in different time periods. Fig. 3 illustrates an execution of the DC motor from the initial state $x_1 = 0$, $x_2 = 0$, where the parameters are set as $R = 1$, $L = 0.5$, $K_T = 0.01$, $b = 0.1$, $J = 0.01$, and $K_e = 0.01$. The blue dashed line in the figure shows the reference values for x_1.

2.5. Sensor Attacks in CPSs

In this section, we present the threat model under consideration. Sensors in CPSs can provide data for the controller to perceive the surrounding environment and monitor the sys-

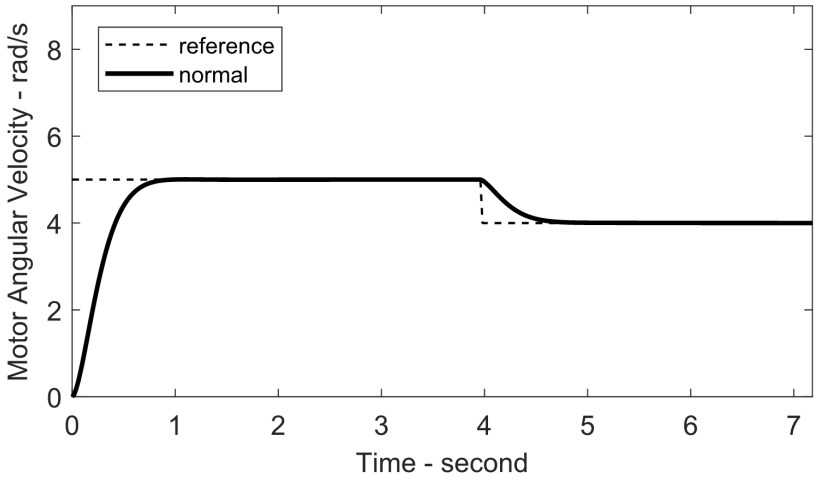


Figure 3: An Execution of the CPS example

tem state, which helps with decision-making and control processes. Sensor attacks, which alter sensor measurements, become a serious and particular threat for CPSs.

2.5.1. Devastating Effects of Sensor Attacks

Sensor attacks alter sensor data and lead to corrupted state estimations, which can not reflect the actual system state, i.e., $\boldsymbol{x} \neq \bar{\boldsymbol{x}}$. Based on the corrupted state estimation, the controller generates an improper control input sent to the actuator. Then, the actuator implements this corrupted control input to deviate the system states from the reference state, and even reach the unsafe states. Note that attackers can predictably manipulate the system states according to their needs by controlling the intensity of the sensor attack. For example, a plant is required to operate at 200 ± 5°F, and a temperature above 240°F will cause an explosion. Once the attacker subtracts 40°F from the temperature sensor measurement, the controller will assume that the plant temperature is 160°F and does not reach the reference temperature of 200°F. As a result, the controller will increase the control input to follow the reference temperature until the actual temperature becomes 240° F, which can cause an explosion. In this case, the attacker can predictably control the temperature according to their goal with the help of the feedback control mechanism. In

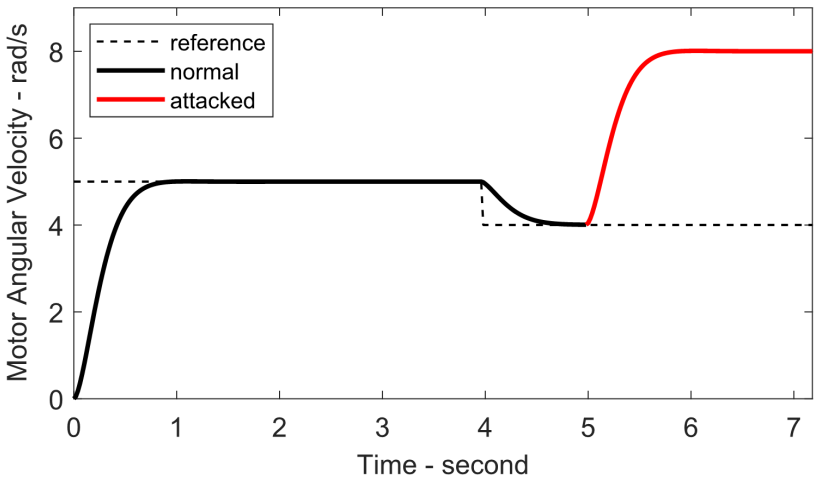


Figure 4: The CPS example under a sensor attack

addition, Example 2.5.1 demonstrates another sensor attack case in the above DC motor system, and we plotted the system state under the sensor attack.

Example 2.5.1. *We show an example of modification attack in Figure 4. Starting from the time* $t = 5$*, the sensor data sent to the controller is modified by an attacker that adds a bias of* 5 *rad/s to* x_1*. Then after a small time period, the PID controller cannot maintain the motor angular velocity near the reference value, which is* 4 *rad/s.*

2.5.2. Categories of Sensor Attacks

The system state is estimated from sensor measurements. Thus, an attacker can manipulate sensor measurements to compromise state estimates. An attacker can compromise state estimates by corrupting the integrity (e.g. transduction attacks) or availability (e.g. DoS attacks) of sensors. These attacks result in misleading control inputs that drive the physical system to undesired states and even cause serious consequences.

• *Compromising Integrity of Sensor Data.* The attack scenarios under this category are bias and reply attacks. (i) Bias attacks. This kind of attack modifies sensor data by adding or subtracting certain values. The difference between state estimate $\bar{\boldsymbol{x}}_t$ and system state

$\boldsymbol{x}_t$ denoted by vector $\boldsymbol{e}_t \in \mathbb{R}^n$. Usually, we cannot know the number of the system state that can be compromised in advance. The state estimate can be partially or fully affected, i.e., the number of non-zero dimensions of $\boldsymbol{e}_t$ or l_0 norm of $\boldsymbol{e}_t$ is $0 < ||\boldsymbol{e}_t||_0 < n$, or $||\boldsymbol{e}_t||_0 = n$. (ii) Replay attacks. This kind of attack sends historical sensor data instead of current ones to the controller. That is, $\boldsymbol{x}(t) = \bar{\boldsymbol{x}}(t-s)$ starting from the attack for some $s > 0$.

• *Compromising Availability of Sensor Data.* The attack scenario we consider here is delay attack. This kind of attack intentionally delays the data sent to the controller, i.e., $\boldsymbol{x}(t) = \bar{\boldsymbol{x}}(t_0)$ for a time period of d where t_0 is the start time of the attack, and then $\boldsymbol{x}(t) = \bar{\boldsymbol{x}}(t-d)$ for $t \geq t_0 + d$. Note that the Denial-of-Service (DoS) attack can also be seen as delay attacks with infinite time delay.

In these attack scenarios, the data observed by the controller may not be consistent with the actual system state, i.e., $\boldsymbol{x}(t) \neq \bar{\boldsymbol{x}}(t)$. As a result, the controller may generate an inappropriate control input based on a corrupted state estimate. In this way, the controller might steer the system to unsafe states with misleading sensor data.

2.5.3. Growing Severity of Sensor Attacks

Sensor attacks are widely recognized as critical threats in CPSs for the following reasons. First, sensor attack surfaces are increasingly expanding as CPSs become more and more complex and open. For instance, there are more than 100 sensors in some modern vehicles, and the number is growing with time. To achieve high-level driving automation, vehicles rely on not only more complicated sensors, such as cameras, LiDAR, and IMU sensors, but also on traffic data through vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-everything (V2X) communication [48]. Security concerns are growing with increasingly open architecture and high autonomy. Second, attackers can launch sensor attacks without expensive equipment or solid domain knowledge. For example, attackers are able to pretend to be road workers and install dirty road patches to compromise the lane

keeping system, causing the vehicle to leave the road [49]. Another example is that an attacker without prior-knowledge of perception algorithms can project pure light onto the stereo cameras to inject a fake obstacle depth [50]. Third, traditional defense mechanisms for cyber systems are inadequate to identify and respond to sensor attacks. Attackers can non-invasively manipulate physical properties in the environment to corrupt sensor data, also known as transduction attacks. For example, an attacker, without physical or cyber access to GPS sensors, can use a radio transmitter broadcasting fake GPS signals to steer a yacht off course [51]. Since all components of CPSs are intact, traditional mechanisms are unable to respond to such attacks. Fourth, a physical system might already have considerably deviated from the desired state before attacks are detected. This is because there is a detection delay from the onset of an attack to its detection. During the time interval, the deviation caused by the above sensor attacks is a devastating repercussion for CPSs.

CHAPTER 3

Real-time Adaptive Detection against Sensor Attacks

In this chapter, we introduce a real-time adaptive sensor attack detection system that dynamically balances usability and safety, depending on the current state estimate of the cyber-physical system (CPS).

3.1. Overview of the Attack-Detection Framework

Our adaptive sensor attack detection framework is illustrated in Figure 5. It consists of three components in the shaded box: (1) *Adaptive Detector*, (2) *Detection Deadline Estimator*, and (3) *Data Logger*. The following briefly introduces these components and their detailed design will be presented in Sections 3.2, 3.4 and 3.3, respectively.

First, the Detection Deadline Estimator conservatively calculates the detection deadline, after which the physical system may reach unsafe states. Thus, the attack detector should find attacks before the deadline. Note that the detection deadline may vary over time as the system state changes and thus the Adaptive Detector needs to adapt the detection delay accordingly. Second, the Data Logger logs state estimates and residuals. At every control period, it predicts expected state and calculates the residual (or difference) between the predicted value and the observed value. It records enough data points for the Adaptive Detector to calculate the cumulative residual, even when the detection delay varies. Third, based on their outputs, the Adaptive Detector will track the cumulative residual. When the average residual in the detection window becomes larger than a predefined threshold, the detector will raise an alarm to signal the detection of an attack. Importantly, the detector can dynamically adapt its detection delay to meet the detection deadline.

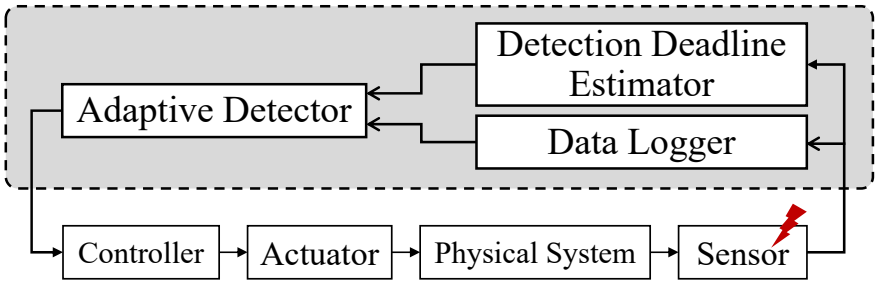


Figure 5: Design overview of the real-time adaptive sensor attack detection system.

3.2. Detection Deadline Estimation

This section presents the design of Detection Deadline Estimator. We use reachability analysis to conservatively estimate the detection deadline. The following first defines the safety analysis problem, then presents how to approximate the reachable set, and finally shows how to calculate the deadline using the support function.

3.2.1. Safety Analysis

When applying a sequence of control inputs $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{T-1} \in \mathcal{U}$ to a physical system, the system state evolves according to its dynamics ψ, i.e., $\boldsymbol{x}_{t+1} = \psi(\boldsymbol{x}_t, \boldsymbol{u}_t)$. The sequence of evolving states is called *State Trajectory* ξ, where ξ_i denotes the i-th state in the trajectory. By applying all possible control sequences within T control steps, we can have all possible state trajectories $\Xi(\boldsymbol{x}_0, T)$ as $\Xi(\boldsymbol{x}_0, T) = \{\xi : \xi_0 = \boldsymbol{x}_0, \xi_{t+1} = \psi(\xi_t, \boldsymbol{u}_t)\}$, where $\boldsymbol{u}_t \in \mathcal{U}$, $t \in \{0, \ldots, T-1\}$, and $\boldsymbol{x}_0$ is an initial state. Then, the reachable set $\mathcal{R}$ includes all possible system states in $\Xi(\boldsymbol{x}_0, T)$.

The *Unsafe State Set* $\mathcal{F}$ is a region within the state space, in which the physical system is unsafe and may cause serious consequences. For example, the distance from a vehicle to an obstacle is less than zero where the unsafe state includes all negative distance values. The complementary set of $\mathcal{F}$ is the safe state set $\mathcal{S}$. To keep the system safe, the reachable set of a system from a certain initial state $\boldsymbol{x}_0$ is required not to intersect with the unsafe state

set, i.e., $\mathcal{R} \cap \mathcal{F} = \emptyset$. Unfortunately, it is very expensive to compute the exact reachable set. Instead, we usually compute an over-approximation of the reachable set, denoted by $\bar{\mathcal{R}}$ and $\bar{\mathcal{R}} \supseteq \mathcal{R}$. If $\bar{\mathcal{R}} \cap \mathcal{F} = \emptyset$, then we can guarantee that $\mathcal{R} \cap \mathcal{F} = \emptyset$. Based on this, we define conservatively safe as Definition 3.2.1.

Definition 3.2.1. *The system is Conservatively Safe, if the over-approximation of the reachable set does not intersect with the unsafe state set $\mathcal{F}$, i.e., $\bar{\mathcal{R}} \cap \mathcal{F} = \emptyset$.*

3.2.2. Over-approximation of the Reachable Set

Given the system model by Eq. (2.1), a state trajectory is affected by both uncertainty and control input. To calculate the reachable set, we need to over-approximate both parts [52]. The over-approximation uses the ball and box defined as follows.

Definition 3.2.2. *A unit ball is the closed set of points whose k-norm distance is less than or equal to 1 from a fixed central point. For n dimensions and any $k > 1$, the origin-centered unit ball $\mathcal{B}_{(k)}$ is defined as $\mathcal{B}_{(k)} = \{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\|_k = (\sum_{i=0}^{n-1} |x_{(i)}|^k)^{\frac{1}{k}} \leq 1\}$.*

Definition 3.2.3. *A box is a set that can be denoted by a product of intervals, i.e., $[x^l_{(1)}, x^u_{(1)}] \times \cdots \times [x^l_{(n)}, x^u_{(n)}]$, where $x^l_{(i)}$ and $x^u_{(i)}$ are the lower and upper bounds of $\boldsymbol{x}$'s the i-th dimension.*

Especially, any 2-norm ball (*Euclidean ball*) can be scaled from a unit Euclidean ball. The infinity-norm unit ball $\mathcal{B}_{(\infty)}$ is a box, i.e., $\mathcal{B}_{(\infty)} = \{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\|_\infty = \max_{0 \leq i < n} |x_{(i)}| \leq 1\}$, where $x_{(i)}$ is the i-th dimension of $\boldsymbol{x}$. Hence, any box can be transformed from an infinity-norm unit ball by scaling in each dimension.

Over-approximation of uncertainty

We assume that $\boldsymbol{v}_t$ in Eq. (2.1) by is bounded an error $\epsilon > 0$ at one control step . Thus, it can be over-approximated by an origin-centered euclidean ball $\mathcal{B}_\epsilon$ with radius ϵ. That is,

we have $\boldsymbol{x}_t \in \tilde{\boldsymbol{x}}_t \oplus \mathcal{B}_\epsilon$, where $\tilde{\boldsymbol{x}}_t = A\boldsymbol{x}_{t-1} + B\boldsymbol{u}_{t-1}$ and $\oplus$ denotes the Minkowski sum. The Minkowski sum is defined as $X \oplus Y = \{x + y | x \in X, y \in Y\}$ for any set X and Y.

Over-approximation of control input set

Consider the control input set $\mathcal{U} = [u_{(1)}, ..., u_{(m)}]$. For the i-th dimension $u_{(i)}$, it has the upper and lower bounds denoted as $u^u_{(i)}$ and $u^l_{(i)}$, respectively. Then, the control input set can be over-approximated by a box $\mathcal{B}_\mathcal{U}$ with a center $\boldsymbol{c}$, where $c_{(i)} = (u^u_{(i)} + u^l_{(i)})/2$. This box can be scaled from a unit infinity-norm ball $\mathcal{B}_{(\infty)}$ with a scaling factor γ_i in the i-th dimension, where $\gamma_i = (u^u_{(i)} - u^l_{(i)})/2$. The box is expressed by the transformation of the infinity-norm unit ball, given by $\mathcal{B}_\mathcal{U} = \boldsymbol{c} + Q\mathcal{B}_{(\infty)}$, where $Q = \text{diag}(\gamma_1, \cdots, \gamma_m)$, i.e., a $m \times m$ matrix with $\{\gamma_1, \cdots, \gamma_m\}$ in its diagonal.

Note that the uncertainty $\boldsymbol{v}_t$ can be over-approximated by a Euclidean ball by nature. In CPS, each actuator has its own control input range or interval. Thus, the control input set can be expressed by a product of these intervals, which is then a box.

Over-approximation of the Reachable set

Given the over-approximated control input set $\mathcal{B}_\mathcal{U}$, the over-approximated uncertainty $\mathcal{B}_\epsilon$, and an initial state $\boldsymbol{x}_0$ according to the system model by Eq. (2.1), the system state $\boldsymbol{x}_t$ will be bounded by the over-approximation of reachable set $\bar{\mathcal{R}}(\boldsymbol{x}_0, t)$, given by Eq. (3.1).

$$\boldsymbol{x}_t \subseteq A^t \bar{\boldsymbol{x}}_0 \oplus \bigoplus_{j=0}^{t-1} A^j B\mathcal{B}_\mathcal{U} \oplus \bigoplus_{k=0}^{t} A^k \mathcal{B}_\epsilon \tag{3.1}$$

3.2.3. Deadline Searching Process

Selection on the initial state $\boldsymbol{x}_0$

First, we calculate the reachable set from the latest trustworthy state estimation $\bar{\boldsymbol{x}}_{t-w_c-1}$ that has just moved outside the detection window, i.e., $\boldsymbol{x}_0 = \bar{\boldsymbol{x}}_{t-w_c-1}$. It correctly reflects

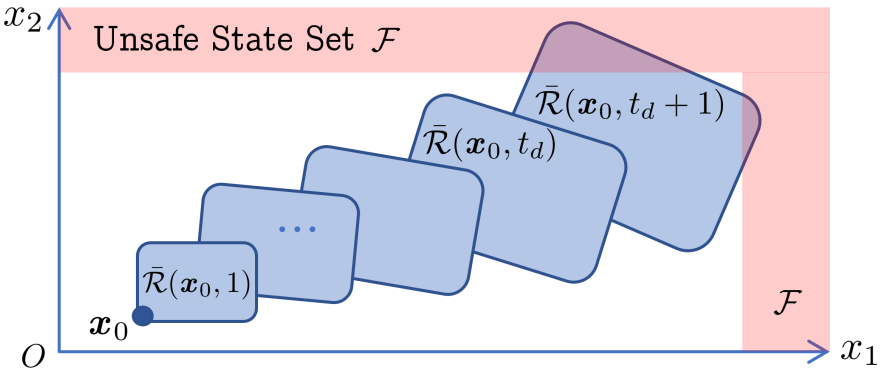


Figure 6: Searching Process for the Detection Deadline t_d.

the physical state at that time, while state estimates within the detection window, i.e., $\{\bar{\boldsymbol{x}}_{t-w_c}, ..., \bar{\boldsymbol{x}}_t\}$, are still questionable. The time point is $t - w_c - 1$, where t and w_c are the current time and window size, as illustrated in Figure 9. (More details on data logging are in Section 3.4.) Second, if we consider some noise in state estimates, we can use an initial state set containing $\boldsymbol{x}_0$. The initial set can be bounded by a ball given bounded noise. Then, we can apply the same reachability analysis as above, too.

Deadline searching process

Starting from $\boldsymbol{x}_0$, we compute the reachable set at each subsequent step until there is an intersection between the over-approximation of reachable set $\bar{\mathcal{R}}(\boldsymbol{x}_0, t)$ and the unsafe state set $\mathcal{F}$, or it exceeds the maximum deadline given beforehand (i.e, the maximum detection window size as in Section 3.3.3). Let say we find the first intersection at the $(t_d + 1)$-th step, the system is safe before this step according to the Definition 3.2.1, and thus t_d is regarded as the detection deadline. The detector is expected to identify an attack within the deadline.

Figure 6 depicts an example that illustrates the detection deadline search process for the system state with two dimensions of x_1 and x_2. At t_d-th step, $\bar{\mathcal{R}}(\boldsymbol{x}_0, t_d) \cap \mathcal{F} = \emptyset$, and at $(t_d + 1)$-th step, $\bar{\mathcal{R}}(\boldsymbol{x}_0, t_d + 1) \cap \mathcal{F} \neq \emptyset$. Hence, the detection deadline is set to t_d.

3.2.4. Computing deadline using support function

According to Eq. (3.1), we can see that the over-approximation is difficult to compute because of the operation of the Minkowski sum. We choose to use the support function method [52] to derive a box over-approximation of the reachable set $\bar{\mathcal{R}}(\boldsymbol{x}_0, t)$. For a vector $\boldsymbol{l}$, the *support function* of a set $\mathcal{S} \subseteq \mathbb{R}^n$ is defined as $\rho_{\mathcal{S}}(\boldsymbol{l}) = \sup_{\boldsymbol{x}\in\mathcal{S}}(\boldsymbol{l}^T\boldsymbol{x})$. Then, according to the properties of support function, we can derive the support function of the reachable set $\bar{\mathcal{R}}(\boldsymbol{x}_0, t)$ using Eq. (3.2).

$$\rho_{\bar{\mathcal{R}}} = \boldsymbol{l}^T(A^t\boldsymbol{x}_0) + \sum_{i=0}^{t-1}\rho_{\mathcal{B}_{\mathcal{U}}}((A^iB)^T\boldsymbol{l}) + \sum_{i=0}^{t-1}\rho_{A^i\mathcal{B}_\epsilon}(\boldsymbol{l}) \tag{3.2}$$

Based on Eq. (3.2), we can have the upper and lower bounds of $\bar{\mathcal{R}}(\boldsymbol{x}_0, t)$ in the i-th dimension, as shown in Eq. (3.3) and (3.4), respectively, where $\boldsymbol{l}$ is set to be a column vector whose i-th entry is 1 and the others are 0.

$$\boldsymbol{l}^TA^t\boldsymbol{x}_0 + \sum_{i=0}^{t-1}\boldsymbol{l}^TA^iB\boldsymbol{c} + \sum_{i=0}^{t-1}\|(A^iBQ)^T\boldsymbol{l}\|_1 + \sum_{i=0}^{t-1}\epsilon\|(A^i)^T\boldsymbol{l}\|_2 \tag{3.3}$$

$$\boldsymbol{l}^TA^t\boldsymbol{x}_0 + \sum_{i=0}^{t-1}\boldsymbol{l}^TA^iB\boldsymbol{c} - \sum_{i=0}^{t-1}\|(A^iBQ)^T\boldsymbol{l}\|_1 - \sum_{i=0}^{t-1}\epsilon\|(A^i)^T\boldsymbol{l}\|_2 \tag{3.4}$$

Finally, by comparing the upper and lower bounds with the unsafe state set (i.e., a similar search process as in Figure 6), we can know when the reachable set has an intersection with the unsafe set, and thus determine the detection deadline.

3.3. Adaptive Window Based Attack Detection

This section presents the design of Adaptive Detector. We enhance window-based detection to accommodate varying window sizes. The following first gives the basic window-based detection and then proposes protocols on adapting the window size or the detection delay according to the deadline from Detection Deadline Estimator.

3.3.1. Basic Window Based Detection

This basic detection algorithm tracks the residual at each control step. The residual $\boldsymbol{z}_t$ is defined as the difference between the predicted state $\tilde{\boldsymbol{x}}_t$ and the state estimate $\bar{\boldsymbol{x}}_t$, where $\tilde{\boldsymbol{x}}_t = A\bar{\boldsymbol{x}}_{t-1} + B\boldsymbol{u}_{t-1}$. Residuals will be provided by the Data Logger.

First, the algorithm will calculate the average residual in the detection window $\boldsymbol{z}_t^{avg} = \frac{1}{w_c}\sum_{i\in[t-w_c,t]} |\tilde{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}_i|$, where t is the current time and w_c is the current detection window size. Then, the algorithm will compare $\boldsymbol{z}_t^{avg}$ with a threshold $\boldsymbol{\tau}$. If $\boldsymbol{z}_t^{avg} \preceq \boldsymbol{\tau}$, no alarm will be raised and the system is regarded as secure. Otherwise, an alarm will be raised to signal the detection of an attack.

Note that there are two hyper-parameters in this algorithm - the threshold $\boldsymbol{\tau}$ and the detection window size. Because we focus on the timing aspect, dynamically adjusting the threshold is not the focus of this paper. The focus is to adjust the detection window size on the fly. To understand the rationale behind this, we need to elucidate the relationship between the window size and the detection delay. Since data points that lay outside the detection window are treated uncompromised, attacked data points are inside the window. Thus, the window size bounds the detection delay, i.e., the maximum detection delay is the window size. Further, with a longer detection delay, a detector tends to have lower false alarm rates but may miss the detection deadline; and vice versa. This is clearly observed in our experimental results in Section 3.5.

3.3.2. Detection Window Adjustment Protocol

Based on the rationale above, our protocol sets the window size as the detection deadline, online calculated by Detection Deadline Estimator. If the deadline decreases, the protocol will shrink the detection window to meet the timing constraint; otherwise, the protocol will enlarge the detection window to reduce false alarms.

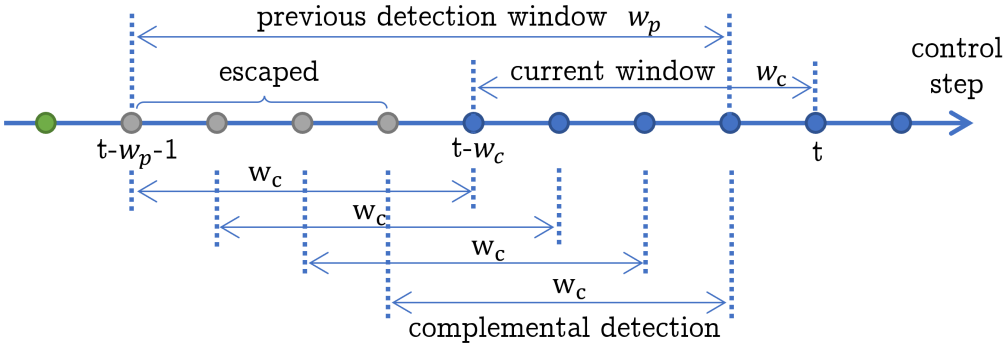


Figure 7: Decreasing the Detection Window Size

Decreasing the window size

Figure 7 shows the case that the detection window is decreasing. The size of the previous detection window is indicated as w_p. The green dots represent the state estimates that move outside the previous window, and their detection results are finalized and trusted to be uncompromised. On the contrary, state estimates within the detection window may be attacked, but not been detected yet. We need to ensure that these data points do not escape detection when reducing the window size. Thus, the protocol performs the following steps.

At the current time t, the detection window becomes w_c, where $w_p > w_c$. First, we set $w_c = t_d$, and t_d is the detection deadline at the current time. Note that the data (marked in grey) within previous detection window but outside current window (i.e., from $t - w_p - 1$ to $t - w_c - 1$) are escaped from the current shorter detection window. Thus, before the detection for current control step t, the complemental detection runs the detection algorithm with window size w_c from control step $t - w_p - 1 + w_c$ to $t - 1$, as shown in Figure 7. By doing this, there will be no data that can escape from the current shorter detection window without checking.

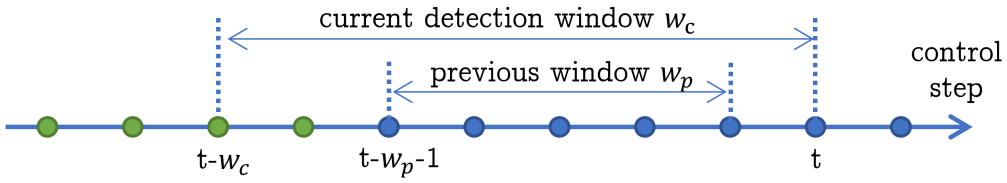


Figure 8: Increasing the Detection Window Size

Increasing the window size

Figure 8 illustrates the case that the detection window size is increasing, where the current detection window w_c is larger than the previous detection window w_p. State estimates marked in green have moved outside the previous detection window, and thus their detection results are finalized. Note that part of these state estimates re-enter the detection window at the current time, but no data points escape from the current longer detection window w_c. Thus, we can continue the window-based detection algorithm with a longer detection window directly, and complemental detection is not needed in this case.

3.3.3. The Maximum Window Size and False Negatives

We predefine a maximum detection window size w_m. At run time, the window size will be adjusted in the range of $[0, w_m]$. If the detection deadline t_d is greater than the maximum window size w_m, then the window size will be set as the maximum, i.e., $w_c = w_m$. Note that as mentioned, this maximum size is also the termination condition for the deadline searching process if no intersection with the unsafe set is found in the first w_m steps.

To decide an appropriate maximum detection window size, we perform offline profiling. The profiling establishes a relationship between false negatives/positives and the window size. We will experiment with a long enough range of window size, and cut out the subrange with an acceptable false negative rate. This cutting line can be given by a specific application. For example, as shown in Figure 11, to avoid false negative experiments (attacks are not detected), the maximum window size can be set as 35 control steps; to tol-

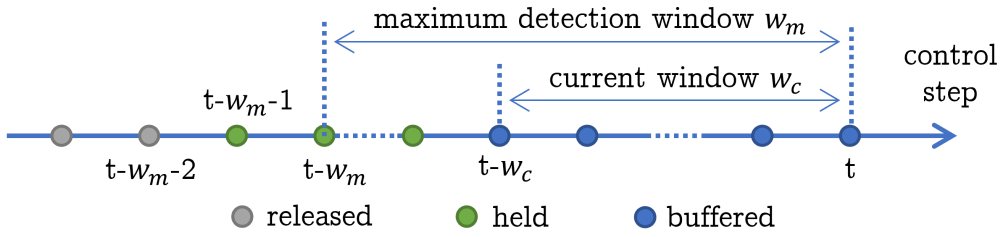


Figure 9: Illustration of the Data Logger.

erate 3 false negative experiments out of 100, the maximum window size can be set as 40 control steps. More details are presented in Section 3.5.

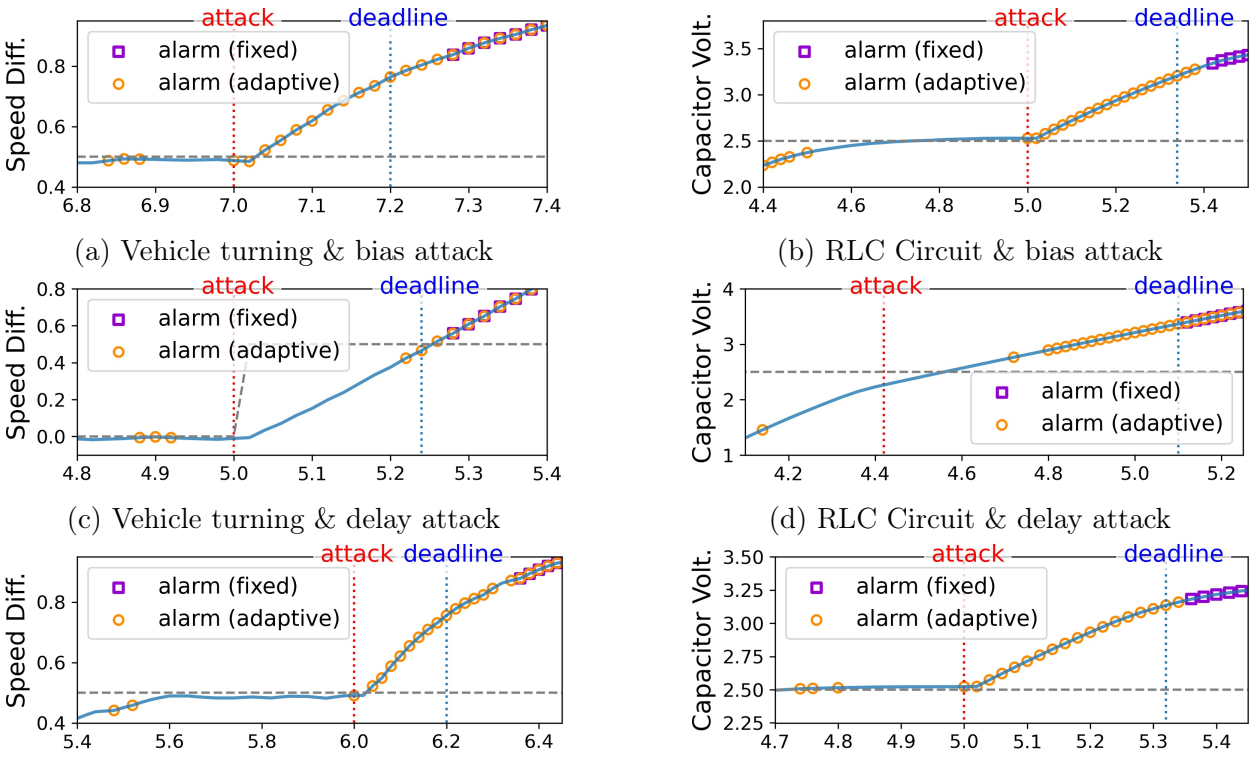
Also, note that adjusting the detection window size makes sense only if attacks can be detected by the window-based detection algorithm. For false negatives, regulating the threshold τ in Section 3.3.1 is more desired, but this is not the focus of this paper. Instead, this paper focuses on the timing aspect, i.e., the detection delay.

3.4. Data Logging Protocol

This section presents the design of the Data Logger. We adapt a sliding-window-based logging protocol to record historical residuals and state estimates [15]. To keep sufficient data for the other two components, the sliding-window size is set as that of the maximum detection window (given in Section 3.3.3).

The sliding window moves forward as time passes. At each control step t, the protocol buffers, holds, and releases certain data points. As shown in Figure 9, the workflow is as follows.

Buffer. Using the current state estimate $\bar{\boldsymbol{x}}_t$, we first calculate the residual $\boldsymbol{z}_t = |\tilde{\boldsymbol{x}}_t - \bar{\boldsymbol{x}}_t|$, where $\tilde{\boldsymbol{x}}_t = A\bar{\boldsymbol{x}}_{t-1} + B\boldsymbol{u}_{t-1}$. Then, $\bar{\boldsymbol{x}}_t$ and $\boldsymbol{z}_t$ are buffered, as shown by the blue dots. These data are within the current detection window w_c, and whether they are intact is still unknown as the detector is still checking them.



(a) Vehicle turning & bias attack

(b) RLC Circuit & bias attack

(c) Vehicle turning & delay attack

(d) RLC Circuit & delay attack

(e) Vehicle turning & replay attack

(f) RLC Circuit & replay attack

Figure 10: Comparison of detection results between adaptive window size and fixed window size for vehicle turning and RLC circuit under three attack scenarios. Blue solid line: actual system state, Grey dashed line: reference state, Red dotted vertical line: attack start time, Blue dotted vertical line: detection deadline, Orange circle marker: alert raised by adaptive detector, Purple square marker: alert raised by detector with fixed window size, Diff.:Difference, Volt.:Voltage.

Hold. The data that has moved outside the current window is regarded trustworthy and thus held, as shown by the green dots.

Release. The historical data before $t - w_m - 1$ are outside the sliding window and no longer need to be used, as shown by the grey dots. Thus, these data can be released to save storage space.

Table 2: Simulation settings. No.: simulator number, δ: control step size (in seconds), PID: PID control parameters, U: control input range, ϵ: uncertainty bound, $\mathcal{S}$: safe state set, $\boldsymbol{\tau}$: detection threshold.

No.	Simulator	δ	PID	U	ϵ	$\mathcal{S}$	$\boldsymbol{\tau}$
1	Aircraft Pitch	0.02	14,0.8,5.7	$[-7, 7]$	7.8e−3	$z \in [[-\infty, -\infty, -2.5], [\infty, \infty, 2.5]]$	$[0.012, 0.012, 0.012]$
2	Vehicle Turning	0.02	0.5,7,0	$[-3, 3]$	7.5e−2	$z \in [-2, 2]$	$[0.07]$
3	Series RLC Circuit	0.02	5,5,0	$[-5, 5]$	1.7e−2	$z \in [[-3.5, -5], [3.5, 5]]$	$[0.04, 0.01]$
4	DC Motor Position	0.1	11,0,5	$[-20, 20]$	1.5e−1	$z \in [[-4, -\infty, -\infty], [4, \infty, \infty]]$	$[0.118, 0.118, 0.118]$
5	Quadrotor	0.1	0.8,0,1	$[-2, 2]$	1.56e−15	$z \in [-5, 5]$	$[0.018, ..., 0.018]$

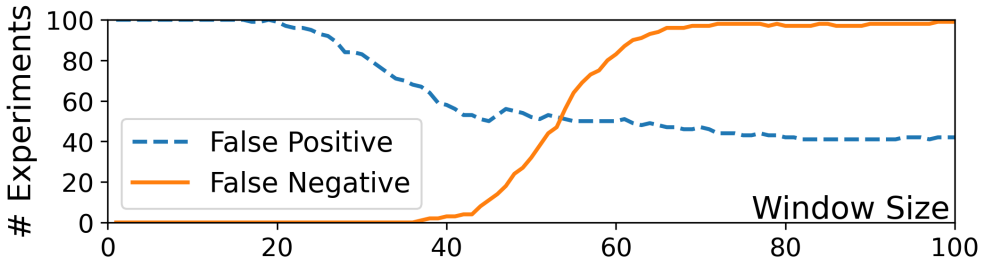


Figure 11: The number of false positive and false negative experiments changes with different window sizes.

3.5. Evaluation

3.5.1. Simulation Setting and Results

Settings

We develop a simulation tool that can load different system models to simulate different physical systems. We consider 5 LTI models: aircraft pitch, vehicle turning, series RLC circuit, DC motor position, and quadrotor, which are used in [15], [33], [53], [54]. The detailed simulation setting is listed in Table 2.

We consider three sensor attack scenarios: bias, delay, and replay attack. Bias attack replaces sensor data with arbitrary values. Delay attack delays sensor measurements sent to the controller for a certain time period, so that the controller cannot update the current state estimate in time. Replay attack replaces sensor data with previously recorded ones.

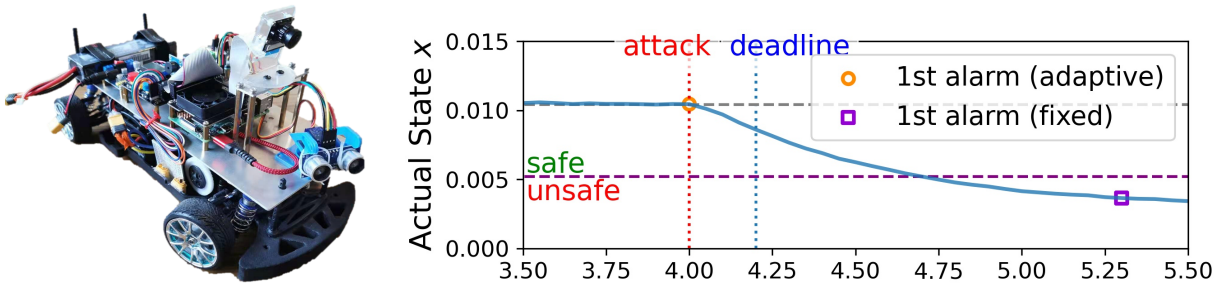
The impact of different window size

The simulation is performed on the aircraft pitch simulator under a bias attack lasting 15 control stepsize (0.3s). We perform 100 experiments for each window size from 0 to 100. It is counted as a false positive experiment if the false positive rate exceeds 10% and counted as a false negative experiment if the attack is not detected. The number of false positive and negative experiments is plotted in Figure 11. The result shows that the false positive number decreases and false negative number increases with increasing window size. According to Section 3.3.3, we choose a maximum detection window whose false negative number is acceptable for the application. Take aircraft pitch simulator as an example, we choose the maximum window size as 40, and the corresponding false negative number is only 3.

Results of the Adaptive Detector

We test our adaptive detection method under all 15 cases (i.e., all the combinations of 5 simulators and 3 attack scenarios). Figure 10 shows part of the results using vehicle turning and RLC circuit simulator under bias, delay, and replay attacks. In all figures, we can see that our adaptive detector can raise alerts before the detection deadline, i.e., in-time detection, while the detector with a fixed window size finds attacks after the deadline, i.e., untimely detection. Note that our adaptive detector may raise some false alarms before real attacks are launched. This is because our adaptive detector chooses a smaller window size to catch up with the detection deadline while increasing the false positives. Note that this situation only occurs when the states are closer to the unsafe set. In practice, we consider noise in our experiments, which is also one of the reasons for false positives.

Table 3 shows all false positive and deadline miss numbers out of 100 simulations for each case. The results indicate that our adaptive detector tends to have larger false positive numbers, but with minimal deadline misses. This is because our detector will choose a



(a) Vehicle Testbed (b) Attack Detection

Figure 12: Attack detection in vehicle testbed.

smaller detection window to catch the detection deadline when the state is close to unsafe states. Note that our adaptive detector may miss the detection deadline in some cases, for instance, just 3 out of 100 experiments for DC motor under delay attack, because those attacks have a negligible effect on the physical system.

3.5.2. Testbed Configuration and Results

Testbed Configuration

We build a testbed (Figure 12a) on a scaled RC car runing a cruise control task with a PID controller. The controller reads a magnetic rotation sensor AS5048A and computes the speed data at 20Hz. We perform system identification and get system model as $\boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t, \boldsymbol{y}_t = C\boldsymbol{x}_t$, where $A = 8.435\text{e}{-1}, B = 7.7919\text{e}{-4}, C = 3.843402\text{e}2$. The vehicle runs in a straight line at a speed of $4m/s$, and at the end of the 79^{th} step, a bias of $+2.5m/s$ is added to the speed. The safe speed range is $[2, 10]$, so safe state range is $[2/C, 10/C]$, i.e. $[5.2\text{e}{-3}, 2.6\text{e}{-2}]$. The threshold τ is set to $3.67\text{e}{-3}$. The control input range is $u \in [0, 7.7]$.

Testbed Results

The testbest result is shown in Figure12b, where the y-axis is actual states $\boldsymbol{x}$ (equals to $\boldsymbol{y}/C$), and the x-axis is time. The purple horizontal line marks the boundary between safe and unsafe states, and the detector should raise an alert before the states reach the unsafe

Table 3: Comparison of detection false positives and deadline misses with adaptive window size vs. a fixed window size. #FP: number of simulations whose false positive rate exceeds a threshold, #DM: number of simulations who miss deadline.

Simulator	Attack	Strategy	#FP	#DM
Aircraft Pitch	Bias	Adaptive	73	0
		Fixed	42	96
	Delay	Adaptive	3	0
		Fixed	2	97
	Replay	Adaptive	34	0
		Fixed	8	100
Vehicle Turning	Bias	Adaptive	71	0
		Fixed	5	34
	Delay	Adaptive	13	76
		Fixed	0	100
	Replay	Adaptive	40	10
		Fixed	3	36
Series RLC Circuit	Bias	Adaptive	54	0
		Fixed	44	65
	Delay	Adaptive	4	0
		Fixed	3	73
	Replay	Adaptive	13	0
		Fixed	5	92
DC Motor Position	Bias	Adaptive	82	0
		Fixed	47	59
	Delay	Adaptive	3	3
		Fixed	3	89
	Replay	Adaptive	3	0
		Fixed	3	46
Quadrotor	Bias	Adaptive	73	7
		Fixed	55	99
	Delay	Adaptive	6	0
		Fixed	2	87
	Replay	Adaptive	6	0
		Fixed	2	59

region. The orange circle is the first alert raised by our proposed adaptive window-based detection, while the purple square is the first one raised by a fixed window-based detection (size=30). We can find our detector alert in the first step after the attack. However, the fixed window-based detection alerts after the vehicle reaching the unsafe state, which may already cause damages. Note that our adaptive detector detects the alert in the first step because the estimator computes the tightest deadline and shrinks the window size, making the average residual within the window larger than the threshold.

3.6. Conclusion

In this chapter, we propose a real-time adaptive sensor attack detection system that has three key components. For Adaptive Detector, we develop a window-based detection algorithm that can dynamically adapt the detection delay and thus false alarms to meet the detection deadline and improve usability. For Detection Deadline Estimator, we develop a reachability analysis based technique to conservatively estimate the detection deadline at run time by computing the reachable set of future potential behaviors of systems. For Data Logger, we adapt a sliding-window-based data logging protocol to keep trustworthy data for deadline estimation and also sufficient data points for attack detection. Finally, we implement our detection system in multiple CPS simulators and a reduced-scale autonomous testbed to validate its efficiency and efficacy.

CHAPTER 4

Real-time Sensor-attack Recovery in Linear CPSs

In this chapter, we propose a real-time attack recovery method for linear CPSs to mitigate the negative impact caused by sensor attacks. This approach is designed to extend the benefits of attack detection by restoring the affected physical system to a safe and desired state within a safety deadline.

4.1. Design Overview of Recovery System

This work follows the novel real-time recovery framework is illustrated in Figure 13. The framework has two operating modes: *normal* and *recovery* mode. The attack detector determines whether the system is under attack. Once the detector identifies an attack, the system will be switched from the normal mode to the recovery mode. As mentioned in the introduction, the attack detection is outside the scope of our paper, and we assume an existing detection method that works with our recovery, such as [5], [16], [17], [20], [55]. Although attack detection is a flourishing track in CPS security, how to extend its main benefits and secure the system is still an open question. This paper aims to fill this gap.

Note that as mentioned above, the recovery problem studied in this paper is a reactive procedure. We assume there is an attack detector already in place and the detector can give us the time when an attack starts. Our goal is to take the alerts that are generated by the detector and respond to them in order to recover the physical system.

Recovery Mode. Switching to the recovery mode, the *recovery controller* takes over the system. The recovery controller consists of three components, as shown by the shaded boxes in Figure 13: (i) recovery control calculator, (ii) state reconstructor, and (iii) deadline estimator. The following briefly describes these components, and we will provide their

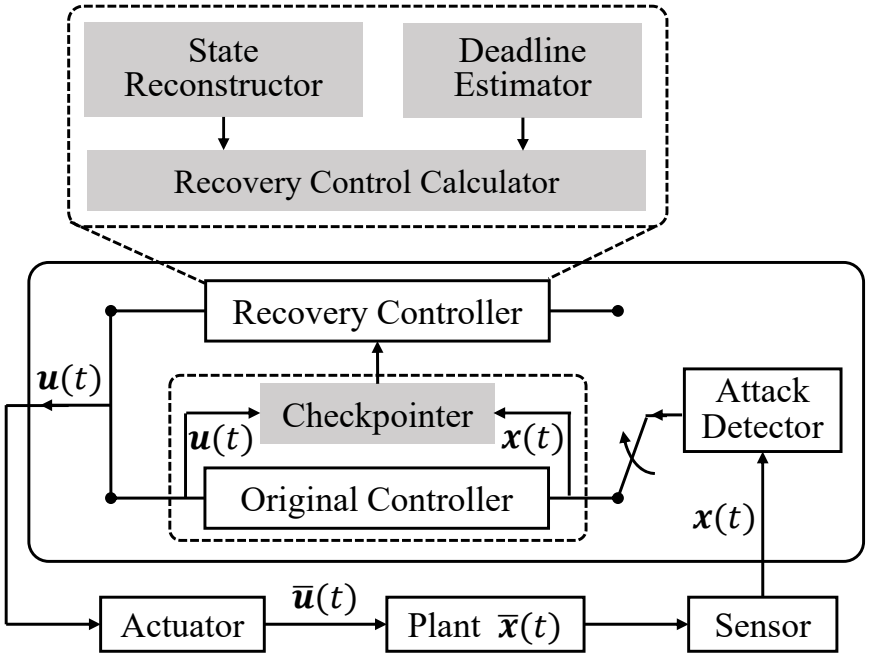


Figure 13: The Framework of Real-Time Attack-Recovery.

detailed design in Sections 4.2 and 4.3.

• *Recovery Control Calculator.* It takes a substantial time for attack detectors to identify the attack after it is launched [4], [5], [20]. During this delay, the attack may drive the system to an undesired state. Thus, this component can compute a Piece-Wise Constant (PWC) control sequence that restores the system from a compromised state into a target state set within a safety deadline and maintain the system in this set before the maintainable time. The control sequence starts from a time point that is close to when an attack is detected. The initial states and two deadlines are obtained from the following two supporting components.

• *State Reconstructor.* Due to a sensor attack, the state estimates may incorrectly reflect the system state during the detection delay. Based on the trustworthy historical data from the checkpointer, this component can reconstruct the state estimate when the recovery control sequence is applied.

• *Deadline Estimator.* This component estimates a safety deadline by which the system should be recovered to a target state set. The deadline is a conservative estimation of the lastest safe time after which the system may reach the unsafe state set and cause serious consequences. Meanwhile, a maintainable time is also estimated, before which the system state can be kept in the target state set.

Normal Mode. In the normal mode, the system runs the original controller, and system states follow the reference or target states. We use a *checkpointer* to record historical data, including state estimates $\boldsymbol{x}(t)$ and control inputs $\boldsymbol{u}(t)$. It uses a sliding window to compensate for the maximum detection delay, during which the attack may corrupt the data but not be detected. The data outside this window is trustworthy and are provided to reconstruct state estimates in the recovery mode.

Discussion on the Proposed Framework

The framework has two controllers: the original controller already in the system and the recovery controller proposed by this work. First, the recovery controller is an extension to an existing system rather than a substitution, and the original plant dynamics and the control algorithm remain unchanged.

Second, this framework can be seen as an extension of the simplex architecture [56]–[58], which consists of a "complex" controller and a "safety" controller. Our framework extends this architecture by adding the four new components, checkpointer, state reconstructor, deadline estimator, and recovery control calculator, as shown in Figure 13.

Third, the proposed framework is different from event-driven control. In our framework, although the recovery controller is triggered by an event of attack detection, the controller is periodic or time-triggered, i.e., with equidistant sampling intervals. That is, the event of the detection of an attack only makes the system switches from the original controller

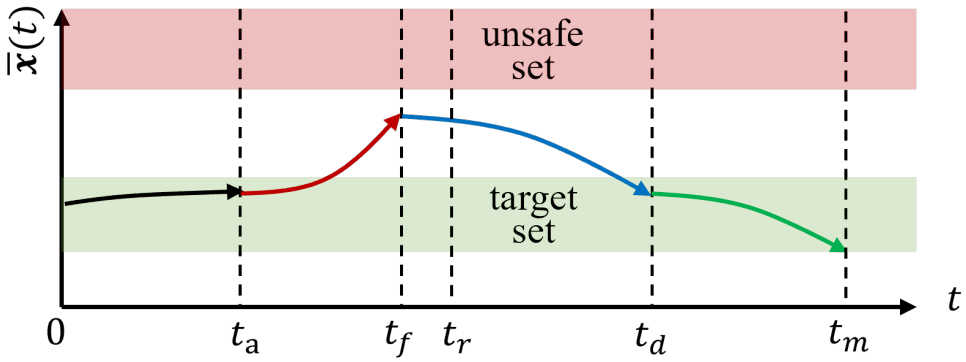


Figure 14: Recovering a system under an sensor attack.

to the recovery controller, which is still time-triggered. By contrast, the sampling is event-triggered in the regime of event-driven control [45], [46]. Hence, event-driven control is inapplicable to our recovery problem.

Fourth, this framework handles multiple types of sensor attacks, which are illustrated in section 2.5. By contrast, some works require the attack to belong to a particular class. For example, some works from networked control systems (NCS) domain pay more attention to the delay attack. They explore to stabilize NCS in the presence of network-induced delay that is inherent to NCS [59], [60].

Recovery Control Sequence

We illustrate an example of the use of recovery control in Figure 14. The system works normally from the start of the time $t = 0$. A sensor attack starts at the time $t = t_a$ and is detected by the attack detector at the time $t = t_f$. Here, we do not require a particular value for t_f but assume that during the time interval $[t_a, t_f]$, although the system is drifted by the attack, no unsafe state is reached. It can be fulfilled by a well-designed attack detector. Our recovery framework takes over the control of the system at $t = t_f$, it firstly computes a recovery control sequence, and then applies it at the time $t = t_r$. The value of $t_r - t_f$ is the maximum bound for the computational overhead of the recovery control and

we assume that it can be conservatively estimated by offline-profiling. The framework is required to obtain the recovery control before t_r.

The recovery control is a piecewise constant sequence which should satisfy the following properties. (i) It consists of two parts: the first D steps, $D = t_d - t_r$, is the *recovery period*, while the second part is the *maintenance period* which has M steps where $M = t_m - t_d$. (ii) The D control steps in the recovery period are guaranteed to steer the system to a state in the target set without reaching any unsafe state. (iii) After the system is recovered, the M control steps in the maintenance period can still keep the system in the target set till the time $t = t_m$. The reason to have the maintenance period is to allow the system to tolerate attacks after the recovery period [4]. For example, to have a time period for resetting the attacked sensors to make them trustworthy again if possible. After the maintenance period, the control of the system is given back to the original controller.

Solving a recovery problem on a system is to find a recovery control satisfying the above properties. However, it requires to compute the key parameters including at least D, M, and the control inputs. Since all of the parameters are dependent, finding the best values for them requires to solve a complex optimization problem and the high time cost does not allow it to be used in an online mode. Hence, we use the 4-step approach presented in Figure15 to find a sound recovery control sequence instead of the optimal one. In the following sections, we firstly introduce the core part which is the Step 3 and 4 in the figure and then present our methods for Step 1 and 2 using reachability computation.

4.2. LQR based Recovery Control Calculator

In this section, we present the design of the recovery control calculator. The component computes a PWC control sequence for real-time recovering a CPS from a sensor attack and based on a Linear Time-Invariant (LTI) model of its plant dynamics. First, we en-

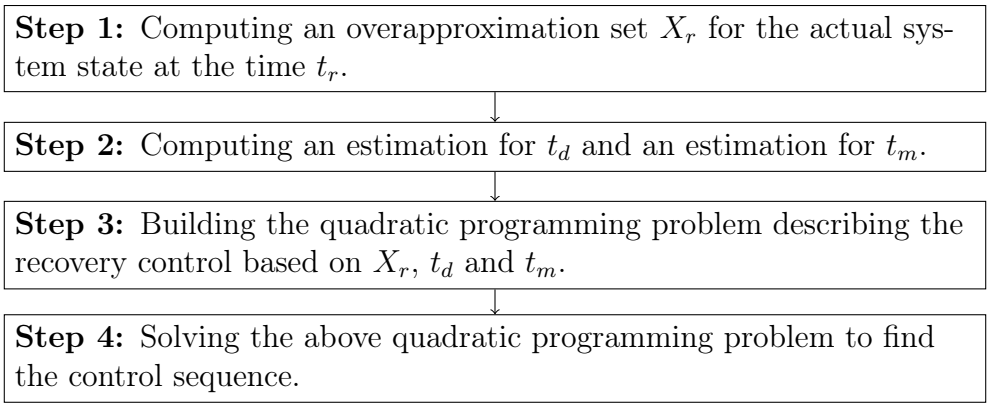


Figure 15: High-level description of our approach to find a recovery control

code the problem of finding such a sequence using a Linear-Quadratic Regulator (LQR) with constraints. The result is guaranteed to safely recover the original system to a target set given the LTI model. In addition, the recovery trajectory is free of oscillations due to the quadratic cost function of states and control inputs. Second, we present an Alternating Direction Method of Multipliers (ADMM) based algorithm to solve the LQR-based recovery problem. The algorithm decomposes a global problem over states and control inputs jointly into two small local subproblems over them separately, which are much easier to handle. Compared to the global problem, the solving time of iterative subproblems is reduced. Third, we also present the discussion on the soundness and completeness of the result.

4.2.1. Recovery Problem Formulation

We consider an LTI system and the dynamics is given by Eq. (2.1). The Linear Quadratic Regulator (LQR) [61] is a well-known optimal feedback controller in control community. It describes the cost as two quadratic terms of states $\boldsymbol{x}$ and control input $\boldsymbol{u}$, which jointly considers the control performance and actuator effort.

As in Figure 14, the (absolute) safe deadline is t_d, estimated by the deadline estimator, and there are $D = t_d - t_r$ control steps before this deadline. In order to stabilize the system

states, $M = t_m - t_d$ control steps are added after t_d, and the physical states are restrained within target set X_T during the M control steps. We use LQR with discrete-time finite horizon to formulate our problem. This recovery process is an optimization problem, and we define the objective function as

$$J(\boldsymbol{x}_r, \ldots, \boldsymbol{x}_l, \boldsymbol{u}_r, \ldots, \boldsymbol{u}_{l-1}) = \sum_{i=r}^{l-1} \left(\boldsymbol{x}_i^T Q \boldsymbol{x}_i + \boldsymbol{u}_i^T R \boldsymbol{u}_i \right) + \boldsymbol{x}_l^T Q_f \boldsymbol{x}_l \tag{4.1}$$

where $\boldsymbol{x}_i$ is the states of LTI system during recovery, $\boldsymbol{u}_i$ is the input used in the i-th step in the recovery control, optimization horizon $N = D + M$ is the number of recovery steps from $t = t_r$ to t_m, and $Q, Q_f \in \mathbb{R}^{n \times n}, R \in \mathbb{R}^{m \times m}$ are semi-definite symmetric matrices that define the state cost, final state cost, and input cost respectively. With properly designed Q, Q_f and R, minimizing J is to minimize the deviation of the states from the target set as well as the size of control signals required within the horizon.

Note that, if we replace the objective function as a constant value, the optimization problem becomes a linear programming problem. This recovery method is also known Linear-programming (LP) based recovery, which has lower computational overhead. However, the recovery trajectory may oscillate since there is no state and control input cost.

To understand the formal definition of a recovery control, we first introduce the notations $\oplus$ and $\ominus$, respectively for Minkowski sum and difference, i.e.: $X \oplus Y = \{x + y \,|\, x \in X, y \in Y\}$, $X \ominus Y = \bigcap_{y \in Y} \{x - y \,|\, x \in X\}$.

Given that the system state is $\boldsymbol{x}_r$ at the time $t = t_r$, we have an over-approximation set X_r around $\boldsymbol{x}_r$, a safe set X_S and a target set X_T. A recovery control is a solution of the

following constraint:

$$\phi(\boldsymbol{x}_r, \ldots, \boldsymbol{x}_l, \boldsymbol{u}_r, \ldots, \boldsymbol{u}_{l-1}) := (\boldsymbol{x}_r = \text{center}(X_r)) \tag{4.2a}$$

$$\wedge \bigwedge_{i=r}^{l} (\boldsymbol{x}_i \oplus \mathcal{B}_i \subseteq X_S \ominus A^{i-r}\mathcal{I}_R) \tag{4.2b}$$

$$\wedge \bigwedge_{i=d}^{l} (\boldsymbol{x}_i \oplus \mathcal{B}_i \subseteq X_T \ominus A^{i-r}\mathcal{I}_R) \tag{4.2c}$$

$$\wedge \bigwedge_{i=r}^{l-1} (\boldsymbol{u}_i \in U) \wedge \bigwedge_{i=r}^{l-1} (\boldsymbol{x}_{i+1} = A\boldsymbol{x}_i + \boldsymbol{B}u_i) \tag{4.2d}$$

such that $\mathcal{B}_V$ is a box around $\boldsymbol{x}_i$ representing some uncertainty in evolution, and $\mathcal{I}_R$ is an box around $\boldsymbol{x}_r$ containing X_r. Formally, $\mathcal{B}_i$ is an origin-centered box whose radius is $v_{\max} \cdot \sum_{j=0}^{i-r-1} |A|^j$, and $X_r \subseteq \{\boldsymbol{x}_0\} \oplus \mathcal{I}_R$. For a later state $\boldsymbol{x}_i$, its over-approximation box should be $A^i\mathcal{I}_R$ based on the dynamics, and we use this box as tolerance, i.e. if the state $\boldsymbol{x}_i$ is inside $X_S \ominus A^{i-r}\mathcal{I}_R$ then we consider it safe, and inside $X_T \ominus A^{i-r}\mathcal{I}_R$ then we consider it hitting the target set.

Referring to Figure 14, the sub-constraint (4.2a) denotes that the first state at $t = t_r$, $\boldsymbol{x}_r$ is the center of X_r. (4.2b) requires that the state $\boldsymbol{x}_i$ is always inside the safe set X_S, considering the accumulation of the uncertainty $\mathcal{B}_i$. (4.2c) requires that the state $\boldsymbol{x}_i$ is always within target set after the deadline t_d, still considering the uncertainty. Finally, (4.2d) defines the control input range and system dynamics, by which the evolution follows. The correctness of the formulation is proved in [15].

Then an LQR recovery control can be obtained by solving the following problem:

$$\min J(\boldsymbol{x}_r, \ldots, \boldsymbol{x}_l, \boldsymbol{u}_r, \ldots, \boldsymbol{u}_{l-1}) \qquad s.t. \quad \phi(\boldsymbol{x}_r, \ldots, \boldsymbol{x}_l, \boldsymbol{u}_r, \ldots, \boldsymbol{u}_{l-1}) \tag{4.3}$$

4.2.2. ADMM algorithm

The alternating direction method of multipliers (ADMM) is an optimization algorithm that decomposes a large global problem into small local subproblems and coordinates to find the global solution [62], [63]. With ADMM, we can solve minimization problems with separable objectives and constraints in the form of:

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) + g(\boldsymbol{z}) \\ \text{s.t.} \quad & \boldsymbol{x} \in C_x,\ \boldsymbol{z} \in C_z,\ A\boldsymbol{x} + B\boldsymbol{z} = \boldsymbol{c} \end{aligned} \tag{4.4}$$

with variables $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{z} \in \mathbb{R}^m$, parameters $A \in \mathbb{R}^{p\times n}$, $B \in \mathbb{R}^{p\times m}$, and $\boldsymbol{c} \in \mathbb{R}^p$, and convex functions f and g. We form the augmented Lagrangian function for (4.4) as

$$L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \boldsymbol{y}^T(A\boldsymbol{x} + B\boldsymbol{z} - \boldsymbol{c}) + (\rho/2)\|A\boldsymbol{x} + B\boldsymbol{z} - \boldsymbol{c}\|_2^2 \tag{4.5}$$

where $\rho > 0$ is the penalty parameter and $\boldsymbol{y} \in \mathbb{R}^p$ is the Lagrangian multiplier.

ADMM solves problem using the following iterations:

$$\boldsymbol{x}^{(k+1)} := \underset{\boldsymbol{x}\in C_x}{\operatorname{argmin}} L_\rho\left(\boldsymbol{x}, \boldsymbol{z}^{(k)}, \boldsymbol{y}^{(k)}\right) \tag{4.6a}$$

$$\boldsymbol{z}^{(k+1)} := \underset{\boldsymbol{z}\in C_z}{\operatorname{argmin}} L_\rho\left(\boldsymbol{x}^{(k+1)}, \boldsymbol{z}, \boldsymbol{y}^{(k)}\right) \tag{4.6b}$$

$$\boldsymbol{y}^{(k+1)} := \boldsymbol{y}^{(k)} + \rho\left(A\boldsymbol{x}^{(k+1)} + B\boldsymbol{z}^{(k+1)} - \boldsymbol{c}\right) \tag{4.6c}$$

Each iteration includes an $\boldsymbol{x}$-minimization step (4.6a), a $\boldsymbol{z}$-minimization step (4.6b), and a dual update step (4.6c). Since the objective function is separable, (4.5) can be minimized over f and g separately in step (4.6a) and (4.6b). Usually, it is easier to solve these two sub-problems than to solve the global problem directly, and the total solving time can be reduced.

4.2.3. ADMM-Based Control Sequence Calculation

Our recovery problem (4.3) can be solved by ADMM method, because the optimization variables $\boldsymbol{x}$ and $\boldsymbol{u}$ are decoupled and thus separable. To construct a separable objective function and equality constraint, we define the following matrices:

$$\bar{Q} = \begin{bmatrix} Q & & & & \\ & Q & & & \\ & & \ddots & & \\ & & & Q & \\ & & & & Q_f \end{bmatrix}, \ \bar{R} = \begin{bmatrix} R & & & \\ & R & & \\ & & \ddots & \\ & & & R \end{bmatrix}, \ \bar{A} = \begin{bmatrix} A & -I & & & \\ & A & -I & & \\ & & \ddots & \ddots & \\ & & & A & -I \end{bmatrix}, \ \bar{B} = \begin{bmatrix} B & & & \\ & B & & \\ & & \ddots & \\ & & & B \end{bmatrix}$$

Therefore, we obtain the matrix form of (4.1) as

$$J(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{x}^T \bar{Q} \boldsymbol{x} + \boldsymbol{u}^T \bar{R} \boldsymbol{u} \tag{4.7}$$

where $\boldsymbol{x} = [\boldsymbol{x}_0^T \ \boldsymbol{x}_1^T \ \cdots \ \boldsymbol{x}_N^T]^T \in \mathbb{R}^{n(N+1)}$, $\boldsymbol{u} = [\boldsymbol{u}_0^T \ \boldsymbol{u}_1^T \ \cdots \ \boldsymbol{u}_{N-1}^T]^T \in \mathbb{R}^{mN}$. The equality constraint (4.2d) becomes $\bar{A}\boldsymbol{x} + \bar{B}\boldsymbol{u} = 0$. Now, we can reformulate our problem in the form of (4.4) for ADMM method. Formally:

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) + g(\boldsymbol{u}) \\ \text{s.t.} \quad & \text{(4.2a)(4.2b)(4.2c)}, \quad \bigwedge_{i=0}^{N-1} (u_i \in U), \quad \bar{A}\boldsymbol{x} + \bar{B}\boldsymbol{u} = 0 \end{aligned} \tag{4.8}$$

The objective function $J(\boldsymbol{x}, \boldsymbol{u})$ can be split into $f(\boldsymbol{x}) + g(\boldsymbol{u})$, where $f(\boldsymbol{x}) = \boldsymbol{x}^T \bar{Q} \boldsymbol{x}$ and $g(\boldsymbol{u}) = \boldsymbol{u}^T \bar{R} \boldsymbol{u}$. Then, we find the augmented Lagrangian function for (4.8):

$$L_\rho(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + g(\boldsymbol{u}) + \boldsymbol{\lambda}^T (\bar{A}\boldsymbol{x} + \bar{B}\boldsymbol{u}) + \frac{\rho}{2} \|\bar{A}\boldsymbol{x} + \bar{B}\boldsymbol{u}\|_2^2 \tag{4.9}$$

Under this ADMM construct, for the (k+1)-th iteration, the $\boldsymbol{x}$-minimizaion sub-problem

is:

$$\boldsymbol{x}^{(k+1)} := \underset{\psi}{\text{argmin}}\ L_\rho(\boldsymbol{x}, \boldsymbol{u}^{(k)}, \boldsymbol{\lambda}^{(k)}), \qquad \text{where } \psi = \ (4.2a)(4.2b)(4.2c) \tag{4.10}$$

the $\boldsymbol{u}$-minimizaion sub-problem is:

$$\boldsymbol{u}^{(k+1)} := \underset{\gamma}{\text{argmin}}\ L_\rho(\boldsymbol{x}^{(k+1)}, \boldsymbol{u}, \boldsymbol{\lambda}^{(k)}), \qquad \text{where } \gamma = \bigwedge_{i=0}^{N-1} (u_i \in U) \tag{4.11}$$

and the dual update step uses the sub-problem results $\boldsymbol{x}^{k+1}$ and $\boldsymbol{u}^{k+1}$ to compute $\boldsymbol{\lambda}^{k+1}$:

$$\boldsymbol{\lambda}^{(k+1)} := \boldsymbol{\lambda}^{(k)} + \rho(\bar{A}\boldsymbol{x}^{(k+1)} + \bar{B}\boldsymbol{u}^{(k+1)}) \tag{4.12}$$

Algorithm 1 shows the procedure to solve our recovery problem. First, at Line 1, we compute the coefficients of the separable objective function and the equality constraints, i.e. $\bar{Q}$, $\bar{R}$, $\bar{A}$ and $\bar{B}$. Line 3-10 are the iterations of ADMM, which solve the two sub-problems (4.10) and (4.11) and do the dual update. Within every iteration, a residual term r is computed to estimate the convergence. When the $||r||_2^2$ is not greater than a tolerance ϵ, the algorithm converges and halts. We output the final control sequence $\boldsymbol{u}$.

Theorem 4.2.1. *Starting from time $t = t_r$, the control input sequence $\boldsymbol{u}$ obtained from our LQR approach is able to recover the system to a state in the target set by the time $t = t_d$, and keep the system inside the target set till the time $t = t_m$.*

Remark 4.2.1. *Due to the formal modeling and methods used to find the recovery control, the obtained control sequence is guaranteed to recover the LTI dynamics if it is applied at the time $t = t_r$. Therefore, our LQR approach is* sound. *On the other hand, in order to achieve good performance, conservative methods are used in overestimating the recovery initial state and the deadlines, which will be described in detail in Section 4.3. It is not guaranteed that our approach can always find a recovery control even there exists one.*

Algorithm 1: ADMM-based Control Input Calculation

Input: Q, Q_f, R, ψ, γ, LTI model, N

/* Q: state cost matrix Q_f: final state matrix R: input cost matrix */

/* ψ: constrains on state $\boldsymbol{x}$ γ: constrain on control input $\boldsymbol{u}$ */

/* LTI model: state space model N: the number of recovery step */

Output: u

/* $u \in \mathbb{R}^{m \times N}$: control input in the last iteration */

Compute $\bar{Q}$ and $\bar{R}$ from Q and R, Compute $\bar{A}$ and $\bar{B}$ from LTI model and N

Initiate proper $\boldsymbol{u}^{(0)}$ and $\boldsymbol{\lambda}^{(0)}$

for $k \leftarrow 0$ *to* MAX_ITERS **do**

  $\boldsymbol{x}^{(k+1)} \leftarrow \underset{\psi}{\operatorname{argmin}}\ L_\rho(\boldsymbol{x}, \boldsymbol{u}^{(k)}, \boldsymbol{\lambda}^{(k)})$ ▷ $\boldsymbol{x}$-minimization

  $\boldsymbol{u}^{(k+1)} \leftarrow \underset{\gamma}{\operatorname{argmin}}\ L_\rho(\boldsymbol{x}^{(k+1)}, \boldsymbol{u}, \boldsymbol{\lambda}^{(k)})$ ▷ $\boldsymbol{u}$-minimization

  $r \leftarrow \bar{A}\boldsymbol{x}^{(k+1)} + \bar{B}\boldsymbol{u}^{(k+1)}$ ▷ calculate residual

  if $\|r\|_2^2 \leq \epsilon$ **then**

    break ▷ stop condition

  else

    $\boldsymbol{\lambda}^{(k+1)} \leftarrow \boldsymbol{\lambda}^{(k)} + \rho\, r$ ▷ dual update

Hence, our approach is not complete.

Remark 4.2.2. *The $O(\frac{1}{\epsilon})$ iteration complexity of ADMM algorithm is shown in [64], where ϵ is the tolerance.*

4.3. Supporting Components for Recovery Control

In this section, we give a detailed description for the design of other components in our real-time attack-recovery framework. We first propose a sliding window based checkpointing protocol to obtain the nearest trustworthy sensor data, which can accommodate varying attack detection delays. Then, based on this, a conservative estimation of the start state of recovery can be obtained using a reachability computation technique. Third, we present a conservative deadline estimation method that uses a safety verification method. Finally, we discuss a conservative way to cover the computation overhead of the three

components in the recovery controller.

4.3.1. A Sliding Window Based Checkpointing Protocol

Attack detection usually comes with a substantial detection delay. Historical state estimates and control inputs during the delay are not trustworthy, i.e., may incorrectly reflect the true physical states and actuation, as they may be already compromised due to the attack. Thus using them can result in unsuccessful recovery. To address this issue, we propose to develop a new sliding window based checkpointing protocol. This protocol will provide trustworthy historical data for the components of the recovery controller.

The protocol proposed in the work [4] assumes a constant detection delay. However, as indicated in the attack detection work [5], [19], [20], the detection delay of approaches based on the cumulative sum of residuals (CUSUM) varies with factors such as attack scale and a drift parameter. To address these limitations, the new protocol can be viewed as a generalization that is applicable to the methods with both constant and varying detection delays.

This new protocol uses a nominal window to accommodate a varying attack detection delay. The length of this window equals the maximum delay of a detection approach. For example, the maximum delay can be analyzed by assuming the worst-case attack (e.g., a stealthy attack) given a drift parameter for CUSUM based detection [5], [20]. The real delay of the detection of an attack can be less than the nominal window. As shown in Figure 16, the interval $[t'_a, t_f]$ denotes the maximum delay and $[t_a, t_f]$ is the true delay, where t_a and t_f are the start and detection of an attack, respectively. The window slides forward as the time ticks. The protocol records estimate $\boldsymbol{x}(t)$ and control input $\boldsymbol{u}(t)$ by the following three steps: buffer, store, and delete.

- *Buffer.* Estimates and control inputs within the window, i.e., $\{(\boldsymbol{x}(t'_a), \boldsymbol{u}(t'_a)), \ldots,$

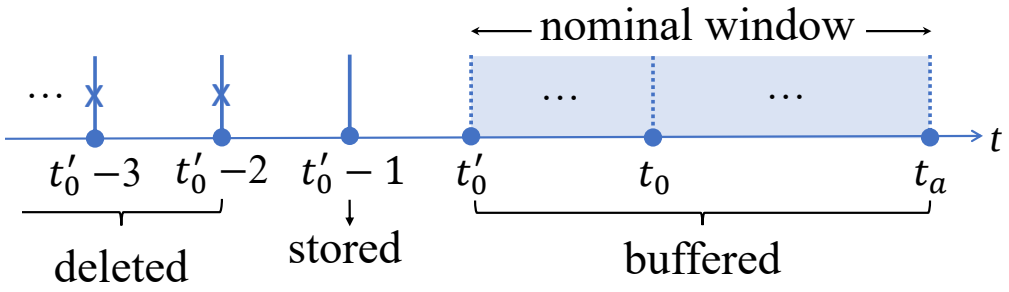


Figure 16: Illustration of the Sliding Window Based Checkpointing Protocol.

$(\boldsymbol{x}(t_a'), \boldsymbol{u}(t_a'))\}$ in $[t_a', t_f]$, are first buffered, because they may be already corrupted and it is still in question whether they are correct. Note that the recovery controller starts to run from the time t_f when the attack is detected. Thus, these data cannot be used for reconstructing $\boldsymbol{x}(t_f)$.

- *Store.* Estimates and control inputs that have moved outside the window are considered to be trustworthy. Thus, $(\boldsymbol{x}(t_a' - 1), \boldsymbol{u}(t_a' - 1))$ is stored.

- *Delete.* The stored data that is no longer needed will be deleted, e.g., $(\boldsymbol{x}(t_a' - 2), \boldsymbol{u}(t_a' - 2))$ is discarded.

When the detector raises an alarm, it gives us the actual time t_a when the attack started (this can be done by finding the time of a breakpoint of the time series of the residuals [5], [20]). Since $t_a \geq t_a'$, the data in $[t_a', t_a]$ has already passed the detection and is also considered as trustworthy. Hence, the data point $(\boldsymbol{x}(t_a - 1), \boldsymbol{u}(t_a - 1))$ will be used to rebuild $\boldsymbol{x}(t_f)$, instead of that of $t_a' - 1$. Using data closer to the time of the detection of an attack can result in better reconstructed estimates. Further, it is worth noting that using the maximum delay as the nominal window guarantees that there is always trustworthy data for state reconstruction.

4.3.2. State Reconstruction

We present the method to construct an overapproximate estimation X_r for the initial system state of recovery, that is the state at the time $t = t_r$ when the recovery control starts to be applied. Given that $\boldsymbol{x}_w$ is the latest trustworthy state recorded at the time $t = t_w$ by our checkpointer, then the actual system state $\boldsymbol{x}(t_r)$ can be overapproximated by the result of the reachability computation from $\boldsymbol{x}_w$ to the time $t = t_r - t_w$.

Given the LTI dynamics (2.1) and the latest trustworthy state $\boldsymbol{x}_w$ recorded at the time t_w, we can obtain the state at time t_{w+1}, i.e, $\boldsymbol{x}_{w+1} = A\boldsymbol{x}_w + B\boldsymbol{u}_0 + \boldsymbol{v}_0$. The the state at time t_{w+2} is $\boldsymbol{x}_{w+2} = A\boldsymbol{x}_{w+1} + B\boldsymbol{u}_1 + \boldsymbol{v}_1 = A^2\boldsymbol{x}_w + AB\boldsymbol{u}_0 + B\boldsymbol{u}_1 + A\boldsymbol{v}_0 + \boldsymbol{v}_1$. In the same way, the state at the following control steps $x_{w+3} \cdots x_a$ can also be derived. Thus, the system reachable state at the time t_f, which is the current time, can be overapproximated by the range of the following linear expression:

$$X_a(\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N_a-1}) = A^{N_a}\ \boldsymbol{x}_w\ +\ \sum_{i=0}^{N_a-1} A^i\, B\, \boldsymbol{u}_i + \sum_{i=0}^{N_a-1} A^i\, \boldsymbol{v}_i$$

where $N_a = t_f - t_w$, $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N_a-1}$ are the historical control inputs used in the past N_a steps respectively, i.e., from the time t_w to t_f, $\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N_a-1}$ are the variables symbolically represent the uncertainty in those steps. Since the uncertainty is not able to be recorded precisely, we use variables constrained by their maximal interval range V to symbolically represent them in the estimation of X_a. Then X_a is essentially a linear constraint over the uncertainty variables.

We then extend the expression of X_a to obtain still a linear expression for overapproximating the reachable state at the recovery start time $t = t_r$ based on the fact that the control

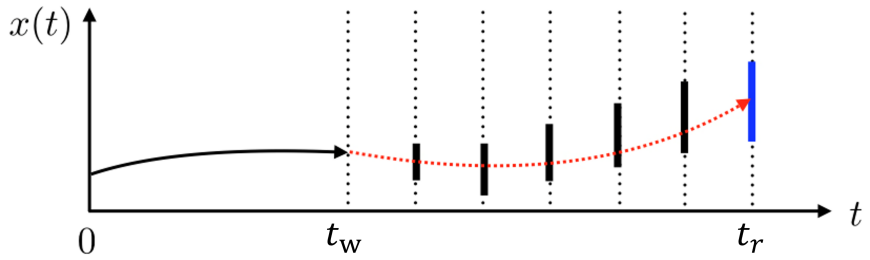


Figure 17: Start state estimation. Line segments: overapproximations of the reachable set at the time $t = t_w + \delta, t_w + 2\delta, \ldots$. The exact system execution which is denoted by the red dotted curve is guaranteed to be contained in the overapproximations at discrete times.

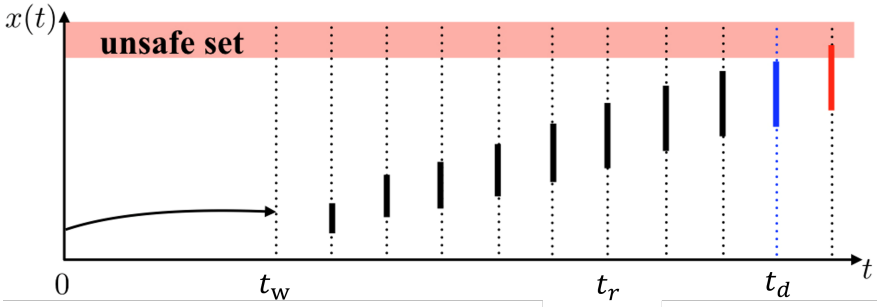


Figure 18: Deadline estimation. Line segments: overapproximations of the reachable set at the time $t = t_w + \delta, t_w + 2\delta, \ldots$. The deadline is computed as $t_d = t_r + 3\delta$.

input will be fixed at $\boldsymbol{u}(t_f)$ after an attack is detected:

$$X_r(\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N_r-1}) = A^{N_r}\ \boldsymbol{x}_w + \sum_{i=0}^{N_a-1} A^i\, B\, \boldsymbol{u}_i + \sum_{i=N_a}^{N_r-1} A^i\, B\, \boldsymbol{u}(t_f) + \sum_{i=0}^{N_r-1} A^i\, \boldsymbol{v}_i$$

where $N_r = t_r - t_w$, and $\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N_r-1}$ are the variable representations for the uncertainty from the time t_w to t_r. X_r is also derived through the evolution of the system states based on the LTI dynamics (2.1). We may use the box (can be computed using support function method in Section 3.2.4) as a conservative estimation for the start state in the recovery problem. The main idea is illustrated in Fig. 17.

4.3.3. Deadline Estimation

We present our approaches to obtain the deadline t_d and t_m.

Estimation of t_d. The purpose of using t_d is to set up a deadline for recovering the system. Since the computation of the optimal deadline requires to solve an optimization problem which is too costly, we use a heuristic to find a feasible control sequence for keeping the system (2.1) safe from any state in X_r, and set $t_d = t_r + D$ where D is the length of the control sequence. To do so, we assume that all of the control inputs used in the future steps from the time t_r are fixed at $\boldsymbol{u}(t_f)$, i.e., we use the constant inputs at the time t_f from the time t_r to estimate the system safety. Then we repeatedly verify the safety for the reachable set overapproximation X_k:

$$\begin{aligned} X_k &= A^k \boldsymbol{x}_0 + \sum_{i=0}^{k-1} A^i \, B \, \boldsymbol{u}(t_f) + \sum_{i=0}^{k-1} A^i \, \boldsymbol{v}_{i+N_r} \\ \boldsymbol{x}_0 &= X_r(\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N_r-1}), \text{ and } \boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N_r+k-1} \in V \end{aligned} \tag{4.13}$$

at the time $t = t_r + k$ for $k = 0, 1, 2, \ldots$ until it has a nonempty intersection with the unsafe set or D reaches the given maximum bound $k_{\max}$. Then D is set to be $k - 1$, or $k_{\max}$ if there is no unsafe intersection till the maximum bound. This deadline estimation process is illustrate in Figure 18.

Estimation of t_m. The purpose to use a maintenance deadline t_m which is no earlier than t_d is to make the recovery trajectories smoother than those without a maintenance period [15]. Unlike the safe deadline t_d, a later t_m might make the recovery problem harder to solve since it requires the recovery control to keep the system in the target set for a longer time. To obtain a reasonable estimate for t_m, we consider using an approach which is similar to the real-time monitoring technique described in [65]. We require that all of the reachable state at the time from $t_d + 1$ to $t_d + N$ which is t_m should be able to be kept

in the target set. Hence, the reachable set overapproximation

$$X_j = A^j \boldsymbol{x}_0 + \sum_{i=0}^{j-1} A^i \, B \, \boldsymbol{u}_{i+N_r} + \sum_{i=0}^{j-1} A^i \, \boldsymbol{v}_{i+N_r}$$

for all $j = D+1, \ldots, N$ can be kept in the target set with at least one instantiation sequence of $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_N$. In order to verify it, we may check whether X_j inevitably exceeds the target set X_T or not. That is, the containment $X_j \subseteq X_T$ is inevitably violated. To do so, we check the emptiness of the intersection between the control envelope

$$E_j = \left\{ \sum_{i=0}^{j-1} A^i \, B \, \boldsymbol{u}_{i+N_r} \mid \boldsymbol{u}_{N_r}, \ldots, \boldsymbol{u}_{N_r+j-1} \in U \right\}$$

which contains all possible control effect at the j-th step, and the Minkowski difference,

$$D_j = X_T \ominus \left\{ A^j \boldsymbol{x}_0 + \sum_{i=0}^{j-1} A^i \, \boldsymbol{v}_{i+N_r} \mid \boldsymbol{v}_{N_r}, \ldots, \boldsymbol{v}_{N_r+j-1} \in V \right\}$$

which is a constrained safe set by the accumulation of uncertainty. If the intersection is empty, then there is no feasible control of the length j to keep the system safe, otherwise there exists at least one.

Lemma 4.3.1 ([65]). *If the intersection $E_j \cap D_j$ is* nonempty *then there exists a control sequence to keep X_j in the target set.*

Since a Minkowski difference is often hard to compute, we resort to a more efficient but also conservative way to verify the emptiness of the intersection. We use the interval hull of the second operand in the Minkowski difference, and the result $\hat{D}_j$ is an underapproximation of the actual difference. Therefore, we may conservatively check the emptiness of $E_j \cap \hat{D}_j$ to verify whether a recovery control can maintain the system in the target set. We may do so using linear programming. We repeatedly check the intersection for

$j = D + 1, \ldots$ until we meet an empty intersection or j reaches a given maximum bound $j_{\max}$. In the first case, t_m is set to be $t_r + j - 1$, and in the second case we set $t_m = t_r + j_{\max}$.

Remark 4.3.1. *Solving the recovery control problem as a whole is difficult even the dynamics is linear, since we need to find all of the parameters such as control sequence length, safe deadline and control inputs by solving a single optimization problem in order to strictly keep their dependencies to obtain the optimal solution. To avoid the high computational cost, we decompose the problem to first estimate the deadline t_d and the "latest" maintainable time t_m, and then use them in the LP modeling of the recovery control problem. Although our method is not complete, i.e., it may not find a feasible recovery control in some cases, but the result is always sound, that is the control sequence found by our method is guaranteed to steer the system to the target set at the time t_d and maintain it there till the time t_m.*

4.4. Evaluation

We implement a prototype simulation tool in Python (in Appendix) to evaluate the effectiveness of our LQR-based real-time recovery, based on 5 CPS simulators under 3 attack scenarios. Then, we also compared the computational overhead between linear programming recovery, LQR-based recovery, and LQR-based recovery with ADMM algorithm.

4.4.1. Simulation Settings

Simulation Scenarios

We consider the following CPS models: vehicle turning, series RLC circuit, DC motor position, aircraft pitch, and quadrotor. The LTI models for these systems are obtained by linearization and discretization. All these simulators are LTI models, which are representative and used in both security and control works, such as [4], [16], [66]–[71].

Vehicle Turning. The following ODE models the turning of a vehicle, which changes the

speed difference between two wheels to steer [4]. The state x denotes the speed difference between two wheels, and control input u is the voltage difference applied to motors controlling the two wheels.

$$\dot{x} = -\frac{25}{3}x + 5u$$

Series RLC Circuit. A basic RLC circuit contains a resistor, an inductor, and a capacitor connected in series. An adjustable voltage source is connected to form a closed loop circuit. The system dynamics are modeled by the right equation such that state x_1 denotes the voltage across the capacitor and state x_2 denotes the electric current in the loop. The control input u is considered the voltage of the voltage source.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{C} \\ -\frac{1}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} u$$

DC Motor Position. The following ODE models the rotary position of a DC motor. The state x_1 denotes the rotation angle, x_2 is the rotary angular velocity, and x_3 is the armature current. The control input u is considered the voltage applied to the motor.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{b}{J} & \frac{K}{J} \\ 0 & -\frac{K}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L} \end{bmatrix} u$$

Aircraft Pitch. The following ODE describes the longitudinal dynamics of motion for the aircraft. The x_1 denotes the angle of attack, x_2 denotes the pitch rate, and x_3 denotes

the pitch angle. The control input u is the elevator deflection angle.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -0.313 & 56.7 & 0 \\ -0.0139 & -0.426 & 0 \\ 0 & 56.7 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0.232 \\ 0.0203 \\ 0 \end{bmatrix} u$$

Quadrotor. A linear quadrotor model is described in [71]. The system consists of 12 state variables: $[x, y, z]^T$ and $[\phi, \theta, \psi]^T$ are linear and angular positions of the quadrotor in the earth frame. $[u, v, w]^T$ and $[p, q, r]^T$ are the linear and angular velocities in the body frame. The control input is denoted by u, in which f_t is total thrust and $[\tau_x, \tau_y, \tau_z]^T$ are the control torques caused by differences of rotor speeds.

Other simulation settings are listed in table 4. For example, we use a PI controller ($K_P = 5, K_I = 5$) to update the control input u of series RLC circuit every 0.02 seconds. The attacks start at time $t_a = 3s$, and are detected at time $t_f = 4.3s$. The system is safe when the capacitor voltage x_1 is in [0,7], and the target set of this scenario is [2.9, 3.1]. The voltage of source can be adjusted between [-15, 15]. We concern the first state, so the first diagonal element of Q is Q_i, others are all 1. We choose control input cost as 0.1.

Table 4: Simulation Scenarios. (Time unit: second) Cyber-physical System Properties - δ: control stepsize, X_S: safe state set, U: control input limits, PID: PID parameters of original controller. Recovery-related Parameters - t_a: attack launched time, t_f: attack detected time, X_T: target state set, Q_i: state cost corresponding to the i^{th} state (other costs set to 1), R: control input cost.

Simulation Scenarios	Cyber-physical System Properties				Recovery-related Parameters				
	δ	X_S	U	PID	t_a	t_f	X_T	Q_i	R
Vehicle Turning	0.02	$x \in [-2.7, 2.7]$	$[-5, 5]$	0.5, 7, 0	4	4.5	$x \in [0.9, 1.1]$	$Q_1 = 10$	5
Series RLC Circuit	0.02	$x_1 \in [0, 7]$	$[-15, 15]$	5, 5, 0	3	4.3	$x_1 \in [2.9, 3.1]$	$Q_1 = 1$	0.1
DC Motor Position	0.1	$x_1 \in [-4, 4]$	$[-20, 20]$	11, 0, 5	6	9.9	$x_1 \in [-1.67, -1.47]$	$Q_1 = 10$	0.01
Aircraft Pitch	0.02	$x_3 \in [0, 2]$	$[-20, 20]$	14, 0.8, 5.7	3	4.3	$x_3 \in [0.68, 0.72]$	$Q_3 = 1$	1
Quadrotor	0.02	$z \in [-1, 8]$	$[-50, 50]$	0.1, 0, 0.6	12	13.1	$z \in [3.9, 4.1]$	$Q_9 = 1$	1

Attack Scenarios

We consider three attack scenarios mentioned in section 2.5, i.e. bias attacks, replay attacks and delay attacks. These attack scenarios are combined with each simulation scenario, which contributes to fifteen situations in total. We list the simulation parameters in table 5. For each simulation scenario, we set up an attack parameter.

Table 5: Attack Scenarios. Bias attack: add a certain value to or subtract it from sensor data. Replay attack: send historical data from a certain time interval. Delay attack: delay data sent to the controller for a certain time.

Attack	Vehicle Turning	Series RLC Circuit	DC Motor Position	Aircraft Pitch	Quadrotor
Bias	$x - 1.5$	$x_1 - 2.5$	$x_1 + 2$	$x_3 + 0.3$	$z - 2$
Replay	$[0s, 6s]$	$[0s, 5s]$	$[0s, 6s]$	$[1s, 2s]$	$[3s, 5s]$
Delay	1s	1s	1s	1s	1s

4.4.2. Baselines

We compare our proposed method with three baselines:

No recovery: the system is attacked during running, and there is no recovery method available. The sensor attack takes effect constantly, and the system state may reach the unsafe set, which causes catastrophic consequences.

Non-real-time recovery: the system performs the recovery method in [4] after a sensor attack is detected. This method cannot guarantee the system is recovered before a deadline.

Linear-programming (LP) recovery: This method is formulated as a linear programming problem, i.e., the objective function is linear.

LQR-based recovery: the system run under the proposed real-time recovery method. This paper formulates the recovery method as an LQR-based optimization problem.

4.4.3. Simulation Results

We compared our method with baselines from recovery effect and recovery overhead.

Recovery Effect

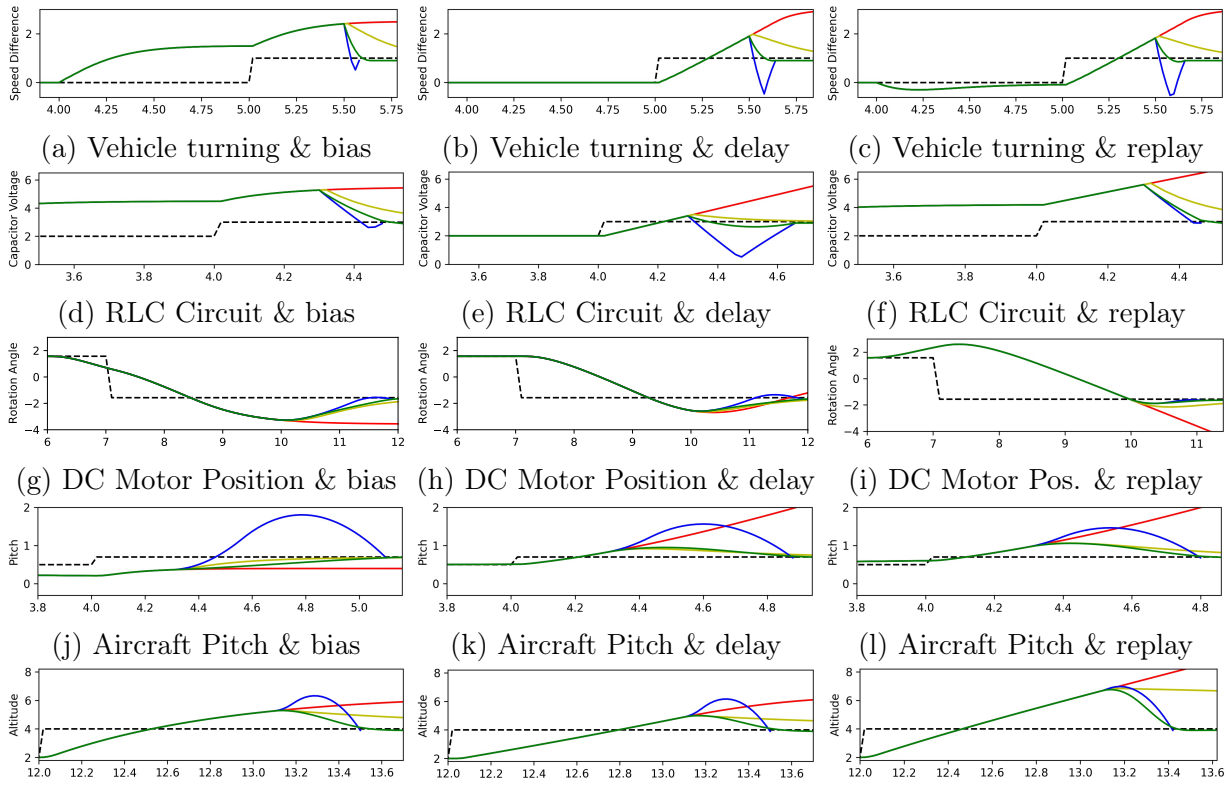
We plot the actual system states in Figure 19, which demonstrates the recovery effect against three types of sensor attacks in five CPS simulations. The following observations are obtained from these figures.

Recovery mechanism is needed during sensor attacks. The red lines represent the baseline without recovery. The results in Figure 19(f)(i)(k)(l)(o) shows the system states reach the unsafe set, which causes catastrophic consequences in CPS system. This indicates that a recovery mechanism is needed to secure the CPS in presence of sensor attacks.

Non-real-time recovery cannot steer system state back to normal within a deadline. The yellow lines show the state recovered using non-real-time recovery method. From Figure 19(a)(b)(c)(d) (f)(m)(n)(o), the recovery processes take a long time, and they failed to pull the system states back to target set before the deadline. There is no guarantee on deadline and safety.

LP recovery may make the system oscillate before the deadline. The blue lines are powered by LP recovery method. They can recover the system within deadline, because this method formulates the safety deadline as a constraint. However, the recovery trajectories are circuitous, shown in Figure 19(a)(b)(c)(e)(j)(k)(l)(m)(n), because of the linear objective function.

LQR-based recovery can recover systems smoothly within the deadline. The Green lines represent the recovery process of our LQR-based method. The proposed method can recover the system within the deadline, and the recovery trajectories are straightfor-



(a) Vehicle turning & bias (b) Vehicle turning & delay (c) Vehicle turning & replay

(d) RLC Circuit & bias (e) RLC Circuit & delay (f) RLC Circuit & replay

(g) DC Motor Position & bias (h) DC Motor Position & delay (i) DC Motor Pos. & replay

(j) Aircraft Pitch & bias (k) Aircraft Pitch & delay (l) Aircraft Pitch & replay

(m) Quadrotor & bias attack (n) Quadrotor & delay attack (o) Quadrotor & replay attack

Figure 19: Comparison of the system executions under three situations for each attack scenario. RED = No recovery. YELLOW = Non-real-time recovery (previous work [4]). BLUE = Linear-Programming recovery (previous work [15]). GREEN = LQR-based recovery (our proposal). Dotted Black Line = Reference state.

ward. The quadratic objective function adds penalty on states and control inputs, thus, play a key role in smooth trajectories. Moreover, our method makes the state maintain within the target set for a while, which is helpful to return the system to original controller or buy time to restart the original controller.

Overhead Analysis

We also analyze the time cost under three conditions. The time cost includes state estimate reconstruction time, deadline estimation time, problem formulation time, and solving time. Note that the solving time is measured from the time *solve* function is called to the time the result is returned. For linear programming recovery, we use PyGLPK solver

with simplex method to generate the control input signals. For LQR-based recovery, we first use CVXPY with ECOS solver [72], and then use OSQP solver **osqp**, a standard implementation of ADMM algorithm, to accelerate the solving process. There are following observations from Table 6.

We considered the larger overhead of LQR-based recovery in the framework. The first two rows show the time cost of LP recovery method, and all recovery can be done in one control stepsize. This work assumes that the recovery sequence can be applied immediately after an attack is detected, but this may not be true for complex system. The middle two rows are the time cost of our LQR-based recovery without ADMM algorithm, and the cost is more than one stepsize for most scenarios. A larger overhead is because we use a quadratic objective function instead of a linear one. However, we considered a small computational time $(t_r - t_f)$ to get the recovery sequence in our framework, shown as Figure 14, and apply our recovery control sequence at time t_r, which makes our method practical.

ADMM algorithm accelerates our LQR-based recovery effectively. The last two rows are the time cost of our method using ADMM algorithm. Compared to the middle two rows, the result shows the overhead of ADMM algorithm is smaller for all benchmarks and can effectively accelerate the computing. This is because the ADMM algorithm can split a global optimization problem into two small subproblems, and solve them iteratively.

The speedup of ADMM is significant for small systems. The benchmarks with fewer state variables, such as vehicle turning, RLC circuit, and DC motor position, run much faster and are close to the overhead of linear programming recovery. The most computational intensive operation in OSQP solver is factorizing the coefficient matrices, which has polynomial time in the matrix dimensions. Thus, ADMM performs especially well for small systems.

Table 6: time cost of computing the recovery controls. the time unit is millisecond. legends: T_{LP}: time cost of Linear-Programming (LP) method, $\%_{LP}$: ratio of T_{LP} to control stepsize, T_{solver}: time cost of LQR-based method using ECOS solver, $\%_{solver}$: ratio of T_{solver} to control stepsize, T_{ADMM}: time cost of LQR-based method using ADMM algorithm with OSQP solver, $\%_{ADMM}$: ratio of T_{ADMM} to control stepsize

	Vehicle turning			RLC Circuit			DC Motor Position		
	bias	delay	replay	bias	delay	replay	bias	delay	replay
T_{LP}	0.83	1.04	1.03	1.31	2.51	1.17	3.92	3.48	1.80
$\%_{LP}$	4.15%	5.20%	5.15%	6.55%	12.55%	5.85%	3.92%	3.48%	1.80%
T_{solver}	24.05	24.54	26.80	27.10	35.21	22.82	82.78	67.92	81.19
$\%_{solver}$	120.25%	122.70%	134.00%	135.50%	176.05%	114.10%	82.78%	67.92%	81.19%
T_{ADMM}	1.69	1.97	1.85	2.09	3.82	2.00	4.48	4.32	3.20
$\%_{ADMM}$	8.45%	9.85%	9.25%	10.45%	19.10%	10.00%	4.48%	4.32%	3.20%
	Aircraft Pitch			Quadrotor					
	bias	delay	replay	bias	delay	replay			
T_{LP}	10.79	6.10	4.78	10.91	11.34	8.10			
$\%_{LP}$	53.95%	30.50%	23.90%	54.55%	56.70%	40.50%			
T_{solver}	81.10	62.02	73.42	44.15	57.09	80.28			
$\%_{solver}$	405.50%	310.10%	367.10%	220.75%	285.45%	401.40%			
T_{ADMM}	27.37	22.80	19.76	38.52	25.18	35.17			
$\%_{ADMM}$	136.85%	114.00%	98.80%	192.60%	125.90%	175.85%			

The Impact of Load Disturbance

We take the quadrotor simulator under bias attacks as an example to demonstrate the impact of load disturbance. In this experiment, the safe set is set to be $z \in [-1, 5.7]$, and other parameters remain the same as those in Table 4. In the Eq. (2.1), we consider that the uncertainty $\boldsymbol{v}(t)$ includes both sensing noise and load disturbance, and is bounded by v_{max}. We change the load disturbance by varying v_{max} in this experiment. We perform multiple simulations with increasing v_{max}, and record the safe deadline and maintainable time in Table 7. In the table, v_{max} denotes the maximum uncertainty/load disturbance that at each control step; D means the recovery length, i.e., the number of control steps from t_r to t_d; N denotes total recovery control length, i.e., the number of control steps from t_r to t_m; the safe deadline t_d and maintainable time t_m are estimated using our deadline estimator illustrated in the Section 4.3.3. We also plot the quadrotor's vertical position z in Figure 20. In the figure, the blue solid line is the real states; the orange solid line

is the desired recovery states predicted by the recovery controller; the red solid line indicates the boundary of target state set. In the Figure 20a, the load disturbance begins from time t_a, and in the Figure 20b, the load disturbance begins from time t_r. Based on the results, we have several observations as follows.

Larger load disturbance leads to earlier safe deadline and earlier maintainable time. The load disturbance exists at each control step and is bounded by v_{max}. If v_{max} is larger, the reachable set overapproximation X_k is bigger according to the Eq. (4.13). Then, the reachable set is more likely to intersect with the unsafe state set, which leads to a shorter recovery length N. Likewise, the constrained safe set with the accumulation of uncertainty, i.e., D_j, is smaller, which results in an earlier maintainable time t_m.

Larger total recovery control length N requires more computational overhead. The state constraints, i.e., Eq. (4.2b) and Eq. (4.2c), cover all states in the recovery and maintenance period. Thus, a larger N means more variables in the optimization problem, which comes with more computational overhead. When v_{max} is smaller,the system state can be maintained within the target set for a longer time once recovered into the set. We may choose a smaller maintenance length to reduce the computational overhead according to different application needs.

The system can still steer the system state back to the target state set and maintain it in the set in presence of load disturbance. In the Figure 20a, the load disturbance starts from time t_a. At time t_r, the reconstructed state is slightly different from the real state because of the load disturbance. From t_d to t_m, the real state (marked in blue solid line) is within the target state set, although there exists difference between predicted states and the real states. This is because we considered the accumulation of load disturbance in our state constraint in the Eq. (4.2c). In the Figure 20b, the load disturbance starts from time t_r. At time t_r, the reconstructed state is almost the same as the

Table 7: The impact of load disturbance on quadrotor simulator under bias attack. legends: v_{max}: the maximum load disturbance in each control step, D: recovery length ($D = t_d - t_r$), N: total recovery control length ($N = t_m - t_r$), $T_{solving}$: the solving time of OSQP solver in millisecond.

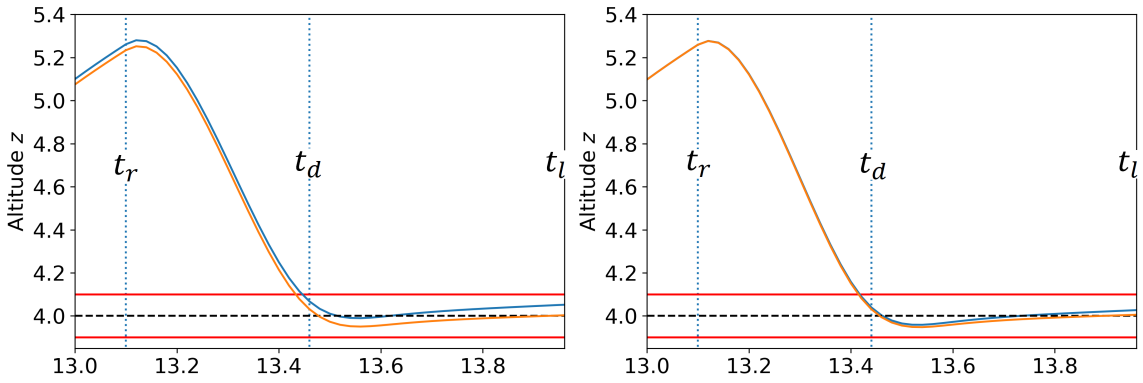
$v_{max}(\times 10^{-4})$	1	2	3	4	5	6	7	8
D	18	18	18	18	17	17	17	17
N	209	122	82	59	43	31	23	16
$T_{solving}$	251	130	103	61	57	40	23	N/A

real state. By comparing the two figures, we can see that the real state at time t_m is closer to the reference state in Figure 20b than the Figure 20a. The reason is that the load disturbance in Figure 20b starts later than that in Figure 20a, and thus affects the recovery for a shorter time. Further, as long as the load disturbance can be bounded to a certain range, the system can still be recovered by our method.

Our recovery method is sound but not complete. The right most column in Table 7 shows a case that our optimization problem is infeasible, when the total recovery control length $N = 16$ is less than the recovery length $D = 17$. This is because $X_T \ominus A^i \mathcal{I}_R$ in the Eq. (4.2c) becomes empty because of the accumulation of load disturbance. Our method cannot guarantee finding a recovery control sequence to steer the system back to target state set under such a large load disturbance. Further, if our optimization is feasible, then the solution can guarantee that the recovery process will be successful.

4.5. Conclusion

Two fundamental elements for the operation of safe and resilient cyber-physical systems are attack detection and recovery. While the vast majority of existing works focus on attack detection, while little attention has been paid to attack-recovery. In this chapter, we study this problem and novel techniques on real-time recovery for securing CPS. These techniques include i) an LQR based recovery control calculator that can smoothly and safely recover the system before a safety deadline and maintain the recovered system for



(a) load disturbance starts from t_a (b) load disturbance starts from t_r

Figure 20: The recovery of vertical position z of quadrotor under bias attacks with $v_{max} = 5 \times 10^{-4}$. The solid blue line represents real system states; the orange solid line shows the desired recovery states predicted by recovery controller; black dashed line is the reference or target states; red solid line marks the boundary of target state set.

a certain amount of time, ii) a checkpointer that keeps enough trustworthy data for the recovery computation, iii) a state reconstructor that rebuilds the current system state, and iv) a deadline estimator that uses reachability analysis to conservatively compute a safety deadline. Using multiple CPS simulators, we show that our methods can recover the attacked-system in a timely and safe manner, outperforming previous related work in terms of smoothness and maintainability.

CHAPTER 5

Real-time Sensor-attack Recovery in Complex CPSs

In this chapter, we propose an online recovery system for nonlinear CPSs that strikes a balance between efficiency and accuracy. Besides, the proposed method also leverages uncorrupted sensor data to enhance recovery performance.

5.1. System Overview of Recovery System

This section summarizes the overview of system design, which will be discussed further in Section 5.1.4, and give the problem statement and assumptions.

5.1.1. Problem Statement

We consider a nonlinear CPS described in Section 2.4 under the sensor attacks shown in Section 2.5. The physical states deviate from the reference states under the influence of such attacks. A recovery controller is triggered after the attack diagnosis identifies the compromised sensors. The problem is to design a recovery controller that can smoothly guide the nonlinear system's physical states to a target set $\mathcal{T}$ before they reach the unsafe state set $\mathcal{F}$. Note that it should leverage uncompromised sensor data during recovery if possible.

5.1.2. Recovery Controller Overview

Our real-time data predictive recovery system is shown in Figure 21. The target CPS is in the bottom part of the figure, and our system extends the original system to secure the system under sensor attacks. The paper focuses on the recovery controller shown in the blue shaded box, which includes (i) adaptive recovery sequence generator, (ii) model adaptor, (iii) state predictor, and (iv) time oracle.

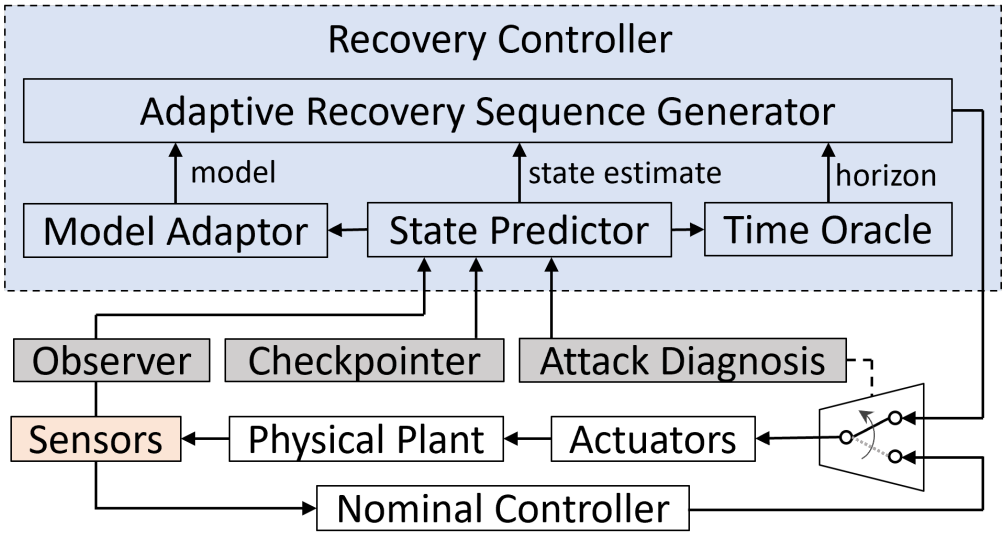


Figure 21: Real-time Data Predictive Recovery Overview

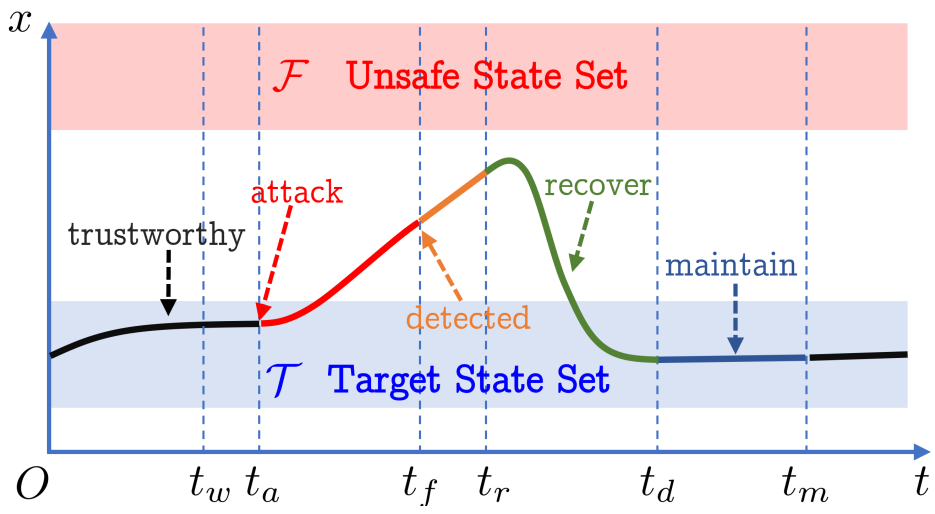


Figure 22: Illustration of Recovery Timeline

When does the recovery controller take effect? The system runs in two possible modes - normal mode and recovery mode. Figure 22 illustrates the timeline of how our recovery system works. In normal mode, the system runs the original nominal controller, and the system states follow the target states (also known as reference states) without attacks. At an unknown time t_a, a sensor attack is launched, so the actual system states begin deviating from the target states. There is a detection delay ($t_f - t_a$) needed for the attack diagnosis to identify the attack. At the time t_f, the detector raises an alert indicating the attack, and the system switches from normal to recovery mode. In recovery mode, our recovery controller is activated. Note that the recovery controller is designed to handle attacked sensors, only running after detecting attacks.

How does the recovery controller work? The adaptive recovery sequence generator draws upon the idea of model predictive control. It formulates the recovery problem as an optimization problem with time and safety constraints, but only implements the first several recovery control inputs and then optimizes again, repeatedly. Each optimization requires the help of supporting components (Section 5.3) for updating the formulation: the model adaptor provides the linear model working on current states, reducing the modeling error; the state predictor provides an accurate initial state for recovery, relieving the uncertainty accumulation; the time oracle adjusts the optimization horizon and time constraints, making a recovery before the system states become unsafe. Note that, through nonlinear reachability analysis, the initial state for recovery is obtained from a trustworthy state provided by checkpointer, and no measurements from attacked sensors are used.

5.1.3. Assumptions

We list our assumptions in this subsection. Note that the assumptions about checkpointer, attack diagnosis, and target set are fulfilled by the previous work, so they are outside the scope of this paper.

We assume that the system operates a closed-loop control process, and the plant can be nonlinear. Moreover, the system is perturbed by noise. This assumption is detailed in Section 2.4. Also, sensor attacks make the nominal controller generate inappropriate control inputs that lead to the deviation from desired physical states, detailed in Section 2.5.

For the CPS components, we assume that there is a checkpointer that can record historical data, including state estimations and control inputs. The data cover at least one time step before the attack occurs. Such a checkpointer has been described in detail in [4], [15], [33], [73]. Under the assumption of fully observable states at the end of Section 2.4, this checkpointer caches the physical state $\boldsymbol{x}_t$ at every time step. In a general case, it records sensor measurement $\boldsymbol{y}$.

Also, we assume a detection/diagnosis module that is able to correctly locate the corrupted component before the system is driven into an undesirable unsafe state. In our recovery system, it can identify which sensors are compromised. The attack diagnosis works give several solutions, such as the sensor attack detectors proposed by [16], [74]. Note our method also applies when all sensors are compromised. Uncompromised sensors, if any, help improve recovery performance.

Furthermore, we assume that the recovery target state set is within the control invariant set of the original nominal controller, so that the nominal controller can take over when it is available after recovery. The control invariant set can be computed through [75]. We assume that the recovery control sequence can be implemented to actuators, or we need a redundant actuator in the systems.

5.1.4. Data Predictive Recovery Algorithm

The blue shaded box in Figure 21 illustrates the data predictive recovery controller. The controller consists of four components that cooperate with each other. The adaptive recov-

ery sequence generator (Section 5.2), the core component, generates the recovery control sequence and guides the physical state of the CPS back to a target state. It formulates and solves an optimization problem opt_i with safety and time constraints in every T control steps. During recovery, its input, an initial reachable set X_0 and a horizon N, comes from the state predictor (Section 5.3.1) and the time oracle (Section 5.3.3), respectively. In addition, the model adapter (Section 5.3.2) updates the linear approximation of the nonlinear dynamics before formulating each optimization problem.

Algorithm 2: Extended Data Predictive Recovery

Input: historical data from checkpointer, attack detection/ diagnosis result, nonlinear system dynamics

Output: real-time control signal $\boldsymbol{u}_t$ for $0 \leq t \leq t_m$

```
while t ≥ 0 do
    if t < t_f then
        Run with normal control
    else if t = t_f then
        t_d, t_m ← time oracle                          ▷ deadline computing
        X_{t_r} ← state predictor                       ▷ state reconstruction
        sys_t ← model adaptor                           ▷ model adaptation
        opt_0(sys_t, X_{t_r}, t_m − t_r)                ▷ first optimization
        u_t, ..., u_{t_r−δ} ← u_t                       ▷ cached from nominal control
        Run with u_t                                    ▷ implement control input
    else if t_f < t < t_r then
        Run with u_{t_f+δ}, ..., u_{t_r−δ}              ▷ cached control
    else if t_r ≤ t ≤ t_m then
        if t = t_r + KTδ with integer K ≥ 0 then
            u_t, ..., u_{t+(T−1)δ} ← opt_K              ▷ optimization result
            X_{t+Tδ} ← state predictor                  ▷ reachability
            sys_t ← model adaptor                       ▷ model adapt.
            opt_{K+1}(sys_t, X_{t+Tδ}, t_m − (t + Tδ))
        Run with u_t                                    ▷ implement control input
```

Algorithm 1 shows the real-time attack-recovery procedure, and the critical time is reflected in the timeline (Figure 22):

(i) Lines 2-3: Before the attack is detected at t_f, the system runs the original nominal controller in normal mode. The state estimate may be corrupted by sensor attacks, while the actual physical state may deviate from the desired state.

(ii) Lines 4-10: At detection time t_f, the system switches from normal mode to recovery mode, where the recovery controller takes over. Since the state estimate is compromised, the state predictor (Section 5.3.1) performs a non-linear reachability analysis and reconstructs the state estimate reachable set at t_r from a trustworthy state at time t_w provided by checkpointer, meanwhile, it uses good sensor data from t_w to t_f to improve prediction accuracy. Also, the algorithm calculates a safety deadline t_d and a maintainable time t_m using the time oracle (Section 5.3.3). Then, the model adaptor (Section 5.3.2) computes an updated linear approximation around current states and control input from nonlinear dynamics. Based on them, the adaptive recovery sequence generator (Section 5.2) formulates the first optimization problem opt_0 and begins to solve it. Since the result of opt_0 is not ready, it prepares the control sequence for the preparation period $[t_f, t_r)$ using the last cached control input $\boldsymbol{u}_t$ from the nominal controller.

(iii) Lines 11-12: During period (t_f, t_r), the system runs with the prepared control sequence, meanwhile solving opt_0.

(iv) Lines 13-19: From t_r to t_m, for every T control step, the state predictor predicts the initial reachable set of the next optimization problem, i.e., $X_{t+T\delta}$. In this process, intact sensor data, if any, can improve prediction accuracy. Also, the model adaptor updates the linear approximation of dynamics. The recovery sequence generator starts to formulate and solve a new optimization problem opt_{K+1}. Consequently, the recovery control sequence from opt_K will be ready in T control steps, but only the first T control input will be implemented. Note that finding a solution within δ is not necessary, since optimization runs every T control step.

5.2. Adaptive Recovery Sequence Generator

Our core component, *adaptive recovery sequence generator*, aims to recover the physical states into a target set $\mathcal{T}$ before they touch an unsafe set $\mathcal{F}$. We consider continuous-time and nonlinear systems in the form of Equation (2.2). To efficiently compute recovery control inputs, we perform local linearization and discretization online using the model adaptor (Section 5.3.2), and encode the linearized-discretized model into the optimization problem. We use the subscript to represent variables at a certain time hereinafter. For example, $\boldsymbol{u}_t$ represents the control input at time t. Note that our approach also applies if we consider a discrete model from the beginning.

5.2.1. Basic Data Predictive Recovery Formulation

After the attack diagnosis identifies the corrupted sensors, the generator formulates the recovery problem as a quadratic programming problem with dynamics, time, and safety constraints. By solving this problem, we get a recovery control sequence, but we only implement the first control input and then repeat this process in subsequent steps. Each optimization problem updates the parameters, including the model from the model adaptor, the initial state of recovery from the state predictor, and the deadline from the time oracle.

The objective of these quadratic programming problems guides the state towards reference state fast and smoothly:

$$\begin{aligned} J = &(\boldsymbol{x}_N - \boldsymbol{x}^*)^T \mathbf{Q}_N (\boldsymbol{x}_N - \boldsymbol{x}^*) \\ &+ \sum_{k=0}^{N-1} (\boldsymbol{x}_k - \boldsymbol{x}^*)^T \mathbf{Q} (\boldsymbol{x}_k - \boldsymbol{x}^*) + \boldsymbol{u}_k^T \mathbf{R} \boldsymbol{u}_k \end{aligned} \tag{5.1}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the system state and control input variables at i^{th} control step; $\boldsymbol{x}^*$ is

the reference state; $\mathbf{Q}, \mathbf{Q}_N \in \mathbb{R}^{m\times m}, \mathbf{R} \in \mathbb{R}^{n\times n}$ are semi-definite symmetric matrices that represent the state, the final state, and the control input cost weight; $N = D + M$ is the optimization horizon length, where recovery time $D = (t_d - t)/\delta$ and maintainable time $M = (t_m - t)/\delta$ at the time t. A fast recovery steers states to the reference state fast, and tends to achieve a small state penalty, represented by the first two terms; A smooth recovery uses small control effort, and tends to achieve a small control penalty, represented by the third term.

The constraints are formulated as follows.

$$\boldsymbol{x}_{i+1} = \mathbf{A}\boldsymbol{x}_i + \mathbf{B}\boldsymbol{u}_i + \boldsymbol{c} \qquad \forall i \tag{5.2a}$$

$$\boldsymbol{u}_i \in \mathcal{U} \qquad \forall i \tag{5.2b}$$

$$X_i \cap \mathcal{F} = \emptyset \qquad \forall i \in [0, M] \tag{5.2c}$$

$$\boldsymbol{x}_i \in \mathcal{T} \qquad \forall i \in [D, M] \tag{5.2d}$$

Notice that Eq. (5.2a) is the discrete linearized dynamic constraint provided by the model adapter as in Equation (5.6), with parameters $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{c}$. It is used to predict the plant's future evolution; Eq. (5.2b) limits our control inputs according to the actuator's capacity; Eq. (5.2c) ensures that all recovery states are safe; Eq. (5.2d) makes sure that the system state goes back into the target state set before the safe deadline t_d and maintains it in the set for the rest of the optimization horizon until t_m. Note that the optimization horizon is receding over time, thus the computational overhead is also decreasing during the recovery.

However, there are some limitations in the standard data predictive recovery formulation: (i) we did not consider the computational overhead of optimization problems. It usually takes more than one control step to solve this problem. Thus, this recovery method may not be applied to complex systems because the computational overhead is large. (ii) we

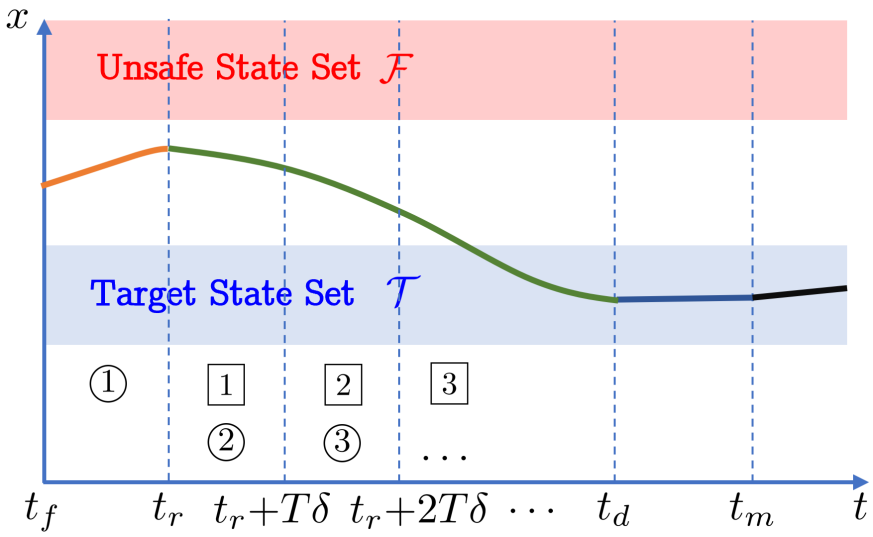


Figure 23: Illustration of Extended Model Predictive Recovery. ⓘ denotes solving the i^{th} optimization problem, and $\boxed{i}$ denotes implementing the recovery control inputs computed from i^{th} optimization problem.

did not consider the uncertainty in the constraints in Eq. (5.2), such as linearization error and estimation noises due to overapproximated reachable sets. Thus, some constraints are not guaranteed to be met in real applications.

5.2.2. Extended Data Predictive Recovery Formulation

To overcome the limitations above, we extend the basic formulation by considering computational overhead and uncertainties, shown in Figure 23.

Considering that the computational overhead may be greater than the control sampling time, the recovery sequence generator optimizes every T control steps instead of every step. Note that the T control steps can cover computational overhead and can be determined in advance. At time t_f, the attack diagnosis identifies the attack, and the generator begins to formulate and solve the first optimization problem. Before time $t_r = t_f + T\delta$, the generator can obtain the result of the first optimization problem. From time t_r, while it starts to implement the first T control inputs computed by the first optimization problem, it begins to formulate and solve the second optimization problem. Similarly, at time

$t_r + T\delta$, it implements the first T control inputs computed by the second optimization problem and begins to formulate the third one. The recovery sequence generator repeats this process until time t_m in such a pipeline manner.

To formulate each optimization problem, the generator requires (i) the initial state x_0 and the uncertainty interval I_0 provided by the state predictor. Since x_0 contains good sensor data, it can be seen as feedback, helping alleviate the accumulation of uncertainty in uncompromised state dimensions. (ii) the system model provided by the model adaptor, i.e. $\mathbf{A}$, $\mathbf{B}$, and $\boldsymbol{c}$ matrix. Since the linear model works well in a small state range, the model adaptor keeps linearizing the nonlinear dynamics around current states. The time-variant model helps to reduce modeling errors. (iii) safe deadline t_d provided by the time oracle. The time oracle performs nonlinear reachability analysis to compute a conservative deadline by which the system may become unsafe. The deadline helps to guarantee the safety of CPS. The detailed design of these supporting components is described in Section 5.3. The constraints of the optimization problem are reformulated as follows.

$$\boldsymbol{x}_0 \in X_0 = \boldsymbol{x}_0 \oplus I_0 \qquad \text{from state predictor} \tag{5.3a}$$

$$\boldsymbol{x}_{i+1} = \mathbf{A}\boldsymbol{x}_i + \mathbf{B}\boldsymbol{u}_i + \boldsymbol{c} \qquad \forall i \tag{5.3b}$$

$$\boldsymbol{u}_i \in \mathcal{U} \qquad \forall i \tag{5.3c}$$

$$\boldsymbol{x}_i \cap (\mathcal{F} \oplus \mathbf{A}^i I_0) = \emptyset \qquad \forall i \in [0, M] \tag{5.3d}$$

$$\boldsymbol{x}_i \in \mathcal{T} \ominus \mathbf{A}^i I_0 \qquad \forall i \in [D, M] \tag{5.3e}$$

where at Eq. (5.3a), x_0 is the initial state of recovery with uncertainty interval I_0 that can be obtained from the state predictor; Eq. (5.3d) and (5.3e) consider the effect of uncertainty I_0, so the unsafe set is larger and the target set is smaller than those of the basic data predictive recovery. Here, $\oplus$ is the Minkowski sum defined as $X \oplus Y = \{x +$

$y \,|\, x \in X, y \in Y\}$ for any set X and Y; $\ominus$ is the Minkowski difference such that $X \ominus Y = \bigcap_{y \in Y}\{x - y \,|\, x \in X\}$.

Furthermore, it is noted that the optimization horizon reduces T for every optimization problem. For example, the horizon of the first optimization is $(t_m - t_r)/\delta$, and the horizon of the second becomes $(t_m - t_r)/\delta - T$. The reducing horizon also leads to a reduction in the number of optimization variables in optimization problems, so computational resources are saved without hurting recovery performance. Note that we can still guarantee safe and time constraints, although the optimization horizon is reduced.

5.3. Supportive Components

The formulation of optimization problems requires the parameters of the supporting components, including the state predictor, the model adaptor, and the time oracle. This subsection introduces each component in detail.

5.3.1. State Predictor

The state predictor performs nonlinear reachability analysis to get the reachable set of states based on historical data.

Input. It relies on (i) attack diagnosis result. The result indicates which sensors are compromised and which sensors are intact. (ii) historical states and control inputs from checkpoint. The state at time t_w is the trustworthy state, and is not affected by sensor attacks. The intact states during sensor attack can also be used to relieve uncertainty accumulation. The control inputs are used for reachability analysis. Since it does not use compromised sensor data as input, the result is not affected by sensor attacks.

Overview. Figure 24 illustrates the process of successive calculation of reachable states. The gray-shaded area highlights a fragment at time t. At time t, the reachable states (X'_t, marked in green) are obtained from the previous reachable analysis. Also, current state

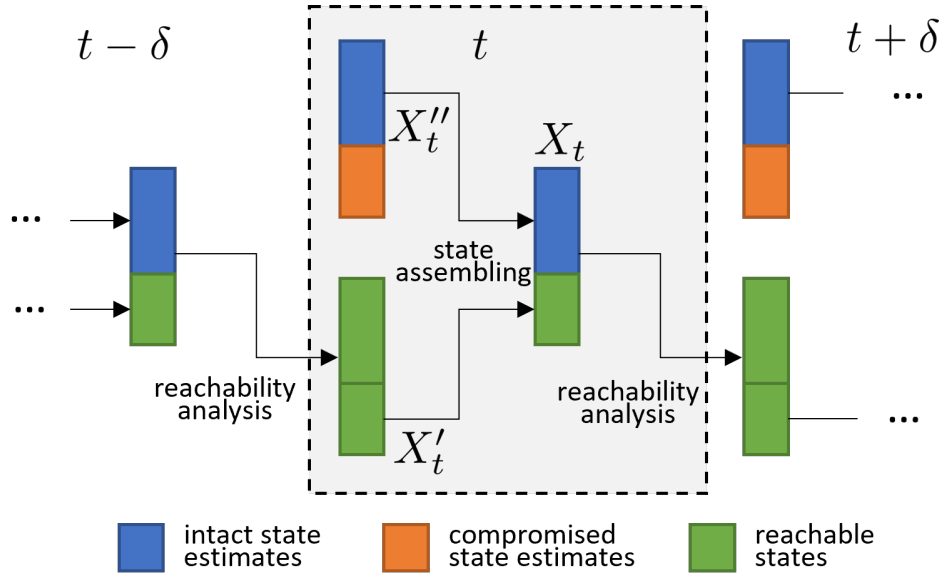


Figure 24: A fragment of successive calculation of reachable states using the state predictor

estimates are obtained from the system observer, but some state estimates are affected by sensor attacks, marked in orange. We replace those compromised ones with the corresponding reachable states, forming sensor-adjusted reachable states X_t, which other components require. From this set, we perform a nonlinear reachability analysis and get the next reachable states. The state predictor repeats the above steps to calculate the multiple-step reachable state. The sensor-adjusted reachable state X_t will be used as X_0 in equation (5.3a). In this process, the state predictor uses good sensor measurements as feedback, preventing uncertainty from exploding in the uncompromised dimension.

Reachability Analysis Given a historical state, we can use the Taylor model-based reachability computation to obtain an overapproximate estimation of the state at a later time. Taylor models are originally proposed as overapproximate representations for smooth functions [76], and are later used in verified integration of nonlinear ODEs [77] and to compute reachable set overapproximations for hybrid systems [78]–[80]. To do so, we may directly use the state-of-the-art tool, Flow* [81]. Since we only use the tool to compute an interval

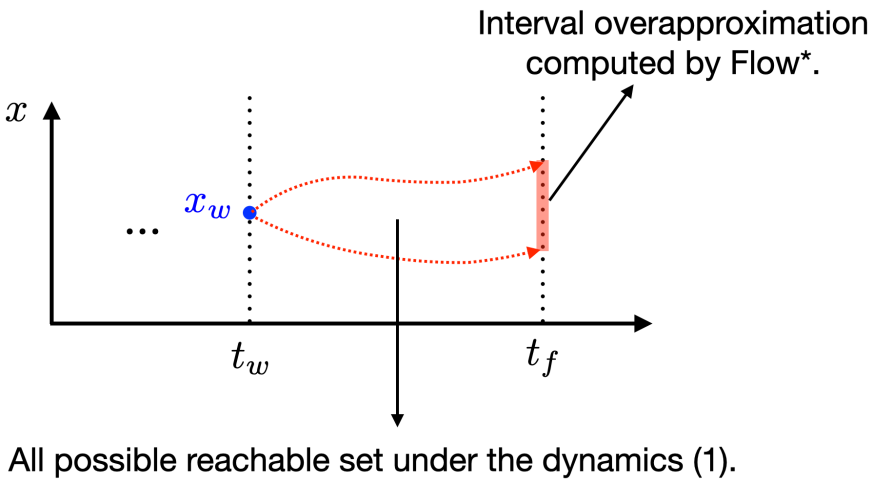


Figure 25: Illustration of the use of Flow*.

reachable set overapproximation, our framework does not need to handle Taylor models from scratch. For example, it can compute a conservative estimation of the system state at t_f when an attack is detected, which is also known as state reconstruction. It takes the ODE (2.2), the latest trustworthy state $\boldsymbol{x}_w$ as the initial state and the historical control sequence that is used from t_w to t_f, and computes an interval set (or box) X that is guaranteed to contain the system state at t_f. Figure 25 illustrates the use of Flow*. The evaluation in [79] shows that Flow* achieves good accuracy and scalability for nonlinear reachability analysis among baseline approaches VNODE-LP, dReach, and SpaceEx.

Considering Observer. As stated at the end of Section 2.4, this algorithm assumes that states are fully observable for concision, since computing state estimates is not our main contribution. However, for more general cases, the above state predictor can be extended and work with existing nonlinear observers, which compute state estimates with good sensor measurements. The observer operates in both normal and recovery modes. As shown in Figure 22, the observer uses the original measurement function $\boldsymbol{y} = h(\boldsymbol{x}) + \boldsymbol{v}$ before t_w, since all sensor measurements are reliable. In contrast, the observer dynamic should be adjusted in the presence of the attack to avoid the impact of attacks. After time t_w, some

sensor measurements could be compromised. The algorithm trims the original measurement function to $\boldsymbol{y}' = h'(\boldsymbol{x}) + \boldsymbol{v}$, excluding the compromised sensor measurements indicated by the attack diagnosis. The observer, for example the extended Kalman filter, predicts state and covariance estimates with checkpointed previous estimates and control input, and updates the estimates with good sensor measurements. In this way, the observer obtains state estimate X_t'' unaffected by attacks.

Since the observer requires a certain period for the state estimates to converge during the start phase, we need to discuss the impact of convergence time on recovery timing. For most cases, since the observer also runs in the normal mode, the trustworthy state estimate $\boldsymbol{x}_w$ calculated by the observer is converged. When the attack is detected at time t_f, the observer needs to reconstruct the state estimates $\boldsymbol{x}_f$ with the trimed measurement function from converged $\boldsymbol{x}_w$. The computational overhead of the reconstruction may enlarge the preparation period from t_f to t_r (marked in orange in Figure 22), where t_r is when the first recovery control input is implemented. Moreover, it is undeniable that there exists an extremely rare case that the system is attacked shortly after it starts operating, and the trustworthy state estimate $\boldsymbol{x}_w$ has not yet converged. In this case, the state estimate reconstruction must wait for convergence, further extending the preparation period.

Tasks. There are four tasks for the state predictor:

(i) State reconstruction. At time t_f, the system state estimates cannot reflect the actual system states because of sensor attacks. Thus, the predictor needs to reconstruct the current state reachable set X_f from a trustworthy state $\boldsymbol{x}_w$ provided by the checkpointer. This process is demonstrated in the reachability analysis above. (Line 7 of Algorithm 2)

(ii) Initial state calculation. The optimization problems formulated by the adaptive recovery sequence generator require an initial state of recovery. The state predictor can

provide sensor-adjusted reachable states as the initial state set $X_0 = \boldsymbol{x}_0 \oplus I_0$, where x_0 is the center of reachable states, and I_0 is the uncertainty interval. (Lines 7 and 17 of Algorithm 2)

(iii) Helper function. The state predictor is called by the model adaptor and the time oracle. The model adaptor needs to linearize and discretize the nonlinear system at current states, which is provided by the state predictor. The time oracle performs reachability analysis to find a safe deadline t_d, after which the system may touch the unsafe set.

(iv) Safety checker. Before implementing the recovery control input, we use the state predictor to compute the reachable states. If there is no intersection between the reachable states and unsafe set $\mathcal{F}$, the recovery is safe.

5.3.2. Model Adaptor

Optimizing using the nonlinear dynamic is time-consuming, and can hardly be solved online. To reduce time overhead, linearized models can be used to approximate the original nonlinear system. However, a linearized model from a nonlinear system only works well around the operation or equilibrium point. The state deviation from the point may cause a large modeling error. Also, the recovery effectiveness depends on the model's accuracy. Therefore, the model adaptor keeps linearizing the nonlinear system during recovery to obtain accurate linear models.

The following model adaptor has been widely used to linearize and discretize a nonlinear continuous ODE to provide the dynamics constraint for each optimization problem, which is later formalized as Eq. (5.3b). Given a continuous nonlinear system described as Eq. (2.2) and denoted as φ^c, the adaptor calculates a linear approximation from its first

order Taylor expansion around the point of interest $\bar{\boldsymbol{x}}_t$ and $\bar{\boldsymbol{u}}_t$:

$$\begin{aligned}\dot{\boldsymbol{x}}_t &= \varphi^c(\boldsymbol{x}_t, \boldsymbol{u}_t) \\ &\approx \varphi^c(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t) + \frac{\partial \varphi^c}{\partial \boldsymbol{x}_t} \mid_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t} \cdot(\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t) + \frac{\partial \varphi^c}{\partial \boldsymbol{u}_t} \mid_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t} \cdot(\boldsymbol{u}_t - \bar{\boldsymbol{u}}_t) \\ &= \mathbf{A}'\boldsymbol{x}_t + \mathbf{B}'\boldsymbol{u}_t + \boldsymbol{c}'\end{aligned} \tag{5.4}$$

where $\mathbf{A}' = \frac{\partial \varphi^c}{\partial \boldsymbol{x}_t} \mid_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t}$, $\mathbf{B}' = \frac{\partial \varphi^c}{\partial \boldsymbol{u}_t} \mid_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t}$, and $\boldsymbol{c}' = \varphi^c(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t) - \frac{\partial \varphi^c}{\partial \boldsymbol{x}_t} \mid_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t} \cdot\bar{\boldsymbol{x}}_t - \frac{\partial \varphi^c}{\partial \boldsymbol{u}_t} \mid_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t} \cdot\bar{\boldsymbol{u}}_t$. In the implementation, the point $\bar{\boldsymbol{x}}_t$ is obtained from the center of the reachable set at time t, calculated by the state predictor. The point $\bar{\boldsymbol{u}}_t$ is the last control signal cached by the checkpointer. Since discrete-time dynamics are required in optimization problems, the model adaptor discretizes the linear approximation into a discrete-time equation with a granular time step $\delta > 0$. This discretization step is a direct result of integrating the continuous time equation (5.4) and is given as follows.

$$\begin{aligned}\boldsymbol{x}_{t+\delta} &= e^{\delta \mathbf{A}'}\boldsymbol{x}_t + (\mathbf{A}')^{-1}(e^{\delta \mathbf{A}'} - \mathbf{I})\mathbf{B}'\boldsymbol{u}_t + \delta \boldsymbol{c}' \\ &\approx (\mathbf{I} + \delta \mathbf{A}')\boldsymbol{x}_t + \delta \mathbf{B}'\boldsymbol{u}_t + \delta \boldsymbol{c}'\end{aligned} \tag{5.5}$$

where $\mathbf{I}$ is the identity matrix with proper dimensions, and δ is the sampling time or the control interval. Thus, we approximate the nonlinear dynamics to the form of Equation (5.6), where $\mathbf{A} = \mathbf{I} + \delta \mathbf{A}'$, $\mathbf{B} = \delta \mathbf{B}'$, and $\boldsymbol{c} = \delta \boldsymbol{c}'$.

$$\boldsymbol{x}_{t+\delta} = \mathbf{A}\boldsymbol{x}_t + \mathbf{B}\boldsymbol{u}_t + \boldsymbol{c} \tag{5.6}$$

For example, on a simple control system $\dot{\boldsymbol{x}}_t = \boldsymbol{x}_t^2 + \boldsymbol{u}_t$ with one-dimensional $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$, we compute centers $\bar{\boldsymbol{x}}_t$ and $\bar{\boldsymbol{u}}_t$ from the state predictor and perform Taylor expansion around

this point following Eq. (5.4).

$$\begin{aligned}\dot{\boldsymbol{x}}_t &= \boldsymbol{x}_t^2 + \boldsymbol{u}_t \approx \bar{\boldsymbol{x}}_t + \bar{\boldsymbol{u}}_t + 2\bar{\boldsymbol{x}}_t \cdot (\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t) + 1 \cdot (\boldsymbol{u}_t - \bar{\boldsymbol{u}}_t) \\ &= 2\bar{\boldsymbol{x}}_t \boldsymbol{x}_t + \boldsymbol{u}_t - 2\bar{\boldsymbol{x}}_t^2 + \bar{\boldsymbol{x}}_t\end{aligned} \tag{5.7}$$

That is, $\mathbf{A}' = 2\bar{\boldsymbol{x}}_t$, $\mathbf{B}' = 1$ and $\mathbf{c}' = -2\bar{\boldsymbol{x}}_t^2 + \bar{\boldsymbol{x}}_t$. the discretization can be done by calling Eq. (5.5) with some time step such as $\delta = 0.01$.

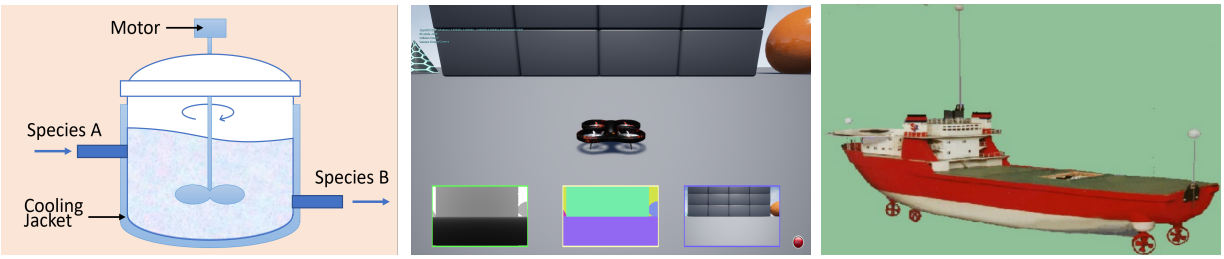
5.3.3. Time Oracle

The time oracle provides the horizon of MPC problems by computing a conservative deadline t_d by which the system should be recovered and a maintainable deadline t_m by which the system states can be maintained in the target state set once they are recovered into the set. This component is first used in [33] for linear systems, and this paper extends it to nonlinear systems.

At the time t_f, it leverages the state predictor and calculates the reachable state set of each following control step as if the norminal controller was running. If the upper or lower bound falls into the unsafe set, then there is a possibility that a severe consequence may happen. Thus, we choose the last time step before unsafe as our safety deadline t_d. Note that this analysis assumes that we do not take any recovery actions, so it is a conservative deadline. Then, it adds a constant maintenance period M to obtain the maintainable deadline $t_m = t_d + M$.

5.4. Evaluation

In this section, we validate our method using three nonlinear system simulators and highlight its effectiveness with observations and result analysis.



(a) CSTR (b) Drone from AirSim (c) Vessel model 1:70 scale[82]

Figure 26: Numerical and High-Fidelity CPS Simulators

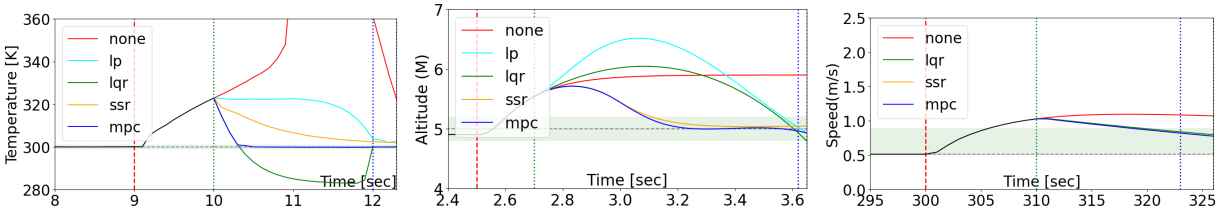


Figure 27: Performance of our method (MPC recovery control) compared to the baselines (no recovery, LP recovery control, LQR recovery control, and Software Sensor Recovery or SSR) on three benchmarks: continuous stirred tank reactor (CSTR) control **[left]**, quadrotor altitude control **[middle]**, naval vessel control **[right]**.

5.4.1. Experiment Environment

We implement a nonlinear system simulation tool using Python (in Appendix). We write these benchmarks' ODE and sensor attack scenarios in a configuration file. Then, the tool can run each configuration one by one. In this process, the necessary data, including system states, sensor values, and control input, are recorded to plot the results. The experiments are implemented on a 64-CPU server, where each one is an Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz. The optimizations are solved by the cvxpy library with OSQP solver. Pyinterval library is applied for the interval arithmetic of the MPC.

5.4.2. Non-linear systems benchmark

We consider three nonlinear system benchmarks: the continuously stirred tank reactor (CSTR), the quadrotor, and the naval vessel. They are representative CPSs, used in many works, from different domains. The CSTR benchmark study presents an interesting setup

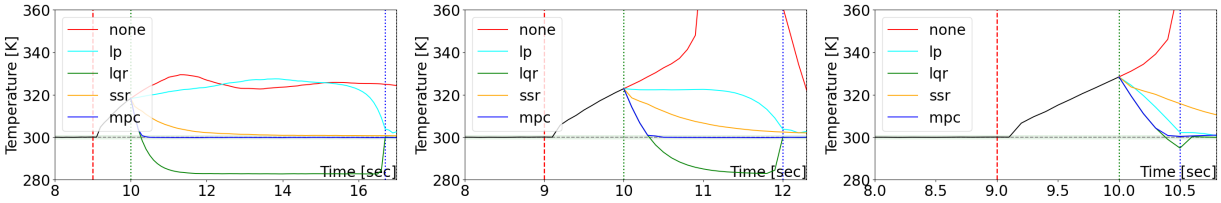


Figure 28: Sensitivity analysis to bias values of $\{-25, -30, -35\}$ on the CSTR benchmark.

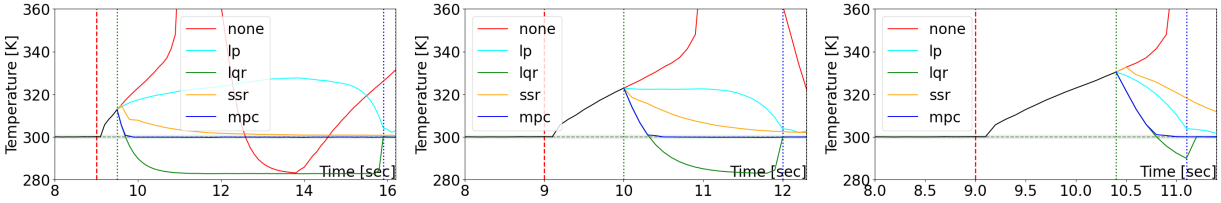


Figure 29: Sensitivity analysis to detection delay values of $\{0.5, 1.0, 1.4\}$ on the CSTR benchmark.

for industrial sabotage, the vessel benchmark has very long control steps, and the quadrotor with its large number of states tests the scalability of each component in the proposed recovery controller.

CSTR: In CSTR [83] dynamics, the exothermic reaction of the species $A \rightarrow B$ is considered with the concentration of A (C_A) and the reactor temperature (T) as states. The control input is given by the temperature of the cooling jacket (T_C). The exact dynamics given various system parameters $(k_0, C_{af}, q, V, E, R, \rho, C_p, T_f, \Delta H, UA)$ is as follows,

$$\begin{aligned} V\dot{C_A} &= q\left(C_{af} - C_A\right) - k_0 \exp\left(\frac{-E}{RT}\right) VC_A \\ \rho C_p V\dot{T} &= \rho C_p q\left(T_f - T\right) + \Delta H k_0 \exp\left(\frac{-E}{RT}\right) VC_A + UA\left(T_C - T\right) \end{aligned} \tag{5.8}$$

We utilize a PID controller to stabilize the temperature T. Furthermore, we consider a bias sensor attack scenario for CSTR where the temperature sensor values are compromised like in [84], and the bias parameter is shown in TABLE 8.

Quadrotor: The quadrotor dynamics [69], [74], [85], [86] describe the evolution of its at-

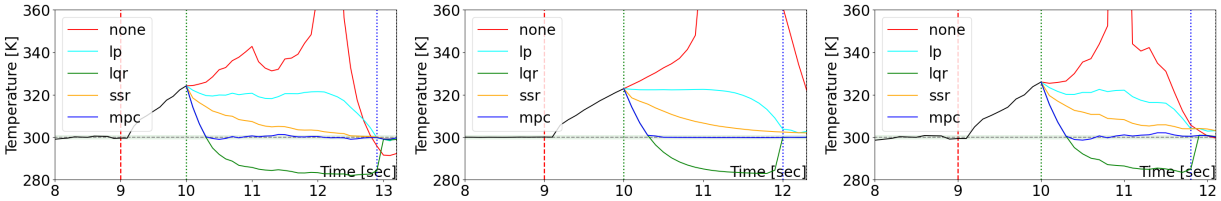


Figure 30: Sensitivity analysis to noise with upper bounds of $\{0.4, 0.1, 0.6\}$ on the CSTR benchmark.

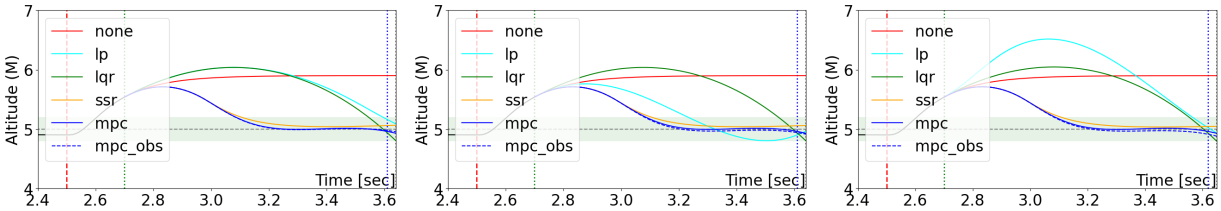


Figure 31: Sensitivity analysis of recovery with an observer to increasing noise with upper bounds of $\{0.05, 0.1, 0.25\}$ $\times 10^{-3}$ on the Quadrotor benchmark.

titude and position with 12 states: roll (ϕ), pitch (θ), yaw (ψ), roll rate (w_θ), pitch rate (w_ϕ), yaw rate (w_ψ), 3D positions and 3D velocities. The control input includes that for thrust, roll, pitch, and yaw represented by $U_t, U_\theta, U_\phi, U_\psi$. Given inertias (I_x, I_y, I_z), mass (m) and acceleration of gravity (g), it is given as,

$$
\begin{aligned}
\dot{\phi} &= w_\phi, \quad & \dot{w}_\phi &= \frac{U_\phi}{I_x} + \dot{\theta}\dot{\psi}\left(\frac{I_y - I_z}{I_x}\right) \\
\dot{\theta} &= w_\theta, \quad & \dot{w}_\theta &= \frac{U_\theta}{I_y} + \dot{\phi}\dot{\psi}\left(\frac{I_z - I_x}{I_y}\right) \\
\dot{\psi} &= w_\psi, \quad & \dot{w}_\psi &= \frac{U_\psi}{I_z} + \dot{\phi}\dot{\theta}\left(\frac{I_x - I_y}{I_z}\right) \\
\dot{x} &= v_x, \quad & \dot{v}_x &= \frac{U_t}{m}(\cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi)) \\
\dot{y} &= v_y, \quad & \dot{v}_y &= \frac{U_t}{m}(\cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi)) \\
\dot{z} &= v_z, \quad & \dot{v}_z &= \frac{U_t}{m}\cos(\phi)\cos(\theta) - g
\end{aligned}
\tag{5.9}
$$

The state-space model is abstracted from a high-fidelity simulator AirSim that Microsoft developed using the Unreal engine as shown in Figure 26b. A PID controller is used to maintain the quadrotor at a certain height. Here, we consider the sensor bias attack sce-

nario in which the attacker compromises height sensors resulting in incorrect feedback (see [15], [33]).

Vessel: The vessel dynamics describe the generic ship with 3 DoF[82], [87]. The state-space model is adapted from the lecture notes of Marine Cybernetics and the high-fidelity simulator developed from NTNU AUR-lab which implements the Matlab scripts in the VESSELS catalogue of the MSS toolbox. The simulator simulates the vessel shown in Figure 26c. The model implements the basic components of the ship such as engine, propeller, rudder, etc., and considers the whole model as a system. There are 8 states in the system, which are the east and north positions, yaw, their velocities, the angular shaft speed of the propeller, and the current of the DC motor.

Two PID controllers are used to maintain the surge speed of the vessel and the yaw of the vessel. Here, we consider the bias sensor attack scenario where the attacker compromises speed sensor, resulting in over speeding.

5.4.3. Experiment Settings

All experiment settings are given in Table 8. We chose our CSTR settings, including reference, frequencies (dt, MPC frequency), delay, safe/target sets based on previous work in [83]. We tuned a PID controller to stabilize the model within control limits determined by the physical properties of the CSTR system [83]. For the quadrotor, we build on the settings for the linearized model [33], while increasing the system frequency to 100Hz (dt=0.01s). We tune a PID controller for this increased speed and broader control limits (as given in Table 8's second column). Finally, for the Naval Vessel, we adapt reference, frequency, safe/target set settings from [82], and tune two PID controllers to stabilize the reference speed. Our control limits are again chosen based on the actuator properties of the vessel [82]. In all three benchmarks, we chose asymmetric positive noise to clearly observe the performance of the recovery algorithms. Symmetric noises, on the other hand, have little

impact on recovery, since the negative and positive noises offset each other. Our implementation can be found at https://github.com/CPSEC/nonlinear-recovery.

	CSTR	Quadrotor	Naval Vessel
attacked state	Temp.	Altitude	Speed
reference	300 K	5 m	0.55 m/s
dt	0.1 s	0.01 s	1 s
noise	unif[0, 0.1]	unif[0, 2e-4]	unif[0, .015]
detetection delay	1 s	0.2 s	10 s
bias	-30 K	-1 m	-0.5 m/s
safe set	(250, 360) K	(0, 200) m	(0, 150) m/s
target set	(299, 301) K	(4.8, 5.2) m	(0.5, 0.9) m/s
control limits	(250, 350) K	(-10,100) N	[(0, 2), (-1, 1)]
MPC freq.	10 Hz	10 Hz	1 Hz
Orig Controller	P=0.50 I=1.33 D=-0.05	P=100 I=0 D=-19	P=0.45, 0.1 I=0.05, 0 D=0, -3.5

Table 8: Settings used in each benchmark.

5.4.4. Baselines

We compare the proposed data-predictive recovery method approach (mpc) with four baselines. We also add an observer to our method and include it in the comparisons as mpc_obs.

(i) no recovery (none). This baseline does not take any recovery countermeasures after the detection of sensor attacks.

(ii) software-sensor-based recovery (ssr) [4], [26]. After the detection of sensor attacks, the baseline replaces the corrupted physical sensor data with the software sensor data predicted by the linear system model.

(iii) linear-programming-based recovery (lp) [15]. The baseline formulates the recovery problem as a linear programming optimization problem. By solving this problem, it gets a recovery control sequence that can steer the system states back to the target set

before a safe deadline.

(iv) linear-quadratic-regulator-based recovery (lqr) [33]. The baseline extends the LP-based recovery by considering state and control input penalty in the objective, and also can maintain the system states in the target set for a while.

Given that all of the above baselines are designed for linear systems. To apply them to our nonlinear benchmarks, we obtain linear models by linearizing the nonlinear systems over the equilibrium point or the operating point.

5.4.5. Recovery Effect

We compared the proposed method with four baselines for the three benchmarks. We plot the actual system physical states in Figure 27, which shows the recovery process targeting sensor attacks. The red line shows the physical states under attacks without any recovery. The cyan, blue, and orange curves are actual physical states that use linear-programming-based recovery, linear-quadratic-regulator-based recovery, and software-sensor-based recovery benchmarks.

We can get the following observations from them: **Without recovery, the system states deviate from the target states, even reaching the unsafe states.** When there is a bias attack, i.e., adding a bias to sensor measurement, the controller computes an error between state estimate and reference state. This error makes the controller generate a control input to eliminate this error, resulting in deviating from the reference state. For example, in the CSTR benchmark, the red curve goes out of the figure, i.e., reaching the unsafe set. For other benchmarks, although the system states do not reach the unsafe set with any countermeasures, the performance of the CPSs is affected. The system states keep deviating from the target states. If we choose a larger bias, the system states may also reach the unsafe set.

The linear model based recovery method may fail to find a recovery control sequence. For lp and lqr baselines, we do not use the original deadline estimator, because they often give a short deadline, resulting in failure to find a recovery control sequence. The short deadline comes from the modeling error. When the error is large, the reachability analysis may quickly touch the unsafe set and get a short deadline.

The proposed method steers the system states closest to the reference states. For the CSTR benchmark, ssr and lp do not drive the system state to the target set before the deadline. For the quadrotor benchmark, the lqr baseline does not drive the physical states to the deadline eventually, and the final recovered states of the proposed method are closer to the reference state than other baselines. This may be caused by uncertainty accumulation. The baseline methods assume that all sensors are not reliable, and do not get any feedback from sensors. Thus, they cannot reject the external disturbance, and cause an inaccurate recovery eventually.

5.4.6. Sensitivity Analysis.

In order to clearly show how different parameters affect the methods' performance, it is ideal to choose a benchmark that has small inertia. Therefore, we perform sensitivity analyses on the CSTR benchmark, and observe changes to variation in attack intensity, detection delay, and noise intensity. We compare all methods by their ability to recover to the target set without encountering the unsafe set *within the deadline*, their time of recovery, and their maintainable time in the target set.

Impact of Attack Intensity. The experiment changes the value of bias added to the sensor measurement, and the biases in the experiments are -25, -30, and -35K. There are two main observations from Figure 28: First, the SSR baseline and the proposed method performed well if the bias is set to -25. Both can drive the system to the target set and maintain it before the deadline. The LQR and LP baseline cannot achieve it. Further-

more, the proposed method can drive the system to the target set and maintain it even when the bias is set to -30 and -35K since the proposed MPC-based method recovery takes less time to recover the system to the target set. A lower attack bias situates the system state closer to the target set at detection time. Yet, we see that methods using linearized models struggle to overcome the accumulation of uncertainty even from a state with a lower bias. SSR, enabled by a nonlinear model in state reconstruction, but with only a linear model during recovery is seen to recover with a large delay compared to our approach. With higher bias, and hence a state further away from the target set at detection time, all other methods struggle to recover and maintain in the target set.

Impact of Detection Delay. Figure 29 shows the recovery results under different detection delays. There are two main observations: First, both SSR and the proposed method can drive the system to the target set and maintain it before the deadline if the detection delay is 0.5s. LQR and LP cannot achieve recovery before the deadline. Secondly, the SSR baseline cannot achieve recovery before the deadline if the detection delay increased to 1s and 1.4s since there is less recovery time. A decrease in detection delay prevents the significant drift from the target state during attack time. Yet, again, only SSR is able to recover but is unable to maintain in the target state without a controller that optimizes for maintain time. On the other hand, a longer detection delay pushes the system further away from the target set and prevents any other method from recovering within the deadline.

Impact of Uncompromised Sensors. Across all three simulators in Figure 27 and analyses in Figs. 28, 29, 28, our method leverages the uncompromised sensor information, while LQR and LP cannot do so. Thus, LQR and LP consistently take longer to recover (sometimes after the deadline), and are unable to maintain states in the target set even with these requirements encoded in the optimization problem formulations.

Impact of an observer. We identify the impact of an observer on the quadrotor bench-

mark (instead of the CSTR benchmark). The CSTR benchmark only has two directly measured state elements. Hence, adding an observer does not create any measurable change. On the other hand, the quadrotor sensors only measure four (out of twelve) elements of the state – height, roll rate, pitch rate, and yaw rate. Height is obtained from an ultrasonic sensor below the quadrotor. The attitude rates are obtained from an onboard inertial measurement unit. We use an extended Kalman filter (EKF) to estimate the full state (with twelve elements). This additional observer step increases the computational overhead minimally (see Figure 32). Moreover, as seen in the left plot of Figure 31, the observer does not significantly affect recovery at lower noise thresholds. With increased noise, as seen in the right plot of Figure 31, there is an increased deviation from the tracked state after recovery. And hence, reduced maintainability in the recovered state. Thus, with an increase in noise, while recovery to the desired region is not affected, maintainability over a long horizon in the desired region may be affected.

Impact of Noise intensity. Two types of noise were implemented on the sensors, uniform noise, and white noise. The lower bound of these noises is 0, and we tested our method with different levels of upper bounds for the noises. Figure 30 shows that the proposed method can drive the system to the target set and maintain it when the noise upper bound is set to 0.1 and 0.4. None of the baselines can achieve before the deadline except our proposed method. If the noise upper bound is large, our proposed method recovers but finds it hard to maintain the system in the target set since the uncompromised data (which we leverage for improved state reconstruction) is itself too noisy. Other methods find it difficult to recover, let alone maintain in the target state for more than one step.

Computational overhead. Figure 32 shows the box plots of the computation overhead for each step of the recovery on the time horizon. Our MPC-based recovery method has the second largest overhead since more computation (than LP or LQR) is involved. But,

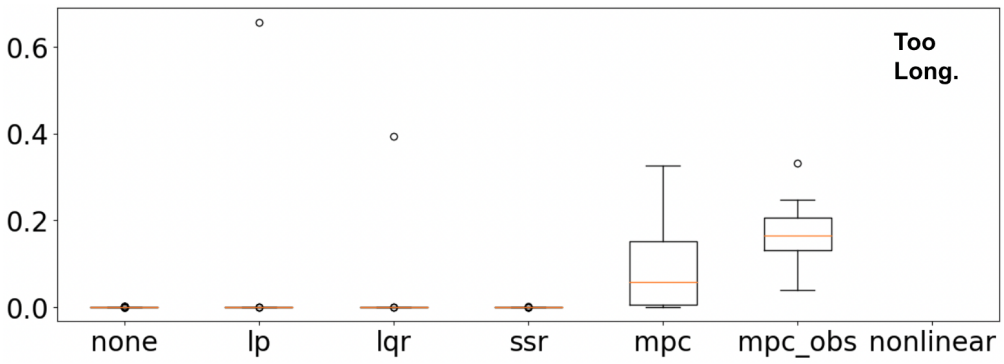


Figure 32: Computational overhead (in seconds) for all methods on the Quadrotor benchmark. For both LP and LQR, the outlier point on top represents the overhead in formulating and solving the problem at time step t_f.

if the MPC frequency is reduced (i.e. if the frequency of linearization is reduced), then our proposed method is applicable to real-time scenarios. On the other hand, our MPC method outperforms the LP and LQR recovery results. Moreover, if the non-linear dynamics (without linearization) were directly applied to the recovery problem, we would have the best recovery result because of zero adaption error. But the nonlinear optimization takes too long and cannot be applied to real-time scenarios. Therefore, from Figure 32, there is a trade-off between usability and recovery performance; and our proposed method delivers both.

5.5. Conclusion

In this chapter, we propose a novel framework for the recovery of nonlinear CPS when faced with sensor attacks. Our framework solves an MPC problem formulated every few time-steps to generate control inputs for recovery to a target set within a dynamically computed recovery deadline (to remain in a safe set). It utilizes a model adaptor to linearize and discretize nonlinear models for the MPC problem and computes both the deadline and initial MPC state based on nonlinear reachability analysis with Flow*. An evaluation of the nonlinear system benchmarks alongside an analysis of recovery sensitivity to our novel framework's components demonstrate the effectiveness of our approach.

CHAPTER 6

Summary

This chapter concludes the dissertation. Section 6.1 provides a summary of the contributions made by this dissertation, while Section 6.2 outlines existing challenges and suggests several directions for future research.

6.1. Conclusions

Cyber-Physical Systems (CPS) combine computing and networking components with physical processes through sensing and actuation units. A significant security risk in these systems is sensor attacks, where attackers manipulate sensor measurements to force the physical system into unsafe states, leading to severe personal injuries, societal damage, and financial losses. Traditional security approaches for IT systems are insufficient to address threats related to physical states. Hence, the primary aim of this dissertation is to develop innovative defense mechanisms that proactively and reliably protect CPSs against sensor attacks. Specifically, we propose adaptive attack detection techniques for identifying sensor attacks. To maximize the benefits of attack detection, we also introduce real-time recovery methods, which enable CPSs to quickly return to a safe and desired physical state.

6.1.1. Real-time Adaptive Detection against Sensor Attack

In Chapter 3, we propose a real-time adaptive sensor attack detection system that dynamically balances detection delay and false alarms based on the current state estimate. This system consists of three key components: (i) The Adaptive Detector employs a window-based detection algorithm that dynamically adapts detection delay and false alarms to meet detection deadlines and improve usability. (ii) The Detection Deadline Estimator utilizes a reachability analysis-based technique to conservatively estimate detection dead-

lines at runtime by computing the reachable set of a system's potential future behaviors. (iii)The Data Logger implements a sliding-window-based data logging protocol to retain trustworthy data for deadline estimation and maintain sufficient data points for attack detection. Ultimately, we implement our detection system in multiple CPS simulators and a reduced-scale autonomous testbed to validate its efficiency and effectiveness.

6.1.2. Real-time Sensor-attack Recovery in Linear CPSs

After detecting sensor attacks, CPSs must respond to these threats and mitigate their adverse effects. Existing approaches either restart the system or replace corrupted sensor data with predicted values, but these methods lack safety and timing guarantees. To address this issue, in Chapter 4, we propose a real-time attack recovery method for linear CPSs that offers strong safety and timing guarantees using formal methods. Our approach develops a linear-programming-based or quadratic-programming-based recovery controller that generates a recovery control sequence, smoothly guiding the physical system under sensor attacks back to a target set before a safety deadline and maintaining its position within the set. The method employs reachability analysis techniques to reconstruct state estimates and compute a safety deadline, beyond which the system may enter unsafe states. Through multiple CPS simulators, we demonstrate that our approach effectively recovers attacked linear systems in a timely and safe manner.

6.1.3. Real-time Sensor-attack Recovery in Complex CPSs

In practical CPSs, plants are typically complex, with dynamics that are often nonlinear. On the one hand, performing reachability analysis and optimization on nonlinear systems directly is time-consuming. On the other hand, relying on a linearized model may result in significant modeling errors and recovery failures, as such models only function well around equilibrium points. In Chapter 5, we propose an innovative recovery system for nonlinear CPSs that strikes a balance between efficiency and accuracy. Our system employs a state

predictor that utilizes Flow*, a tool designed for rapid nonlinear reachability analysis. Additionally, it continuously updates linear approximations based on the current state estimate, ensuring high accuracy within a small range. The proposed method also leverages uncorrupted sensor data to enhance recovery performance. Evaluations using nonlinear system benchmarks, as well as an analysis of recovery sensitivity to various components of our novel framework, demonstrate the effectiveness of our approach.

6.2. Future Work

There remain several challenges in defending CPSs. First, in order to ensure timing and safety guarantees, this dissertation utilizes system models for detecting sensor attacks and recovering CPSs. However, obtaining accurate system models is not always straightforward. To address this challenge, we plan to explore data-driven methods that do not require a system model in advance. Second, we assume that uncertainties, such as sensor noise or process disturbances, are bounded. In real applications, however, this assumption may not hold true. In future work, we aim to investigate how to ensure safety properties in the presence of unbounded noise. Third, our attack recovery approach involves solving optimization problems with safety and timing guarantees, which may not always yield valid solutions in rare cases. This means that our methods are sound but not complete. We will further explore how to make the attack recovery approach more complete and robust for real-world applications.

APPENDIX A

Simulation and Security Toolbox for Cyber-Physical Systems

Attack detectors aim at identifying attacks at the earliest time, and attack recovery methods try to eliminate the impact caused by these attacks and even steer the system's physical states to a target set[32], [88]. However, there are few solutions that help to evaluate the efficacy and efficiency of these security counter measurements due to the following challenges: (i) tremendous efforts are required to collect benchmark plants, design controllers, customize attacks, build defense approaches, and evaluate these approaches. (ii) the existing solutions are difficult to add new features or integrate with existing simulators. (iii) besides cyber states, the physical behavior of systems also requires to be simulated.

To address these challenges, we develop a simulation and security toolbox with high extendibility and flexibility. One can easily switch between different experiment settings and apply defense prototypes responding to different attacks. The source code is available at https://github.com/lion-zhang/CPSim.

A.1. Security Toolbox Design

A.1.1. Toolbox Overview

The proposed toolbox includes a CPS simulator and a set of security tools. As shown in Figure 33, the simulator mimics the behavior of a CPS: Sensors measure system states and forward measurements to observers. Meanwhile, the measurements could suffer from external uncertainties and attacks to meet experiment needs. On the basis of them, the

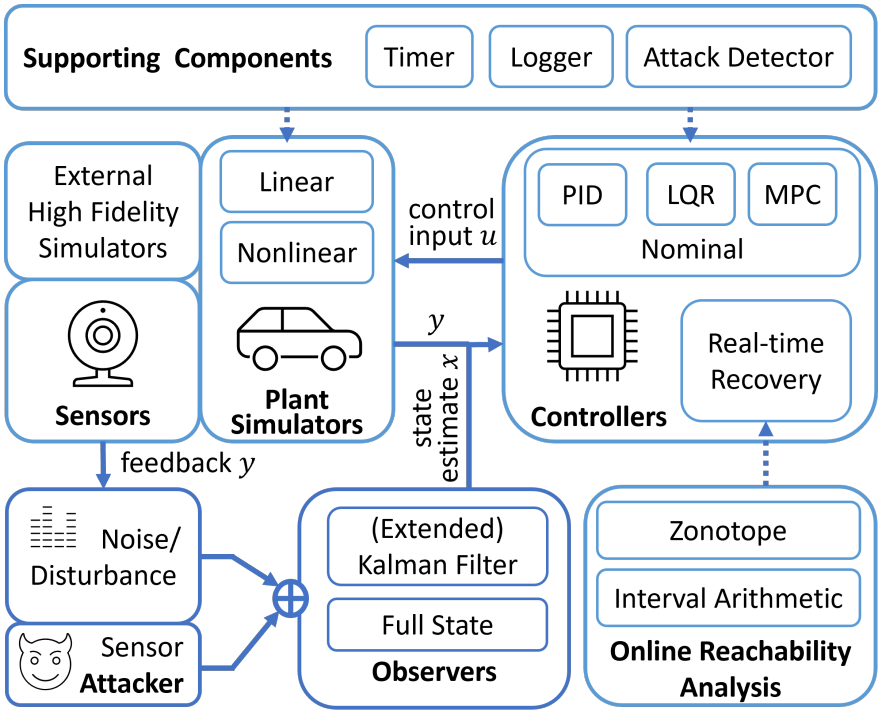


Figure 33: Design Overview of Simulation and Security Toolbox

observers are responsible for providing state estimates for the controllers. Then, the controllers generate control input to be implemented in physical plants. Plant simulators update system states according to system dynamics and control input. To secure CPS, various attack detectors and real-time attack recovery controllers are included to respond to those attacks.

A.1.2. Component Implementations

This section elaborates on all components in the toolbox.

Plant Simulators: The toolbox provides some out-of-the-box CPS benchmarks from different domains. *Linear benchmarks* are defined using state-space linear time-invariant (LTI) models, including the F16 fighting falcon, serial RLC circuit, motor speed, etc. *Nonlinear benchmarks* are defined by order differential equations (ODEs), including continuous stirred tank reactor, inverted pendulum, quadrotor, etc. It is simple to switch between all

these benchmark plants with controllers by modifying the configuration file.

Controllers: The toolbox integrates common *nominal controllers*, such as PID, LQR, and MPC controllers. It is convenient to complete various control tasks, such as cruise control and lane keeping in an autonomous vehicle, using those nominal controllers with appropriate parameters. In addition, the toolbox also supports *real-time attack-recovery controllers*, which take over the system after identifying an attack. The controllers can generate a recovery control sequence that steers CPS's physical states back to a target set after an attack. The recovery controllers rely on the formal method component.

Observers: Some controllers rely on state estimates calculated from sensor measurements by observers. The toolbox provides common observers, such as the Kalman filter for linear systems and the extended Kalman filter for nonlinear systems. In most cases, we obtain the ground-truth states from the simulators directly, and the observers then become optional.

Online Reachability Analysis: Reachability analyses predict the system's reachable states, all possible physical states at subsequent control steps. If reachable states do not intersect with an unsafe set, the safety property must be satisfied. The toolbox contains efficient approaches to online reachability analysis. For linear systems, it leverages the properties of a linear transformation of the zonotope and the support function method. For nonlinear systems, it uses interval arithmetic. In addition, the toolbox supports other formal representations, such as half-space and strip, to express unsafe and target state sets. Moreover, it also supports operations on Gaussian distributions to deal with stochastic systems.

Noise and Attacks: The toolbox simulates the ubiquitous noise or disturbance in real systems. The uncertainty may follow bounded uniform distributions, unbounded Gaussian

distributions, etc. Besides, the toolbox simulates attacks that compromise the integrity or availability of sensor measurements, such as bias, replay, and delay attacks.

Supporting Components: The timer device simulates the system clock and activates control steps. The logger checkpoints historical data, such as the state estimate, sensor measurement, and control input, and prints the necessary debug information. The toolbox also reserves the interface for different attack detectors, such as CUSUM, chi-square.

A.1.3. Requirements and Customizability

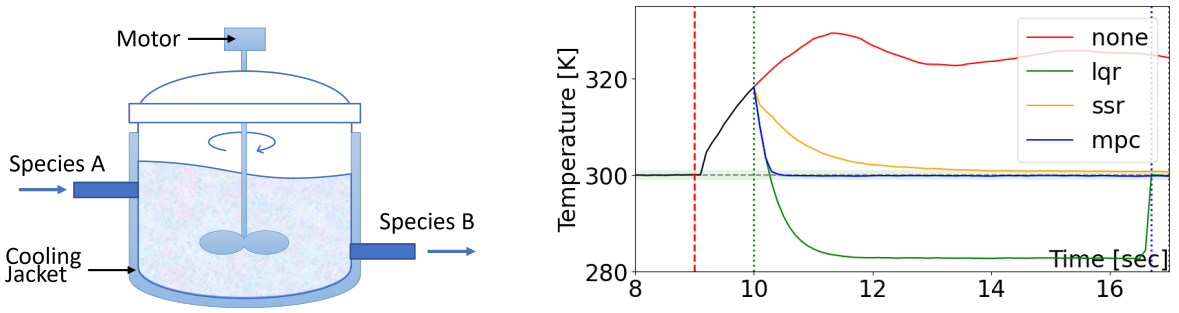
The toolbox is implemented in Python 3, and thus can be installed in various operating systems with a Python environment. The main dependencies are scipy, numpy and cvxpy packages. Moreover, it is convenient to carry out secondary development because of two aspects:

(i) **high extendibility.** The toolbox is written in a modular fashion, and each component is organized into a package. Thus, it is easy to extend its built-in functions or add new features. For example, users can add a new CPS according to their needs by modifying the system dynamics and controllers from the template file.

(ii) **high flexibility.** Besides numerical simulations, the toolbox can be easily deployed in common high-fidelity simulators, such as AirSim and CARLA. Also, it can be integrated into the Robot operating system (ROS), a set of open-source software libraries and tools for building robot applications. Thus, the toolbox is effortlessly deployed in real robots or CPS testbeds.

A.2. Toolbox Demonstration

To demonstrate the toolbox usage, we show a real-time attack recovery on CSTR numerical simulator and another one on SVL high-fidelity simulator.



(a) CSTR (b) Recovery Results

Figure 34: Attack Recovery Performance for Baselines

(a) Attacked (b) Recovering (c) Recovered

Figure 35: Attack Recovery Performance for Baselines

A.2.1. Working with a built-in numerical simulator

First, we aim to evaluate the recovery performance of different baseline recovery controllers. We only require modifying the configuration file rather than writing simulation code. In this file, we choose a benchmark plant, CSTR shown in Figure 34a, controlled by a PID controller. Also, we define the bias sensor attack that subtracts 25K from the temperature sensor feedback starting from the ninth second. The detector identifies the attack at the 10th second, and triggers the recovery controllers. Baseline recovery controllers include (i) no recovery method (none), (ii) software-sensor-based recovery (ssr [4]), (iii) linear-quadratic-regulator-based recovery (lqr [33]), and (vi) data-predictive recovery (mpc [89]).

Figure 34b plots the ground truth temperature from the simulator. From the curve, we can intuitively analyze the recovery performance of each baseline recovery controller.

A.2.2. Working with an external high-fidelity simulator

Then, we demonstrate how to use the toolbox to recover an autonomous vehicle in the SVL simulator. The vehicle suffers from an IMU sensor attack, deviates from its own lane, and even enters the oncoming lane, as shown in Figure 35a. To apply the lqr recovery controller after detecting the attack, we need to integrate the toolbox with the SVL simulator. Since there is a ROS bridge communicating with the simulator, we load the toolbox in a ROS node, which is responsible for recovering the vehicle from the attack within a safety deadline.

Figure 35b shows that vehicle returns to its lane during the recovery process. Figure 35c shows that the recovery controller steers the vehicle to a safe region, the road shoulder, to avoid a collision after recovery.

APPENDIX B

Robotic vehicle Testbed

Autonomous vehicles are a type of CPS that rely on sensor information to perform tasks such as path tracking and lane keeping. We built scaled autonomous vehicles, measuring 24 cm in length and 19 cm in width, as testbeds (see Figure 36) to evaluate the proposed attack detection and recovery methods.

B.1. Vehicle Architecture

Autonomous vehicles sense states and environments, make decisions, and control mobility. Our robotic vehicle testbeds, whose hardware architecture is shown in Figure 37, simulate these functions through the following stages:

(i) Perception: Our testbed is equipped with an Inertial Measurement Unit (IMU), Ultra-wideband (UWB), and encoder sensors that measure attitude, position, and velocity, respectively. We can also use cameras and LiDAR to collect additional environmental data. However, these sensors are vulnerable to sensor attacks.

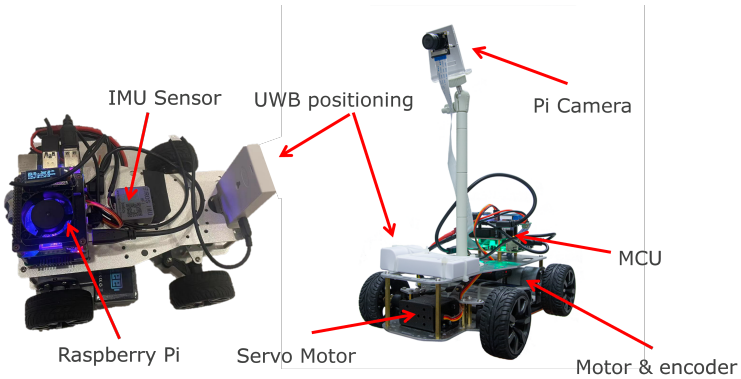


Figure 36: Robotic Vehicle Testbed

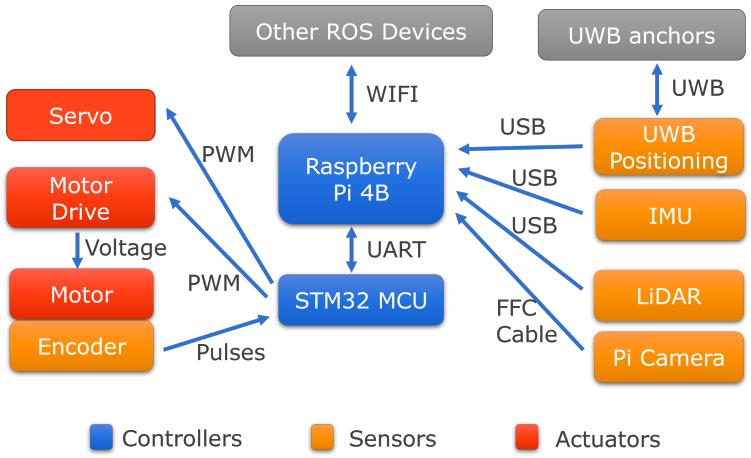


Figure 37: Hardware Architecture of Robotic Vehicle Testbed

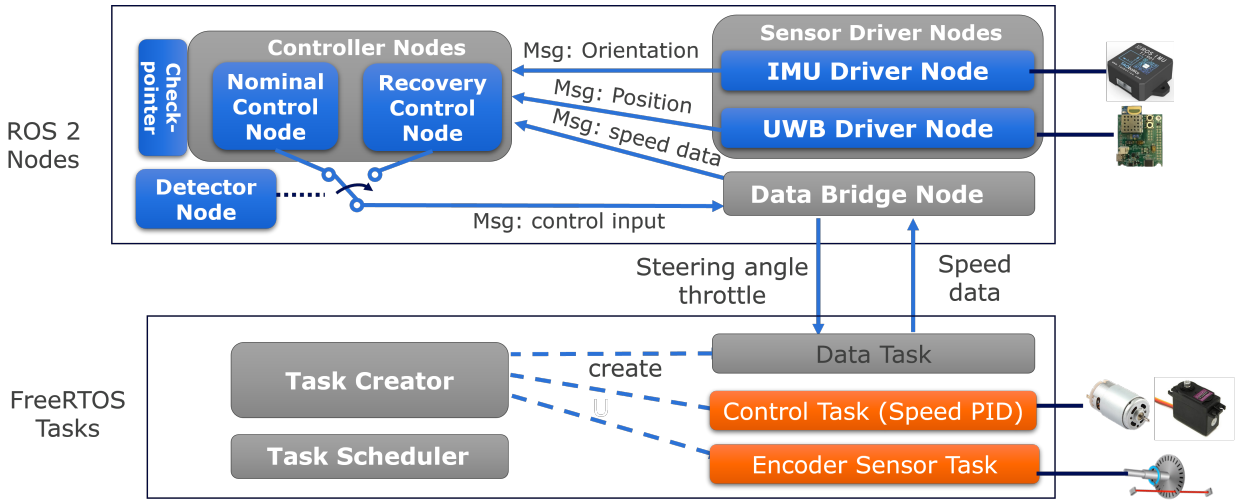
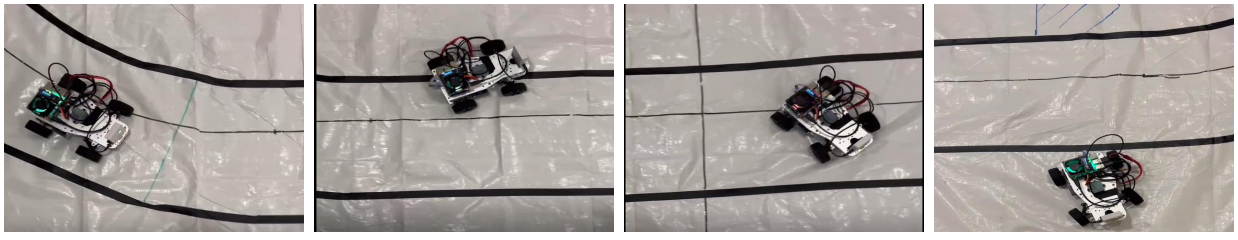


Figure 38: Real-time Attack Recovery Implementation on Robotic Vehicle Testbed



(a) normal (b) attacked (c) recovering (d) recovered

Figure 39: Recovery demonstration from our testbed.

(ii) Decision Making: A Raspberry Pi with Robot Operating System (ROS2) serves as the main controller. It collects sensor data, estimates vehicle states, and generates control signals. The system uses different controllers for longitudinal and lateral control. For cruise control, a PID controller stabilizes the testbed's velocity based on encoder feedback. For lane keeping, a Stanley Controller uses the front axle as its reference point, considering both heading error and cross-track error. We can also deploy the proposed attack detection and recovery algorithms on this system.

(iii) Control: An STM32 microcontroller with FreeRTOS system receives control signals from the Raspberry Pi through a Universal Asynchronous Receiver/Transmitter (UART) protocol. Running a real-time operating system, it performs time-sensitive tasks such as generating Pulse Width Modulation (PWM) signals to drive actuators.

(iv) Actuator: The actuator stage includes components such as motors and servos that execute vehicle movements according to control signals.

B.2. Case Study

We implement the attack recovery method in Chapter 4 on the robotic vehicle testbed, and the design is illustrated in Figure 38.

B.2.1. System Model

Autonomous vehicles perform lateral control to track paths provided by path planning modules. High-fidelity models of vehicle dynamics are complex, non-linear, and discontinuous. However, to reduce computational complexity, path tracking controllers typically consider a simplified lateral dynamics model of the vehicle [90], [91] (as shown in Equation (B.1)). This model approximates dynamic effects to improve tracking performance.

$$\frac{d}{dt}\begin{bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-\left(c_f+c_r\right)}{mv_x} & 0 & \frac{\left(l_r c_r - l_f c_f\right)}{mv_x} - v_x \\ 0 & 0 & 0 & 1 \\ 0 & \frac{l_r c_r - l_f c_f}{I_z v_x} & 0 & \frac{-\left(\ell_f^2 c_f + \ell_r^2 c_r\right)}{I_z v_x} \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{c_f}{m} \\ 0 \\ \frac{l_f c_f}{I_z} \end{bmatrix} \delta \qquad \text{(B.1)}$$

Here, c_f and c_r represent cornering stiffness for the front and rear tires; l_f and l_r are the distances from the front and rear tires to the vehicle's center of gravity; I_z is the vehicle's moment of inertia; and v_x is the longitudinal velocity. The system states are lateral distance from the path (y), lateral error rate ($\dot{y}$), yaw error (ψ), and yaw error rate ($\dot{\psi}$). The control input is the front wheel steering angle (δ).

B.2.2. Attack Scenario

To achieve path tracking, we implement the Stanley lateral controller from [91] in ROS. The control law is expressed as $\delta(t) = \psi(t) + \tan^{-1}\left(\frac{ky(t)}{v_x(t)}\right)$. The yaw error ($\psi$) is obtained from the IMU sensor, and the lateral distance from the path (y) is obtained from the UWB sensor for indoor use. In the absence of sensor attacks, the controller can perform path tracking tasks with good performance.

The attacker launches an attack on the IMU sensor, reducing the value of ψ by 0.60 radians from the start of the attack, with a detection delay of 60 control steps. Figure39b shows the attack result as detected by the detector. Subsequently, our proposed method begins controlling the vehicle back to the safe zone, as shown in Figure39d.

BIBLIOGRAPHY

[1] A. Zambrano, A. P. Betancur, L. Burbano, *et al.*, "You make me tremble: A first look at attacks against structural control systems," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1320–1337.

[2] A. Staff, *How amazon is building its drone delivery system*, Aug. 2022. [Online]. Available: https://www.aboutamazon.com/news/transportation/how-amazon-is-building-its-drone-delivery-system.

[3] A. Humayed, J. Lin, F. Li, and B. Luo, "Cyber-physical systems security—a survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1802–1831, 2017.

[4] F. Kong, M. Xu, J. Weimer, O. Sokolsky, and I. Lee, "Cyber-physical system checkpointing and recovery," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, IEEE, 2018, pp. 22–31.

[5] R. Quinonez, J. Giraldo, L. Salazar, E. Bauman, A. Cardenas, and Z. Lin, "Savior: Securing autonomous vehicles with robust physical invariants," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[6] C. Yan, H. Shin, C. Bolton, W. Xu, Y. Kim, and K. Fu, "Sok: A minimalist approach to formalizing analog sensor security," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 480–495.

[7] Y. Tu, Z. Lin, I. Lee, and X. Hei, "Injected and delivered: Fabricating implicit control over actuation systems by spoofing inertial sensors," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1545–1562.

[8] J. Shen, J. Y. Won, Z. Chen, and Q. A. Chen, "Drift with devil: Security of multi-sensor fusion based localization in high-level autonomous driving under gps spoofing," in *Proceedings of the 29th USENIX Conference on Security Symposium*, 2020, pp. 931–948.

[9] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote attacks on automated vehicles sensors: Experiments on camera and lidar," *Black Hat Europe*, vol. 11, p. 2015, 2015.

[10] M. Pajic, J. Weimer, N. Bezzo, *et al.*, "Robustness of attack-resilient state estimators," in *ACM/IEEE 5th International Conference on Cyber-Physical Systems (ICCPS)*, IEEE Computer Society, 2014, pp. 163–174.

[11] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, no. 4, pp. 617–636, 2014.

[12] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful gps spoofing attacks," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 75–86.

[13] J. Noh, Y. Kwon, Y. Son, *et al.*, "Tractor beam: Safe-hijacking of consumer drones with adaptive gps spoofing," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1–26, 2019.

[14] A. Cardenas, "Cyber-physical systems security knowledge area issue," *The Cyber Security Body Of Knowledge.[Online]. Available: https://www.cybok.org/media/downloads/Cyber-Physical_Systems_Security_issue_1.0.pdf*, no. 1.0,

[15] L. Zhang, X. Chen, F. Kong, and A. A. Cardenas, "Real-time recovery for cyber-physical systems using linear approximations," in *41st IEEE Real-Time Systems Symposium (RTSS)*, IEEE, 2020.

[16] H. Choi, W.-C. Lee, Y. Aafer, *et al.*, "Detecting attacks against robotic vehicles: A control invariant approach," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 801–816.

[17] T. He, L. Zhang, F. Kong, and A. Salekin, "Exploring inherent sensor redundancy for automotive anomaly detection," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218557.

[18] R. Wang, F. Kong, H. Sudler, and X. Jiao, "Hdad: Hyperdimensional computing-based anomaly detection for automotive sensor attacks," in *27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Brief Industry Paper Track*, IEEE, 2021.

[19] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, *et al.*, "Limiting the impact of stealthy attacks on industrial control systems," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1092–1105.

[20] J. Giraldo, D. Urbina, A. Cardenas, *et al.*, "A survey of physics-based attack detection in cyber-physical systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[21] J. Giraldo, E. Sarkar, A. A. Cardenas, M. Maniatakos, and M. Kantarcioglu, "Security and privacy in cyber-physical systems: A survey of surveys," *IEEE Design & Test*, vol. 34, no. 4, pp. 7–17, 2017.

[22] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks against process control systems: Risk assessment, detection, and response," in *Proceedings of the 6th ACM symposium on information, computer and communications security*, 2011, pp. 355–366.

[23] F. Akowuah and F. Kong, "Physical invariant based attack detection for autonomous vehicles: Survey, vision, and challenges," in *2021 Fourth International Conference on Connected and Autonomous Driving (MetroCAD)*, IEEE, 2021, pp. 31–40.

[24] F. Akowuah, R. Prasad, C. O. Espinoza, and F. Kong, "Recovery-by-learning: Restoring autonomous cyber-physical systems from sensor attacks," in *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, IEEE, 2021, pp. 61–66.

[25] F. Fei, Z. Tu, D. Xu, and X. Deng, "Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyber-physical attacks," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 7358–7364.

[26] H. Choi, S. Kate, Y. Aafer, X. Zhang, and D. Xu, "Software-based realtime recovery from sensor attacks on robotic vehicles," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020, pp. 349–364.

[27] R. Ma, S. Basumallik, S. Eftekharnejad, and F. Kong, "Recovery-based model predictive control for cascade mitigation under cyber-physical attacks," in *2020 IEEE Texas Power and Energy Conference (TPEC)*, IEEE, 2020, pp. 1–6.

[28] R. Ma, S. Basumallik, S. Eftekharnejad, and F. Kong, "A data-driven model predictive control for alleviating thermal overloads in presence of possible false data," *IEEE Transactions on Industry Applications*, 2021.

[29] K. Zhou and J. C. Doyle, *Essentials of robust control*. Prentice hall Upper Saddle River, NJ, 1998, vol. 104.

[30] M. Pajic, J. Weimer, N. Bezzo, O. Sokolsky, G. J. Pappas, and I. Lee, "Design and implementation of attack-resilient cyberphysical systems: With a focus on attack-resilient state estimators," *IEEE Control Systems Magazine*, vol. 37, no. 2, pp. 66–81, 2017.

[31] C. Fan, B. Qi, S. Mitra, M. Viswanathan, and P. S. Duggirala, "Automatic reachability analysis for nonlinear hybrid models with c2e2," in *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, Springer, 2016, pp. 531–538.

[32] L. Zhang, Z. Wang, M. Liu, and F. Kong, "Adaptive window-based sensor attack detection for cyber-physical systems," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 919–924.

[33] L. Zhang, P. Lu, F. Kong, X. Chen, O. Sokolsky, and I. Lee, "Real-time attack-recovery for cyber-physical systems using linear-quadratic regulator," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5s, Sep. 2021, ISSN: 1539-9087. DOI: 10.1145/3477010. [Online]. Available: https://doi.org/10.1145/3477010.

[34] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A survey of fault detection, isolation, and reconfiguration methods," *IEEE transactions on control systems technology*, vol. 18, no. 3, pp. 636–653, 2009.

[35] A. A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *The 28th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, IEEE, 2008, pp. 495–500.

[36] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 13, 2011.

[37] R. Ivanov, M. Pajic, and I. Lee, "Attack-resilient sensor fusion for safety-critical cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 1, pp. 1–24, 2016.

[38] P. Lu, L. Zhang, B. B. Park, and L. Feng, "Attack-resilient sensor fusion for cooperative adaptive cruise control," in *21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 3955–3960.

[39] A. Chovancová, T. Fico, L. Chovanec, and P. Hubinsk, "Mathematical modelling and parameter identification of quadrotor (a survey)," *Procedia Engineering*, vol. 96, pp. 172–181, 2014.

[40] F. Akowuah and F. Kong, "Real-time adaptive sensor attack detection in autonomous cyber-physical systems," in *27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2021.

[41] M. Wolf and D. Serpanos, "Safety and security in cyber-physical systems and internet-of-things systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 9–20, 2017.

[42] M. Green and D. J. Limebeer, *Linear robust control*. Courier Corporation, 2012.

[43] G. Welch, G. Bishop, *et al.*, "An introduction to the kalman filter," 1995.

[44] Y. Mo, E. Garone, A. Casavola, and B. Sinopoli, "False data injection attacks against state estimation in wireless sensor networks," in *49th IEEE Conference on Decision and Control (CDC)*, IEEE, 2010, pp. 5967–5972.

[45] W. Heemels, J. Sandee, and P. Van Den Bosch, "Analysis of event-driven controllers for linear systems," *International journal of control*, vol. 81, no. 4, pp. 571–590, 2008.

[46] J. Sandee, W. Heemels, and P. Van Den Bosch, "Event-driven control as an opportunity in the multidisciplinary development of embedded controllers," in *American Control Conference*, IEEE, 2005, pp. 1776–1781.

[47] J.-P. Gauthier and I. A. Kupka, "Observability and observers for nonlinear systems," *SIAM journal on control and optimization*, vol. 32, no. 4, pp. 975–994, 1994.

[48] M. M. Waldrop *et al.*, "No drivers required," *Nature*, vol. 518, no. 7537, p. 20, 2015.

[49] T. Sato, J. Shen, N. Wang, Y. Jia, X. Lin, and Q. A. Chen, "Dirty road can attack: Security of deep learning based automated lane centering under physical-world attack," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3309–3326.

[50] C. Zhou, Q. Yan, Y. Shi, and L. Sun, "DoubleStar: Long-Range attack towards depth estimation based obstacle avoidance in autonomous systems," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 1885–1902, ISBN: 978-1-939133-31-1.

[51] A. H. Rutkin, *Spoofers use fake gps signals to knock a yacht off course*, MIT Technology Review, Online; accessed May 2020, 2013.

[52] C. Le Guernic, "Reachability Analysis of Hybrid Systems with Linear Continuous Dynamics," Theses, Université Joseph-Fourier - Grenoble I, Oct. 2009. [Online]. Available: https://tel.archives-ouvertes.fr/tel-00422569.

[53] F. Kong, O. Sokolsky, J. Weimer, and I. Lee, "State consistencies for cyber-physical system recovery," in *Workshop on Cyber-Physical Systems Security and Resilience (CPS-SR)*, 2019.

[54] F. Sabatino, "Quadrotor control: Modeling, nonlinearcontrol design, and simulation," M.S. thesis, KTH, Automatic Control, 2015, p. 61.

[55] R. Mitchell and I.-R. Chen, "A survey of intrusion detection techniques for cyber-physical systems," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 1–29, 2014.

[56] S. Mohan, S. Bak, E. Betti, H. Yun, L. Sha, and M. Caccamo, "S3a: Secure system simplex architecture for enhanced security of cyber-physical systems," *arXiv preprint arXiv:1202.5722*, 2012.

[57] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. Kumar, "The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures," in *28th IEEE International Real-Time Systems Symposium (RTSS)*, IEEE, 2007, pp. 400–412.

[58] X. Wang, N. Hovakimyan, and L. Sha, "L1simplex: Fault-tolerant control of cyber-physical systems," in *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, IEEE, 2013, pp. 41–50.

[59] M. S. Branicky, S. M. Phillips, and Wei Zhang, "Stability of networked control systems: Explicit analysis of delay," in *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, vol. 4, 2000, 2352–2357 vol.4. DOI: 10.1109/ACC.2000.878601.

[60] H. Gao, X. Meng, and T. Chen, "Stabilization of networked control systems with a new delay characterization," *IEEE Transactions on Automatic Control*, vol. 53, no. 9, pp. 2142–2148, 2008. DOI: 10.1109/TAC.2008.930190.

[61] H. Kwakernaak and R. Sivan, *Linear optimal control systems*. Wiley-interscience New York, 1972, vol. 1.

[62] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[63] G. He, Z. Chai, X. Lu, F. Kong, and B. Sheng, "Admm-based decentralized electric vehicle charging with trip duration limits," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, IEEE, 2019, pp. 107–119.

[64] B. He and X. Yuan, "On the $O(1/n)$ convergence rate of the douglas–rachford alternating direction method," *SIAM Journal on Numerical Analysis*, vol. 50, no. 2, pp. 700–709, 2012. DOI: 10.1137/110836936. eprint: https://doi.org/10.1137/110836936. [Online]. Available: https://doi.org/10.1137/110836936.

[65] X. Chen and S. Sankaranarayanan, "Model-predictive real-time monitoring of linear systems," in *IEEE Real-Time Systems Symposium (RTSS)*, IEEE Press, 2017, pp. 297–306.

[66] K. J. Åström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

[67] K. Tan and Y. Li, "Performance-based control system design automation via evolutionary computing," *Engineering Applications of Artificial Intelligence*, vol. 14, no. 4, pp. 473–486, 2001.

[68] B. Lu *et al.*, "Linear parameter-varying control of an f-16 aircraft at high angle of attack," 2005.

[69] F. Sabatino, *Quadrotor control: Modeling, nonlinear control design, and simulation*, 2015.

[70] J. Giraldo, A. Cardenas, and R. G. Sanfelice, "A moving target defense to detect stealthy attacks in cyber-physical systems," in *2019 American Control Conference (ACC)*, 2019, pp. 391–396. DOI: 10.23919/ACC.2019.8815274.

[71] F. Sabatino, "Quadrotor control: Modeling, nonlinear control design, and simulation," M.S. thesis, KTH Royal Institute of Technology, 2015.

[72] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *European Control Conference (ECC)*, 2013, pp. 3071–3076.

[73] K. Sridhar, R. Ivanov, V. Lesi, *et al.*, "A framework for checkpointing and recovery of hierarchical cyber-physical systems," *arXiv preprint arXiv:2205.08650*, 2022.

[74] R. Quinonez, J. Giraldo, L. Salazar, E. Bauman, A. Cardenas, and Z. Lin, "{Savior}: Securing autonomous vehicles with robust physical invariants," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 895–912.

[75] M. Fiacchini, T. Alamo, and E. Camacho, "On the computation of convex robust control invariant sets for nonlinear systems," *Automatica*, vol. 46, no. 8, pp. 1334–1338, 2010, ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2010.05.007.

[76] M. Berz, *Modern Map Methods in Particle Beam Physics* (Advances in Imaging and Electron Physics). Academic Press, 1999, vol. 108.

[77] K. Makino and M. Berz, "Rigorous integration of flows and ODEs using Taylor models," in *Proceedings of the 2009 conference on Symbolic numeric computation (SNC'09)*, ACM, 2009, pp. 79–84.

[78] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Taylor model flowpipe construction for non-linear hybrid systems," in *Proc. of RTSS'12*, 2012, pp. 183–192.

[79] X. Chen, "Reachability analysis of non-linear hybrid systems using taylor models," Ph.D. dissertation, RWTH Aachen University, 2015.

[80] X. Chen and S. Sankaranarayanan, "Decomposed reachability analysis for nonlinear systems," in *Proc. of RTSS'16*, 2016, pp. 13–24.

[81] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Proc. of CAV'13*, ser. LNCS, vol. 8044, 2013, pp. 258–263.

[82] A. J. Sørensen, "Marine cybernetics, towards autonomous marine operations and systems," *Department of Marine Technology, NTNU*, 2018.

[83] J. D. Hedengren, "A nonlinear model library for dynamics and control," *Yeast*, vol. 7, p. 24, 2008.

[84] A. Golabi, A. Erradi, and A. Tantawy, "Towards automated hazard analysis for cps security with application to cstr system," *Journal of Process Control*, vol. 115, pp. 100–111, 2022.

[85] R. Giorgiani do Nascimento, K. Fricke, and F. Viana, "Quadcopter control optimization through machine learning," in *AIAA Scitech 2020 Forum*, 2020, p. 1148.

[86] K. Sridhar and S. Sukumar, "Finite-time, event-triggered tracking control of quadrotors," in *5th CEAS Specialist Conference on Guidance, Navigation & Control (EurGNC 19) Milano, Italy*, 2019.

[87] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[88] M. Liu, L. Zhang, P. Lu, *et al.*, "Fail-safe: Securing cyber-physical systems against hidden sensor attacks," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 240–252. DOI: 10.1109/RTSS55097.2022.00029.

[89] L. Zhang, K. Sridhar, M. Liu, *et al.*, "Real-time data-predictive attack-recovery for complex cyber-physical systems," in *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2023.

[90] R. Rajamani, *Vehicle Dynamics and Control* (Mechanical Engineering Series). Springer US, 2011, ISBN: 9781461414339. [Online]. Available: https://books.google.com/books?id=cZJFDox4KuUC.

[91] J. M. Snider *et al.*, "Automatic steering methods for autonomous automobile path tracking," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

VITA

Lin Zhang was born in Liaoning, China. He enrolled at Dalian University of Technology in 2011 and graduated with a Bachelor of Engineering in Computer Science and Technology in 2015. Throughout his undergraduate studies, Lin earned several accolades in academic competitions related to electrical design and robotics. In 2015, Lin began his PhD studies at the University of Chinese Academy of Sciences, concentrating on vulnerability analysis for Internet of Things (IoT) infrastructures.

In 2019, Lin Zhang continued his PhD journey at Syracuse University under the guidance of Prof. Fanxin Kong, with a research focus on cyber-physical systems (CPS) security. He conducted extensive research in this domain, leading to the publication of numerous research papers on attack detection and recovery for CPS in prestigious venues, including RTSS, RTAS, EMSOFT, and DAC.

During his PhD, Lin received the Syracuse University fellowship and distinguished himself in various competitions. Some of his notable achievements include winning the Best Scientific Research Award at the ACM SIGBED Student Research Competition (SRC) in 2022, becoming the recipient of the Pramod K. and Anju Varshney Endowed Graduate Scholarship for the 2022-2023 academic year, securing first place in the Oral Presentation Competition at the 2022 ECS Research Day of Syracuse University, and claiming the Overall College Poster Prize at the 2020 ECS Research Day.