Recovery-Guaranteed Sensor Attack Detection for Cyber-Physical Systems

Weizhe Xu[1], Xin Chen[2], Matthew Anderson[3], Steve Drager[3], Fanxin Kong[1]
[1] Department of Computer Science and Engineering, University of Notre Dame, South Bend, IN
[2] Department of Computer Science, University of New Mexico, Albuquerque, NM
[3] Air Force Research Laboratory, Rome, NY
wxu3@nd.edu, chenxin@unm.edu, matthew.anderson.37@us.af.mil, steven.drager@us.af.mil, fkong@nd.edu

***Abstract*—Sensor attacks on Cyber-Physical Systems (CPS) can cause substantial damage in the physical world, which motivates two major threads of defense works including attack detection and attack recovery. The former aims to identify whether any sensors are compromised while the latter seeks to restore a system to safety once an attack is detected. Although either thread has drawn many efforts, how to coordinate the detection and recovery has been barely studied. Overlooking the coordination, existing works may result in ineffective and even failed defense. For example, if a detector raises an alarm too late, there may not be enough time for a system to recover but reach the unsafe region anyway, even though the detection result is accurate. By contrast, raising an alarm earlier allows more time for recovery, but may come with more false positives and thus unnecessarily trigger the recovery. To fill this gap, we aim to co-design attack detection and recovery, and propose a novel recovery-guaranteed sensor attack detection framework. The framework dynamically adjusts the detection sensitivity and authenticates state estimates at run time to guarantee timely and safe recovery once an attack is detected. The detection will always reserve sufficient time for the recovery while minimizing unnecessary activation of recovery. We conduct extensive simulations and real-world testbed experiments to show the efficiency of our solution.**

***Index Terms*—cyber-physical systems, recovery, detection, sensor attack, guarantee, online verification, reachability analysis**

I. INTRODUCTION

Cyber-Physical Systems (CPS) couple computing resources with physical processes through sensors and actuation components. CPS enables various applications such as autonomous vehicles, manufacturing systems, and power systems. For these life-critical CPS, security and safety are significant concerns. Due to their long working time, complex operational environments, and high connectivity, CPS is susceptible to various kinds of malicious attacks [1]–[3]. Among them, sensor attacks are particularly damaging and can cause serious safety issues. An attacker may manipulate sensor readings to mislead the controller to produce dangerous control commands. For instance, an attacker may corrupt the values of an altitude sensor to crush a drone [4] or modify position information to guide an autonomous vehicle to hit obstacles [5].

These potential consequences have driven many efforts to defending against sensor attacks in CPS. Among them, there are two major research threads: attack detection and attack recovery. Attack detection aims to identify if any sensors are compromised. Existing detection works usually compare the predicted and observed sensor values [6], [7], or leverage sensor redundancy [8], to make the decision. Attack recovery is triggered once an attack is detected, and aims to mitigate the impact caused by the attack. Usually, existing works design recovery methods that compute control demand sequences to steer a system back to a desired and safe state [9]–[15], by solving an optimization problem or training a recovery control policy. These works ensure to avoid the unsafe region while yielding a degraded control performance.

Although there are extensive efforts addressing attack detection and recovery separately, the coordination of the two parties has been barely studied. On one side, existing detection methods mainly seek to increase the accuracy of the detection result, while overlooking whether a system can be recovered to a safe state after the detection [7], [8], [16], [17]. On the other side, existing recovery methods usually assume a detector already in place and thus are subject to the accuracy of the detection [9]–[15]. That is, these recovery methods do not consider how the recovery should affect the detection. For example, for a false positive alarm by a detector, the recovery is unnecessarily triggered, while for a false negative, the recovery is not activated, but should be, and thus may cause safety violation. Therefore, we deem the importance of coordinating attack detection and recovery for an effective and safe defense against sensor attacks.

Coordinating attack detection and recovery is, however, non-trivial, due to both timing and functional challenges below.

i) Timing challenges. We consider two levels of timing requirements. The first is the global timing requirement, where after detecting an attack, it should be ensured that a system can be recovered before a safety deadline, after which the system may touch the unsafe region. This requires not only the recovery to be fast enough to drive a system back to safety, but also the detection reserves enough time to allow execution of the recovery. If a detector raises an alarm too late, there may not be enough time for a system to recover to a safe state even though the detection result is accurate. By contrast, if raising an alarm too early, much time is allowed for the recovery. However, the detector may have many false alarms and unnecessarily trigger the recovery, which lowers the

Approved for Public Release; Distributed Unlimited: PA cleared 26 November 2024 AFRL-2024-6649.

control performance. The other is the local timing requirement, where the detection and recovery need to be completed and outcome the computing result (i.e., the control demand) in a timely manner; otherwise, the control demand cannot be applied to the physical system in time, which will harm the control performance and even compromise safety.

ii) Functional challenges. First, we need to address an intricate interplay between detection and recovery for the coordination. On the one hand, a more sensitive detector usually comes with more false positives and thus more frequently triggers the recovery even though it is unnecessary [6], [18], [19]. This can ensure safety in a more conservative way but results in lower control performance. Note that a system operating in the recovery mode has degraded control performance compared to operating in the normal mode [9]–[15]. On the other hand, a less sensitive detector usually has fewer false positives but may incur more risk of safety violations. When such a detector raises an alarm, a system may have been significantly drifted off by an attack. For example, the system may have already reached the unsafe region or is too close to the region to be steered back. Hence, an effective coordinated defense should ensure that a system not only can be recovered after the detection of an attack but also trigger the recovery when truly necessary. Second, after detection, the attacked sensors are no longer trustworthy for estimating a system's physical state. However, trustworthy state estimates are needed for controlling the physical system during the recovery. This dilemma requires that a detector can not only identify attacks but also provide certain trustworthy data for the recovery to rely on.

To address these challenges, we propose a novel recovery-guaranteed sensor attack detection framework. In general, the framework dynamically adjusts the detection sensitivity and authenticates state estimates at run time to guarantee timely and safe recovery once an attack is detected. To be specific, the framework coordinates the detection and recovery by several components as follows. The *Residual Calculator* first predicts nominal sensor values and computes the difference or residual between them and sensor measurements. If the residual is greater than the current threshold given by the *Threshold Adjuster*, an alarm is raised to signify the occurrence of an attack. The adjuster determines the threshold based on the assessment given by the *Recoverability Calculator*, which online verifies whether a system can be timely and safely recovered in certain future time steps. The verification uses bounded state estimates given by the *Error Estimator*, which further relies on historical trustworthy state estimates authenticated by the *Authenticator*. The Authenticator only runs every few time steps to reduce the overhead.

Our technical contributions to the key components in our framework are as follows. 1) For the Authenticator, we propose an authentication algorithm to obtain trustworthy state estimates combined with estimation errors, which prevents the accumulative estimation error from becoming too large. 2) For the Recoverability Calculator, we design an incremental online verification algorithm to efficiently find a window of consecutive time steps by when a system can be recovered. 3) For the Threshold Adjuster, we develop an efficient algorithm to determine the alarm threshold based on the size of the window that is needed to maintain. 4) We comprehensively compare our proposed framework with existing methods on numerical simulations and a real-world testbed, showcasing its superior performance across key metrics.

The remainder of the paper is organized as follows. Section II presents the related works. Section III provides the preliminaries. Section IV overviews of our recovery-guaranteed sensor attack detection framework. Section V designs the core components while Section VI describes the supporting components of the framework. Section VII evaluates our solution. Section VIII concludes the paper.

II. RELATED WORK

A significant amount of research has been dedicated to the security of CPS, particularly in terms of defending against sensor attacks. One primary group of methods involves preventing sensor attacks through the use of sensor redundancy, such as determining vehicle speed with both encoders and GPS sensors.

The second group of methods for enhancing CPS security focuses on detecting sensor attacks through the use of physical invariants. A physical invariant is a property that remains unchanged under specific physical laws. Some methods [6], [11] use system models, often expressed as differential equations, to represent physical dynamics, while others [14], [20] leverage neural networks to extract and establish invariants. All these methods raise an alarm by comparing the difference (referred to as the residual) between the current system state predicted by the physical invariant and the system state obtained through sensor readings. We classify detectors into three types based on the amount of physical invariant data used. The first one is *memory-less detectors*, which only checks the current residual, and raises an alert when the residual is larger than the given threshold. The second one is *short-memory detectors*, such as window-based detectors. It monitors residuals within a certain detection window and alerts when the sum of residuals is larger than the given threshold. The third one is *long-memory detectors* which takes all historical data into account, such as the CUSUM (cumulative sum) detector and χ^2 detector [7]. For example, the CUSUM detector computes the CUSUM score of residuals over time. Neural network-based detectors can also be categorized as long-memory detectors. While existing detection methods vary in how they decide when to raise an alarm, they primarily focus on accuracy and false positive rates. However, they are not designed with consideration for the actions to be taken after the alert.

Currently, some researches focus on system recovery after an attack is detected, primarily categorized into three types. (1) Restart-based response [21], [22]: This approach reboots the system to eliminate attack impact. (2) Virtual sensors [23], [24]: the software uses a mathematical model to simulate sensor readings, bypassing compromised sensors. (3) Recovery controller [10], [11]: In addition to a normal controller, the system is equipped with a specialized recovery controller.

Upon detecting an attack, the system switches to this recovery controller to bring the system back to its target area.

Scope of Work. As mentioned earlier, existing methods of attack detection focus primarily on accuracy and false positive rate. However, these methods are not perfect as they overlook whether the system can recover when the alert is issued. High-sensitive detectors immediately trigger an alarm when attacked but may cause a large number of false alarms due to environmental noise and other factors. Low-sensitive detectors may alert when the system state has already deviated too far to be recovered. Therefore, it is essential to develop a detector capable of automatically adjusting sensor sensitivity and ensuring the system's successful recovery after raising an alert. This work aims to develop a detector that can guarantee recovery from sensor attacks without generating excessive false alarms. The discussion on methods for recovering from attacks falls outside the scope of this work.

The closest related work to this article is [6], which introduces an adaptive window-based sensor attack detector. However, this approach does not consider recovery feasibility in its design. This detector adjusts sensitivity based on the estimated time to an unsafe state, while our approach uses the system's recoverable time window, ensuring successful recovery after an alarm. Unlike their method, which relies on previous sensor data as trusted points, our approach estimates the real state range, avoiding reliance on potentially compromised sensor readings during stealthy sensor attacks.

III. PRELIMINARIES

A. Basic Definitions

A set of ordered variables or constants, e.g., $x_1, \ldots, x_n$, is denoted as a column vector $\boldsymbol{x}$. We use interval $[a, b]$ where $a \leq b$ to denote the set of all reals from a to b. Multidimensional intervals are denoted as $[\boldsymbol{a}, \boldsymbol{b}]$ such that $\boldsymbol{a}$ is the vector that consists of the lower bounds in all dimensions, while $\boldsymbol{b}$ consists of all upper bounds. We also call multidimensional interval boxes. We use $lo(X, i)$ to denote the lower bound of set X in the i-th dimension, while $lo(X)$ represents the vector formed by the lower bounds of set X in each dimension. Similarly, $up(X, i)$ and $up(X)$ correspond to the upper bounds.

We use zonotopes to represent the reachable sets of systems. Zonotopes are a subclass of symmetric bounded polyhedra [25]. A zonotope can be viewed as the image of a unit box under an affine mapping. We follow the generator-based definition used by Girard [26].

Definition III.1.

An n-dimensional zonotope Z *is defined by*

$$Z = \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \boldsymbol{x} = \boldsymbol{c} + \sum_{i=1}^{p} \lambda_i \boldsymbol{g}_i, \lambda_i \in [-1, 1] \right\}$$

where $\boldsymbol{c} \in \mathbb{R}^n$ is the center*, and $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_p \in \mathbb{R}^n$ are the* generators*. We denote Z as $(\boldsymbol{c}, \langle \boldsymbol{g}_1, \ldots, \boldsymbol{g}_p \rangle)$.*

Intuitively, a zonotope $Z = (\boldsymbol{c}, \langle \boldsymbol{g}_1, \ldots, \boldsymbol{g}_p \rangle)$ is the Minkowski sum of $\boldsymbol{c}$ and the line segments defined by the generators, i.e., the line segment defined by $\boldsymbol{g}_i$ is $\{\boldsymbol{x} \in \mathbb{R}^n \,|\, \boldsymbol{x} = \lambda_i \boldsymbol{g}_i, \lambda_i \in [-1, 1]\}$.

The Minkowski sum between zonotopes is easy to calculate following the definition: $X \oplus Y = \{x + y | x \in X, y \in Y\}$. Given two zonotopes $Z_1 = (\boldsymbol{c}_1, \langle \boldsymbol{v}_1, ..., \boldsymbol{v}_k \rangle)$ and $Z_2 = (\boldsymbol{c}_2, \langle \boldsymbol{g}_1, ..., \boldsymbol{g}_k \rangle)$, the result of their Minkowski sum is $Z_3 = (\boldsymbol{c}_1 + \boldsymbol{c}_2, \langle \boldsymbol{v}_1, ..., \boldsymbol{v}_k, \boldsymbol{g}_1, ..., \boldsymbol{g}_k \rangle)$.

Notation	Description
$\boldsymbol{x}$	Real system states
$\widetilde{\boldsymbol{x}}$	Sensors measurements
$\hat{\boldsymbol{x}}$	State predicted from system dynamics
$\boldsymbol{\theta} \in \Theta$	The error of $\hat{\boldsymbol{x}}$
$\boldsymbol{u} \in U$	The control inputs
$\boldsymbol{v} \in V$	Process and measurement noise in system
X_T	Target set
X_S	Safe set
$X_{j,d}$	The reachable set of the system at d-th step from time j
$\boldsymbol{r}$	Residual between $\hat{\boldsymbol{x}}$ and $\widetilde{\boldsymbol{x}}$
$\boldsymbol{\tau}$	Detector's threshold
w	Detector's observation window
$\oplus$	Minkowski sum, i.e., $X \oplus Y = \{x + y \| x \in X, y \in Y\}$
$\ominus$	Minkowski difference, i.e., $X \ominus Y = \bigcap_{y \in Y} (X - \{y\})$

TABLE I: Notations and symbols

B. System Model

The dynamics of a CPS considered in the paper is a linear time-invariant (LTI) model governed by a discrete-time feedback controller:

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

such that $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the system state and control input respectively at the time $t = i$, $\boldsymbol{v}_i$ is the process and environmental disturbance at that time, characterized as a value with range-bounded uncertainty, with $\boldsymbol{v}_i \in V$. We assume that all control inputs u are range-bounded due to their physical constraints, with $\boldsymbol{u} \in U$. Both U and V are boxes that are symmetric about the origin. We assume that all state variables are visible to sensors, therefore we can use the sensor readings to represent the state estimated based on sensor readings $\widetilde{\boldsymbol{x}}$. We also assume that matrix A is invertible.

C. Threat Model

In this article, we consider a CPS and an attacker that uses sensor attacks only. The attacker can modify the sensor readings $\widetilde{\boldsymbol{x}}_{t-1}$ which will then be sent to the controller. The attacker aims to drive the system state to an unsafe set using a sequence of sensor attacks, employing methods like bias attack [27], replay attack [28]–[30], and some other well-designed attack methods [31] [32]. Under these sensor attacks, the system's state $\widetilde{\boldsymbol{x}}_t$ estimated by sensors' measurement may have large deviations from the real system states $\boldsymbol{x}_{t-1}$, like $\widetilde{\boldsymbol{x}}_{t-1} \neq \boldsymbol{x}_{t-1}$. This can make the controller unable to produce the correct control command, that is $\Omega(\widetilde{\boldsymbol{x}}_{t-1}) \neq \Omega(\boldsymbol{x}_{t-1})$, where Ω defines the feedback law of the controller. Consequently, this may lead to the system state deviating from the safe set X_S, shown as $\boldsymbol{x}_t = A\boldsymbol{x}_{t-1} + B\Omega(\widetilde{\boldsymbol{x}}_{t-1}) + \boldsymbol{v}_t$.

D. Ordinary window-based detector

We use a similar idea to the window-based detector in our proposed recovery-guaranteed detector. Here, we briefly explain the principle of the window-based detector. It predicts the nominal sensor values $\hat{\boldsymbol{x}}_t$, using the previous sensor reading $\widetilde{\boldsymbol{x}}_{t-1}$, the equation is shown below

$$\hat{\boldsymbol{x}}_t = A\widetilde{\boldsymbol{x}}_{t-1} + B\boldsymbol{u}_{t-1} \tag{2}$$

It then collects the residual $\boldsymbol{r}_t$ between $\hat{\boldsymbol{x}}_t$ and $\widetilde{\boldsymbol{x}}_t$, and accumulates the previous residuals in its observation window W time steps, shown in equation (3).

$$\boldsymbol{r}_t = \hat{\boldsymbol{x}}_t - \widetilde{\boldsymbol{x}}_t, \quad \boldsymbol{r}_t^{\text{sum}} = \sum_{i=t-W}^{t} |\boldsymbol{r}_i| \tag{3}$$

The detector then compares the sum of residuals $\boldsymbol{r}_t^{sum}$ in its observation window to a threshold $\boldsymbol{\tau}$, and if it exceeds the threshold, an alarm is triggered. In this system, the threshold $\boldsymbol{\tau}$ is a vector, and the detector separately judges whether each dimension of $\boldsymbol{r}_t^{\text{sum}}$ exceeds the corresponding threshold.

E. Recovery controller

When a residual $\boldsymbol{r}_t^{\text{sum}}$ breaches a threshold, the system switches to the recovery controller immediately. Then the recovery controller tries to solve the following problem to find a control sequence that can safely steer the system to a target set.

Definition III.2 (Recovery control sequence [11])**.** *Given an initial state $\boldsymbol{x}_t$, a designated safe set X_S, and a target set $X_T \subseteq X_S$, alongside an LTI system as defined in Equation* (1)*, we define the concept of a recovery control sequence for the LTI system. At time step t, such a sequence of length d exists if it is capable of moving the system from $\boldsymbol{x}_t$ to X_T, staying within X_S.*

A recovery sequence should satisfy the following constraints:

$$\boldsymbol{x}_{t+d} \subseteq X_T \wedge \bigwedge_{i=t}^{t+d} \boldsymbol{x}_i \subseteq X_S \wedge \bigwedge_{i=t}^{t+d-1} (\boldsymbol{x}_{i+1} = A\boldsymbol{x}_i + B\boldsymbol{u}_i + \boldsymbol{v}_{i+1})$$

Hence, the *recoverability* at time step t is defined by whether or not such a sequence can be found.

Definition III.3 (Recoverability)**.** *We say that a system is recoverable at a state $\boldsymbol{x}_t$ if there exists a control sequence that can safely steer the system to a target state.*

As previously mentioned, our proposed detector considers not just the current step's recoverability. *Recoverable time window* is used to represent the recoverability over multiple consecutive future time steps.

F. Motivating example

We provide a motivating example in Fig. 1, where the left and right images depict the same CPS with different detectors. The green, yellow, and orange areas represent the target set, safe set, and unsafe set, respectively. The x-axis shows the time steps and the y-axis represents the system state. Before time t_0, the system operates normally. At time t_0, a sustained sensor attack begins, gradually shifting the system state from the target set, shown by the orange curve. In the left image, a more sensitive detector detects the attack at $t_0 + t_a$ and triggers recovery, allowing the recovery controller to bring the system back to the target set without entering the unsafe set, as shown by the blue curve. At time $t_0 + t_a$, the system is recoverable with a recoverable time window greater than 1. In the right image, however, the detector triggers an alarm at $t_0 + t_a + k$, by which time the system has deviated further, losing recoverability. During recovery, it enters the unsafe set, resulting in failed recovery. To prevent this, we propose a recovery-guaranteed sensor attack detector to ensure successful recovery upon detection.

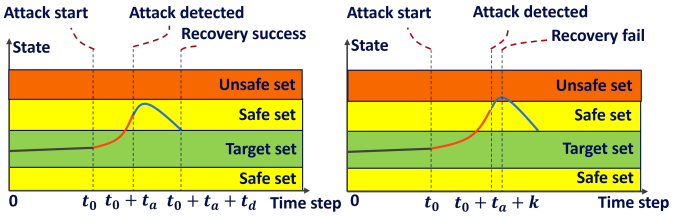


Fig. 1: A motivating example of the recovery-guaranteed detector

G. Problem statement

We consider a linear CPS described in Section III-B that experiences the sensor attack as outlined in Section III-C. Due to these attacks, the physical states deviate from their reference states. The detector then raises an alert and a recovery controller is activated to recover the system to the target set. The problem is to design a recovery-guaranteed detector that can dynamically adjust its threshold based on the system's recoverable time window. This guarantees that when the detector issues an alert, the system can safely recover to the target set. Additionally, to reduce false alarms caused by overly sensitive detectors, it is necessary to decrease the detector's sensitivity when the system's recoverable time window exceeds our predefined value.

Specifically, the problem can be formulated as the following optimization problem:

$$\max_{\Theta_t, \boldsymbol{u}} \ \boldsymbol{\tau}_t$$

$$s.t.\ \Theta_t = J(\hat{\boldsymbol{x}}, \widetilde{\boldsymbol{x}}, \boldsymbol{u}, \boldsymbol{\tau}_t, V) \tag{4a}$$

$$(X_t = \{\hat{\boldsymbol{x}}_t\} \oplus \Theta_t) \wedge (\boldsymbol{x}_t \in X_t) \tag{4b}$$

$$\bigwedge_{j=t+1}^{t+K} X_j = AX_{j-1} \oplus BU \oplus V \tag{4c}$$

$$\bigwedge_{j=t+1}^{t+K} (X_{j+d} \subseteq X_T \wedge \bigwedge_{i=j}^{j+d} X_i \subseteq X_S \wedge \bigwedge_{i=j+1}^{j+d} (X_i = AX_{i-1} \oplus \{B\boldsymbol{u}_{i-1}\} \oplus V)) \tag{4d}$$

where the goal is to maximize the detector's threshold $\boldsymbol{\tau}_t$ while maintain the given recoverable time window K. Θ_t indicates the estimation error between $\hat{\boldsymbol{x}}_t$ and $\boldsymbol{x}_t$. The function $J(\cdot)$ in Equation (4a) represents a nonlinear process for doing attack resilient state estimation to get Θ_t in each time step. As shown in Equation (4b), the set X_T contains all possible real system state $\boldsymbol{x}_t$. Equation (4d) indicates the sets of all possible real states of the system from time $j = t+1$ to $j = t+K$. K is the desired recoverable time window the system needs to maintain. Equation (4d) states that at each time from $j = t+1$ to $j = t+K$, a recovery control sequence exists, implying that the system maintains recoverability at each of these times. This indicates a recoverable time window of K.

This is not a simple problem since the constraints are complex and involve nonlinear processes. Moreover, the entire calculation must be completed within a single time step.

IV. System Design Overview

Our recovery-guaranteed sensor attack detection framework is shown in Fig. 2. Similar to the ordinary window-based detector we described in III-D, the recovery-guaranteed detector continuously monitors the system to identify sensor attacks by judging whether each dimension of the sum of residuals $\boldsymbol{r}_t^{sum}$ exceeds the threshold $\boldsymbol{\tau}_t$. Meanwhile, to ensure a successful recovery after an attack is detected, the detector adjusts its threshold $\boldsymbol{\tau}_t$ based on the system's recoverable time window K_t which is computed online. The five components: residual calculator, authenticator, error estimator, recoverability calculator, and threshold adjuster are explained as follows.

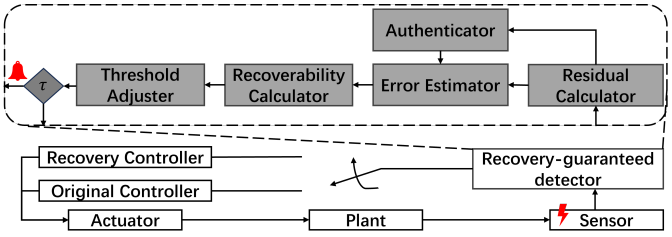


Fig. 2: The overview of system design

The **residual calculator** predicts nominal sensor value $\hat{\boldsymbol{x}}_t$ based on Equation (2), and calculates the residual $\boldsymbol{r}_t$ according to Equation (3).

The recoverable time window for the current system state is computed in the following way. Since the current sensor data does not indicate the actual system state due to the attack, we compute an overapproximate range X_t which is guaranteed to contain the actual system state. The set is obtained as $\{\hat{\boldsymbol{x}}_t\} \oplus \Theta_t$ wherein $\hat{\boldsymbol{x}}_t$ is the nominal sensor value and Θ_t is a bloating set containing all possible differences between the actual state and $\hat{\boldsymbol{x}}_t$. The **authenticator** and **error estimator** work cooperatively to obtain the set of Θ_t. The authenticator periodically computes the previous Θ_{t-p} by solving an optimization problem, where p is a parameter set by the user and is not less than the number of dimensions of the system's state. Using Θ_{t-p}, the latest result from the authenticator, as a starting point, the error estimator infers the current Θ_t based on the previous time step's Θ_{t-1} and the detector's threshold $\boldsymbol{\tau}_t$ at each time steps. Once the authenticator completes calculation, the error estimator updates its starting point based on this data.

The **recoverability calculator** leverages reachability analysis to estimate the system's recoverable time window K_t based on X_t. It employs an *incremental online verification method* to assess the recoverability of future time steps, thereby determining the recoverable time window. With this current system's K_t, the **threshold adjuster** changes the threshold τ_t of the detector, ensuring that the system's recovery time tolerance K_t is not less than a user-given size K while maximizing the threshold $\boldsymbol{\tau}_t$. Finally, the detector evaluates whether the residuals breach the threshold $\boldsymbol{\tau}$ and provides this as the final result of the recovery-guaranteed detector. By maintaining the system's recoverable time window, our method ensures that the system can successfully recover when the detector alerts.

V. Reacoverability Calculator

This section details the recoverability calculator, the core component of our framework.

To calculate the current system's recoverable time window K_t, we need to assess the system's recoverability at future time steps. The task of checking recoverability is simplified to detecting the emptiness of the intersection between a general zonotope and a box. Such a problem can be addressed using linear programming (LP). However, as an alternative, we propose a method with a time complexity of $O(kn)$ to approximate the results, where k represents the number of generators the zonotope has, and n is the number of dimensions. This approach provides a more efficient means of handling the problem. In addition, we employ an incremental online verification method to assess the recoverability of future time steps. That is, the recoverability verification at a time can be done based on the recoverability at the previous time.

At time step t, the recoverability calculator receives the nominal sensor values $\hat{\boldsymbol{x}}_t$, and the estimated error Θ_t, then calculates the recoverable time window K_t.

To calculate the system's recovery time tolerance at time t, we need to check the system's recoverability at future time $t+1, ..., t+K$, where K is the desired length of recoverable time window the system wants to maintain. If all these times, the system maintains recoverability, then the system keeps a recoverable time window of length K.

The recoverability at a specific time j, where $t+1 \leq j \leq t+K$ can be computed in the following way. To ensure successful system recovery, it is essential to consider all possible values of the system's real states at time j. We represent the range of values at the current time step t with X_t, as illustrated in the equation $X_t = \{\hat{\boldsymbol{x}}_t\} \oplus \Theta_t$. Then all the possible values of the system's real states in time j can be represented as follows. For clarity, we set $t = 0$ for the remainder of this section.

$$X_j = A^j \boldsymbol{x}_0 + \sum_{i=1}^{j} A^{j-i} B \boldsymbol{u}_{i-1} + \sum_{i=1}^{j} A^{j-i} \boldsymbol{v}_i$$
$$(\boldsymbol{x}_0 \in X_0, \boldsymbol{u}_0, ..., \boldsymbol{u}_{j-1} \in U, \boldsymbol{v}_1, ..., \boldsymbol{v}_j \in V)$$

Then, we employ a heuristic to identify all control sequences that meet two requirements: (1) steer the system toward the target set and (2) maintain the system's safety from any state within X_j. To do so, we employ a method similar to the real-time monitoring technique used in [33]. The reachable set for all $d = 1, 2, ...t_d$-th step from time j is shown below. We will discuss how to determine the deadline t_d later.

$$X_{j,d} = A^d \boldsymbol{x}_j + \sum_{i=1}^{d} A^{d-i} B \boldsymbol{u}_{j-1+i} + \sum_{i=1}^{d} A^{d-i} \boldsymbol{v}_{j+i}$$
$$(\boldsymbol{x}_j \in X_j, \boldsymbol{u}_j, ..., \boldsymbol{u}_{j+d-1} \in U, \boldsymbol{v}_{j+1}, ...\boldsymbol{v}_{j+d} \in V)$$

where the subscript of $X_{j,d}$ represents the d-th step from time j.

As for steering the system toward the target set from any state within X_j, the reachable set $X_{j,d}$ should inevitably interact with the target set X_T with the recovery control sequence $\boldsymbol{u}_j, ..., \boldsymbol{u}_{j+d-1} \in U$ for any noise $\boldsymbol{v}_{j+1}, ...\boldsymbol{v}_{j+d} \in P$. To do so, we check the emptiness of the intersection between the control envelope

$$E_{j,d} = \bigoplus_{i=1}^{d} A^{d-i} BU \tag{5}$$

which contains all possible control effect at the d-th step from time j, and the Minkowski difference $D_{j,d} = X_T \ominus L_{j,d}$ where

$$\begin{aligned} L_{j,d} &= A^d X_j \oplus \bigoplus_{i=1}^{d} A^{d-i} V \\ &= \{A^{j+d} \hat{\boldsymbol{x}}_0\} \oplus A^{j+d} \Theta_0 \oplus \bigoplus_{i=1}^{j} A^{j+d-i} BU \oplus \bigoplus_{i=1}^{j+d} A^{j+d-i} V \end{aligned} \tag{6}$$

which is a constrained target set by the accumulation of uncertainties.

Proposition V.1 ([10], [33]). *Given $0 \leq d \leq t_d$, if the intersection $E_{j,d} \cap D_{j,d}$ is not empty, then there exists a recovery control sequence to work on all states in X_j.*

We want to check if $E_{j,d} \cap D_{j,d} \neq \emptyset$. To do so, we seek to find a point that lies in both $E_{j,d}$ and $D_{j,d}$. Such a point exists if and only if the intersection of the two sets is nonempty. Although $D_{j,d}$ is a Minkowski difference and computing a Minkowski difference of two sets is a difficult task, the Minkowski difference between a box and a zonotope is a box [33] and therefore computing $D_{j,d}$ only requires to calculate its upper and lower bounds in all dimensions. Hence, the emptiness checking problem becomes to find a point E^* in the intersection of a zonotope and a box. Instead of locating a random point in the intersection set $E_{j,d} \cap D_{j,d}$, we aim to find a point as close as possible from the zonotope $E_{j,d}$ to the center of the box $D_{j,d}$. Identifying a point as close as possible to the center of the box will enhance the accuracy of our subsequent incremental recoverability verification method. The details will be given in Section 16.

Given a zonotope $E = (\boldsymbol{c}_1, \langle \boldsymbol{v}_1, ..., \boldsymbol{v}_k \rangle)$ and a box $D : [lo(D), up(D)]$, we need to compute a group of scalars $\alpha_1, ..., \alpha_k \in [-1, 1]$ to find a point $E^* = \alpha_1 \boldsymbol{v}_1 + ... + \alpha_k \boldsymbol{v}_k$, which is as close as possible to the center of D among all points in E. The reason to do so is to make the obtained *recoverable time window* from Algorithm 2 as large as possible. This problem can also be solved using LP, however, we propose Algorithm 1 which only requires $O(kn)$ operations of real arithmetic where k is the number of generators and n is the number of dimensions. It returns an approximate point which is as close as possible to the center of D and sufficient to be used in checking the recoverability of the following time steps. The algorithm starts with the center of the zonotope E and explores toward the center of the box D until it finds the first point in the intersection.

In line 1, E^* is initialized to the center $\boldsymbol{c}_1$ of the zonotope E. $\boldsymbol{c}_2$ is the target point of exploration, which is the center of box D. $flag$ indicates whether sets E and D intersect, initialized to 0. The algorithm loops through each generator to explore towards $\boldsymbol{c}_2$. In line 3, t represents the projection of the line connecting the current exploration point E^* and the center of the box $\boldsymbol{c}_2$ onto the generator $\boldsymbol{v}_i$. It indicates the scale that the generator $\boldsymbol{v}_i$ should take to reduce the distance to $\boldsymbol{v}_2$. However, the scale α_i should be in the range of -1 to 1. In lines 4-10, we obtain $\alpha_i \in [-1, 1]$ from t. Then in line 11, the algorithm uses $\alpha_i \boldsymbol{v}_i$ to update E^* and explore towards $\boldsymbol{c}_2$. In lines 12-14, Set the $flag$ to 1 if the current E^* is already in the box D. It is important to note that the algorithm does not stop upon finding a point E^* inside D, instead, it continues to explore towards the center of the box to find a point as close as possible to the center. After utilizing all the generators, the algorithm terminates and returns E^* as the approximate closest point, the corresponding $\alpha_1, ..., \alpha_k$, and $flag$ which indicates intersection or not.

Algorithm 1: Finding an as-close-as-possible point.

Data: $E = (\boldsymbol{c}_1, \langle \boldsymbol{v}_1, ..., \boldsymbol{v}_k \rangle)$, $D : [lo(D), up(D)]$
Result: E^*, $\alpha_1, \alpha_2, ...\alpha_k$, $flag$

1 $E^* \leftarrow \boldsymbol{c}_1$, $\alpha_1, ..., \alpha_k \leftarrow 0$,
  $\boldsymbol{c}_2 \leftarrow (lo(D) + up(D))/2$, $flag = 0$;
2 **for** $i = 1$ *to* k **do**
3 $t \leftarrow \frac{(\boldsymbol{c}_2 - E^*) \cdot \boldsymbol{v}_i}{\boldsymbol{v}_i \cdot \boldsymbol{v}_i}$;
4 **if** $t > 1$ **then**
5 $\alpha_i \leftarrow 1$;
6 **else if** $t < -1$ **then**
7 $\alpha_i \leftarrow -1$;
8 **else**
9 $\alpha_i \leftarrow t$;
10 **end**
11 $E^* \leftarrow E^* + \alpha_i \boldsymbol{v}_i$;
12 **if** $lo(D) \leq E^* \leq up(D)$ **then**
13 $flag \leftarrow 1$;
14 **end**
15 **end**
16 return E^*, $\alpha_1, ..., \alpha_k$, $flag$;

Proposition V.2. *In each iteration in Algorithm 1, the update in Line 11 makes E^* closer to the center of D.*

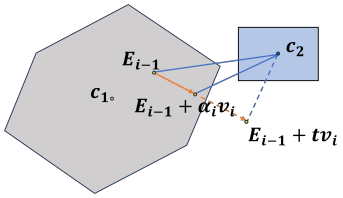


Fig. 3: Update of an as-close-as-possible point.

Proof. At the beginning of the i-th iteration in Algorithm 1, the as-close-as-possible point is $E^* = \boldsymbol{c}_1 + \alpha_1 \boldsymbol{v}_i + \cdots + \alpha_{i-1}\boldsymbol{v}_{i-1}$. We denote it by E_{i-1}. Then the modification term $\alpha_i \boldsymbol{v}_i$ is computed based on $t\boldsymbol{v}_i$ such that α_i is the truncated value of t with respect to the range $[-1, 1]$. We illustrate the relations among these terms in Fig. 3. Since vectors $t\boldsymbol{v}_i$ and $\boldsymbol{c}_2 - E_{i-1} - t\boldsymbol{v}_i$ is orthogonal, we can conclude that the line segment from E_{i-1} to $\boldsymbol{c}_2$ is longer than the line segment from $E_{i-1} + \alpha_i \boldsymbol{v}_i$ to $\boldsymbol{c}_2$, and hence the update in Line 11 makes E^* closer to $\boldsymbol{c}_2$ which is the center of D. □

Calculation of deadline. As for maintaining the system's safety from any state within X_t, the equation $X_{j,d} \subseteq X_S$ confirms that the system will not reach any unsafe state at the d-th step from the time j.

Utilizing the work on deadline estimation given in [11], we set $\boldsymbol{u}_j, ...\boldsymbol{u}_{j+d-1} = [0, ..., 0], 1 \leq d \leq t_d$ to calculate the deadline t_d. A departure from the safe set occurs only if the current system state cannot be rectified through any recovery control sequence. Therefore, the above equation can be expressed as $L_{j,d} \subseteq X_S$.

Proposition V.3 (Deadline [11])**.** *At the time j, the system will not reach any unsafe state at d-th step if $L_{j+d} \subseteq X_S$.*

Therefore, the maximum d that ensures $L_{j,d} \subseteq X_S$ is t_d. The recovery process must be completed in t_d-th steps. To check whether $L_{j,d} \subseteq X_S$, we first use the support function to get the maximum and minimum values in each dimension of the zonotope $L_{j,d}$, and then compare it with the boundary of the box X_S.

Incremental online recoverability verification. We propose an incremental online verification method for getting the recoverable time window K_t. That is, the recoverability verification at a time can be done based on the recoverability at the previous one.

For illustration purposes, we use the following situation as an example to introduce our incremental online recoverability verification method, and similarly, we set $t = 0$. In this example, we have already checked the recoverability at j and we are going to check the recoverability at $j + 1$. Compare the cases when checking whether the control envelope $E_{j,d}$ and constrained target set $D_{j,d} = X_T \ominus L_{j,d}$ are empty at d-th step from time j and at $d - 1$th step from time $j + 1$.

We can find the following equations from (5) (6).

$$L_{j,d} \oplus A^{d-1}BU = L_{j+1,d-1},\ E_{j,d} = E_{j+1,d-1} \oplus A^{d-1}BU$$

Since U is a box symmetric with respect to the origin, $A^{d-1}BU$ is a zonotope that is also symmetric about the origin. Then the centers of $D_{j,d}$ and $D_{j+1,d-1}$ are the same point, and the center of $E_{j,d}$ and $E_{j+1,d-1}$ are both the original point. Assume the generators of $E_{j,d}$ is $\boldsymbol{m}_1, ..., \boldsymbol{m}_{dn}$, then the generators of $E_{j+1,d-1}$ is $\boldsymbol{m}_1, ..., \boldsymbol{m}_{dn-n}$, where n is the number of the system state variables. Therefore, since algorithm 1 explores from $E_{j+1,d-1}$ and $E_{j,d}$ towards the same target point, we can incrementally obtain the point in $E_{j+1,d-1}$ that is closest to the center of $D_{j+1,d-1}$. It is the point in the exploration path of $E_{j,d}$ that has traversed all the generators of $E_{j+1,d-1}$. If the group of scalars for the closest point in E is $\alpha_1, ..., \alpha_{dn}$, then the closest point in E' is $\alpha_1 \boldsymbol{m}_1, ..., \alpha_{dn-n}\boldsymbol{m}_{dn-n}$. Therefore, we can do incremental online verification for recoverability.

Lemma V.1. *Given a zonotope $E_1 : (\boldsymbol{c}, \langle \boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_{dn} \rangle)$, where the as-close-as-possible point to the center of box D calculated by Algorithm 1 is $\boldsymbol{c} + \alpha_1 \boldsymbol{m}_1 + \cdots + \alpha_{dn}\boldsymbol{m}_{dn}$. The zonotope $E_2 : (\boldsymbol{c}, \langle \boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_{dn-n} \rangle)$ which is same as E_1 except that the last n generators are removed. Then the as-close-as-possible point in E_2 to the center of D can be calculated as $\boldsymbol{c} + \alpha_1 \boldsymbol{m}_1, ..., \alpha_{dn-n}\boldsymbol{m}_{dn-n}$.*

The overall process for calculating recoverability can be represented as Algorithm 2, as shown below:

Algorithm 2: Calculate the recoverable time window

Data: $\hat{\boldsymbol{x}}_0, \Theta_0$
Result: $adjust, closest, recoverability, K_0$

```
adjust ← False;
Calculate the deadline t_d for X_1;
for d = 1 to t_d do
    Calculate E_{1,d} and D_{1,d};
    E*, α_1, ..., α_k, flag ← Algorithm 1;
    closest[1][d] ← E*, α_1, ..., α_k;
    empty[1][d] ← flag;
    if flag = 1 then
        recoverability[1] ← 1,  K_0 = 1;
    end
end
for j = 2 to K + 1 do
    Calculate the deadline t_d for X_j;
    for d = 1 to t_d do
        Calculate D_{j,d};
        closest[j][d] ← Get(closest[j − 1][d + 1]);
        empty[j][d] ← InBox(closest[j][d], D_{j,d});
        if empty[j][d] = 1 then
            recoverability[j] ← 1,  K_0 = K_0 + 1;
        end
    end
end
if K_0 ≠ K then
    adjust ← True;
end
return adjust, closest, empty, recoverability, K_0;
```

At time step $t = 0$, this algorithm takes the nominal sensor values $\hat{\boldsymbol{x}}_0$ and the estimation error Θ_0 as input data. As for the return, the value $adjust$ indicates whether the recovery time tolerance K_0 meets the required one K. The list $closest$ represents the groups of scalars in set $E_{j,d}$ that is as close as possible to the center of the box $D_{j,d}$. The list $recoverability$

indicates the recoverability status. From line 2 to line 11, we compute the recoverability at time $t+1$. In line 5, it checks the intersection between $E_{1,d}, D_{1,d}$ by leveraging Algorithm 1. If the intersection set is not empty, the system maintains recoverability at time 1 and we record it in line 8. Starting from line 12, we check the system's recoverability from 2 to $K+1$. In line 16, the function $Get(\cdot)$ indicates getting the as-close-as-possible point at d-th step from time j based on the closest point at $d+1$-th step from time $j-1$. Then in line 17, we check whether the closest point is in the box of $D_{j,d}$. In lines 23-25, we determine whether it is necessary to adjust the current system's recoverable time window K_0.

It is important to note that our framework is based on reachability analysis, allowing us to guarantee that when the detector raises an alarm, at least one recovery control sequence is capable of safely recovering the system. Therefore, our framework is well-suited for recovery algorithms that can compute and execute this recovery control sequence, such as those proposed in [10], [11], [13].

VI. Supporting components

In this section, we give a detailed explanation of three key components of our framework: the authenticator, error estimator, and threshold adjuster. We first describe how the authenticator and error estimator work together to compute the estimated error set Θ_t for the nominal sensor value $\hat{\boldsymbol{x}}_t$ at each time step, which is the input of the recoverability calculator.

Attack-resilient state estimation, which determines the range of a system's real state under potential sensor attacks, has been extensively studied [34], [35]. These studies typically involve solving a time-consuming Mixed Integer Linear Program (MILP) problem. The computation often spans at least several tens of milliseconds, which fails to meet our requirement for estimating the real system state at each time step.

We first implement an authenticator to get an accuracy estimation range for previous real state x_{t-c-p}, where c is the computation time, and p is the authentication window for data collection. It employs optimization techniques with each calculation taking longer than one time step, specifically, $c > 1$. In addition, we develop a low-overhead error estimator that infers the current real state range from the previous time step's range. This estimator operates within each time step and uses the latest authenticator results as its starting point. Fig. 4 illustrates the interaction between the authenticator and error estimator. The black line represents the actual state of the system. Starting from time $t-c-p$, the error estimator calculates the range of the real system state at each time step, depicted as the yellow line. Meanwhile, the authenticator determines the precise range of x_{t-c-p} by solving an optimization problem, completing its calculation at t. These results then update the error estimator's start point, shown as the green line, significantly reducing the estimation error for the real system state at time t.

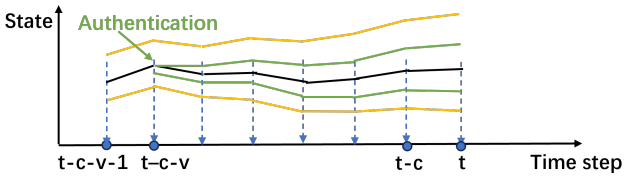


Fig. 4: An example of authenticator and error estimator

A. Authenticator

The authenticator uses recorded system data, including $\boldsymbol{u}$, $\hat{\boldsymbol{x}}$ and $\widetilde{\boldsymbol{x}}$, to output the error Θ in the nominal sensor value $\hat{\boldsymbol{x}}$. We formulate a series of MILP problems under the same assumptions as those in [35] to determine the range of actual system states. This enables straightforward calculation of the error bounds for state estimation.

For demonstration purposes, we assume that the current time step is p. The system states at time steps $0, 1, ..., p$ as $\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_p$. Equation (7) is the evolution of a real linear system from time 0 to p.

$$\widetilde{\boldsymbol{x}}_0 = \boldsymbol{x}_0 + \boldsymbol{a}_0, \ \ldots, \widetilde{\boldsymbol{x}}_p = A\boldsymbol{x}_{p-1} + B\boldsymbol{u}_{p-1} + \boldsymbol{v}_p + \boldsymbol{a}_p \quad (7)$$

Then we define $\widetilde{X}$ to express the effect of $\widetilde{x}$, $\Phi(\boldsymbol{x}_0)$ to express the effect of $\boldsymbol{x}_0$ and u, E to indicate sensor attacks, and Δ^* to represent the actual effect of noise v.

$$\widetilde{X} = (\widetilde{\boldsymbol{x}}_0, \widetilde{\boldsymbol{x}}_1, ..., \widetilde{\boldsymbol{x}}_p)$$

$$\Phi(\boldsymbol{x}_0) = (\boldsymbol{x}_0, A\boldsymbol{x}_0 + B\boldsymbol{u}_0, ..., A^{p-1}\boldsymbol{x}_0 + \sum_{i=0}^{p-1} A^i B\boldsymbol{u}_{p-1-i})$$

$$E = (\boldsymbol{a}_0, \boldsymbol{a}_1, ..., \boldsymbol{a}_p), \ \Delta^* = (0, A\boldsymbol{v}_0 + \boldsymbol{v}_1, ..., \sum_{i=0}^{p} A^i \boldsymbol{v}_{p-i})$$

where each element, like $\widetilde{\boldsymbol{x}}_0$, is a column of the matrix $\widetilde{X}$. All subsequent $(, ...,)$ here means the combination of the column vectors. By replacing the noise in the Δ^* with the upper bound of the noise, we get the new $\Delta = (up(V), Aup(V) + up(V), ..., \sum_{i=0}^{p-1} A^i up(V))$. where $lo(V)$ and $up(V)$ represent the vector formed by the lower or upper bounds of set V in each dimension. Then the constraint: $|\widetilde{X} - \Phi(\boldsymbol{x}_0) - E| \leq \Delta$ should always hold. which means the system noise cannot exceed the upper bound. Based on the assumption that attackers tend to attack fewer sensors to achieve their desired impact, researchers [35] estimate the previous system state x_0 by solving the following optimization problem:

$$\min_{E, \boldsymbol{x}_0} \| E \|_{l_0}, \ s.t. |\widetilde{X} - \Phi(\boldsymbol{x}_0) - E| \leq \Delta \quad (8)$$

where $\| E \|_{l_0}$ represents the number of non-zero row vectors in matrix E. This problem can be straightforwardly transformed into an MILP problem. The best results are obtained when $n \leq p \leq 2n$ holds, where n is the dimension of system state x. We denote the result of problem (8) as x_0^* and E^*.

We extend this optimization problem to calculate the estimation error range for $\boldsymbol{x}^*$. We set the real system value as $\boldsymbol{x}_0$ and the real sensor attack as E, which are also the feasible point of the optimization problem (8) above. Hence we have

the following optimization problem to get the error bound, based on the same assumption.

$$\max_{E,\boldsymbol{x}_0} |\boldsymbol{x}_0[i]-\boldsymbol{x}_0^*[i]|, s.t.|\widetilde{X}-\Phi(\boldsymbol{x}_0)-E| \leq \Delta, \ \| E \|_{l_0} \leq \| E^* \|_{l_0}$$

where $\boldsymbol{x}_0[i]$ refers to the value of $\boldsymbol{x}_0$ in the i-th dimension. The object is to maximize the deviation of $\boldsymbol{x}_0^*$ from $\boldsymbol{x}_0$ in the i-th dimension. Similarly, this problem can also be reduced to an MILP problem. With $\boldsymbol{x}_0^*$ and the error range calculated above, we can easily get the corresponding Θ_0 for $\hat{\boldsymbol{x}}_0$. Then update the start point of the error estimator.

B. Error estimator

Using the latest authenticator results as the starting point, an error estimator is designed to infer the current range of the real system state from the previous time step's range.

We initially utilize the information from whether the detector triggers an alarm. The inequation $\sum_{i=t-W}^{t} |\hat{\boldsymbol{x}}_i - \widetilde{\boldsymbol{x}}_i| < \boldsymbol{\tau}$ holds when the detector is not alerted at $t-W$ to t time steps, where W is the observation window of the detector. Then, we can express $\hat{\boldsymbol{x}}$ in terms of $\boldsymbol{x}$ and $\widetilde{\boldsymbol{x}}$ in equation below, by subtracting Equation (1) from Equation (2).

$$\hat{\boldsymbol{x}}_t = \boldsymbol{x}_t + A\widetilde{\boldsymbol{x}}_{t-1} - A\boldsymbol{x}_{t-1} + \boldsymbol{v}_t, \ \boldsymbol{v}_t \in V$$

Following this, we can construct an inequality involving $\boldsymbol{x}_t$.

$$|\boldsymbol{x}_t + A\widetilde{\boldsymbol{x}}_{t-1} - A\boldsymbol{x}_{t-1} + \boldsymbol{v}_t - \widetilde{\boldsymbol{x}}_t| < \boldsymbol{\tau} - \sum_{\text{i=t-W}}^{\text{t-1}} |\hat{\boldsymbol{x}}_i - \widetilde{\boldsymbol{x}}_i|, \ (\boldsymbol{v}_t \in V)$$

We obtain the following inequality by substituting $\boldsymbol{\theta}$ for $\boldsymbol{x}$.

$$\begin{aligned} |\boldsymbol{\theta}_t| < \boldsymbol{\tau} - \sum_{\text{i=t-w}}^{t} |\hat{\boldsymbol{x}}_i - \widetilde{\boldsymbol{x}}_i| + A(\hat{\boldsymbol{x}}_{\text{t-1}} - \widetilde{\boldsymbol{x}}_{\text{t-1}}) + A\boldsymbol{\theta}_{\text{t-1}} + \boldsymbol{v}_t \\ (\boldsymbol{\theta}_{t-1} \in \Theta_{t-1}, \boldsymbol{v}_t \in V) \end{aligned} \tag{9}$$

One important thing to note is that when calculating the value of $A\boldsymbol{\theta}_{t-1}$, we need to select the upper or lower bound of Θ_{t-1} for computation according to the sign of the elements in A, to ensure an overestimation of Θ_t.

From Inequation (9), it is obvious that the size of Θ_t monotonically grows along with time. It will eventually lead to an estimation that is trivially large. However, our authenticator recalculates/resets the estimation Θ_t periodically and that helps to prevent the explosion of the set.

C. Threshold adjuster

In this part, we present the design of the threshold adjuster. It receives the recoverable time window K_t, the as-close-as-possible points, and the recoverability status in each future time j, where $t+1 \leq j \leq t+K+1$, calculated by the recoverability calculator, as input. It adjusts the detector's threshold $\boldsymbol{\tau}_t$ to ensure that recoverable time window is equal to K. The detector uses this new threshold to evaluate the alarm condition and provides this as the final results of the recovery-guaranteed detector.

To adjust the system's recoverable time window K_t, we need to determine how to modify the threshold for a possible set of real states X_j, to ensure the system gains or loses recoverability at time j.

According to the previous section, since Θ_t is related to $\boldsymbol{\tau}_t$, by adjusting the threshold $\boldsymbol{\tau}_t$, we can alter the size of constrained target set $D_{j,d}$, which may change the result of whether the sets $E_{j,d}$ and $D_{j,d}$ have a non-empty intersection, thereby affecting recoverability.

According to Inequality (9), the change in Θ_t under the adjustment of the threshold is illustrated by the formula below.

$$up(\Theta_t') = up(\Theta_t) + \Delta\boldsymbol{\tau}, \ lo(\Theta_t') = lo(\Theta_t) - \Delta\boldsymbol{\tau}$$

where Θ_t' indicates the Θ_t after threshold adjustment, τ' is the new threshold, and $\Delta\tau$ is the adjustment, $\tau' = \tau + \Delta\boldsymbol{\tau}$. Then, we get the new box $D_{j,d}'$:

$$up(D_{j,d}') = up(D_{j,d}) - A^{j+d}\Delta\boldsymbol{\tau}, lo(D_{j,d}') = lo(D_{j,d}) + A^{j+d}\Delta\boldsymbol{\tau} \tag{10}$$

where $D_{j,d}'$ indicates the new $D_{j,d}$ after adjustment.

It can be easily observed that the centers of the box $D_{j,d}'$ and $D_{j,d}$ are the same. Given Equation (10) and the as-close-as-possible point from zonotope $E_{j,d}$ to the center of the box $D_{j,d}$, we can easily adjust the threshold $\boldsymbol{\tau}_t$ to ensure that the intersection of the two sets is non-empty. The adjustment $\Delta\boldsymbol{\tau}$ can be calculated by $\Delta\boldsymbol{\tau} = A^{j+d^{-1}}dist$, where $dist$ is the distance from the closest point to the box in each dimension, and we already assume that A is invertible.

VII. EVALUATION

In this section, we compare the performance of our proposed recovery-guaranteed sensor attack detector with other previous works. Moreover, further analysis will be conducted.

A. Experimental settings

The experiments are tested under three numerical simulations [6], [10], [36] and a 4-wheel testbed.

1) Hardware and software configurations: The experiments for numerical simulation are conducted on a lab computer equipped with a 12th Gen Intel(R) Core(TM) i5-12400F CPU operating at 2.5GHz, along with 16 GB of RAM. Our code is written in Python and utilizes the cvxpy library with CBC solver to solve optimization problems.

2) Numerical simulations: We implement and test our framework on three numerical simulators: vehicle platoon [36], aircraft pitch [10], and lane keeping [6]. All these three simulators are LTI models that get from linearization and discretization, representing typical models used in security and control research [10], [11], [35].

Vehicle platoon. In this system, four vehicles maintain a stable distance and form a platoon. The system has 7 state variables, from x_0 to x_6, and 4 control input variables, from u_0 to u_3. Each vehicle uses the onboard distance sensors (e.g. lidar) to obtain the distance from the front vehicle as states x_0 to x_2. For example, x_0 reflects the distance between the front vehicle and the one behind. The system also measures the velocity of the four vehicles as states x_3 to x_6. The control inputs u_0 to u_3 correspond to the acceleration of four vehicles.

	Platoon	Lane Keeping	Aircraft Pitch
Target set X_T	$x_0 \in$ [0.4, 1.2]	$x_0 \in$ [-0.5, 0.5]	$x_2 \in$ [-0.7, 0.7]
Safe set X_S	$x_0 \in$ [0.0, 5]	$x_0 \in$ [-0.5, 0.5]	$x_2 \in$ [-2, 2]
Control inputs $\boldsymbol{u}$	[-2, 2]	[-0.26, 0.26]	[-20, 20]
Noise $\boldsymbol{v}$	[-0.002, 0.002]	[-0.005, 0.005]	[-0.01, 0.01]
Control stepsize	40	40	40

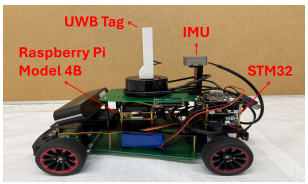
TABLE II: Simluation Scenarios. Time unit: ms(millisecond).

Aircraft pitch. The object of this system is to control the pitch angle of the aircraft. It has 3 state variables x_1, x_2, x_3, and one control input. In this framework, x_1 symbolizes the angle of attack, x_2 represents the pitch rate, and x_3 corresponds to the pitch angle. The control input denoted as $\boldsymbol{u}$, is the elevator deflection angle.

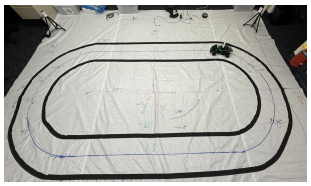
Lane keeping. This system considers the vehicle to perform lateral control on the track with a pre-planning path. The vehicle in this system is modeled using a kinematic bicycle framework [37]. We conducted simulations of an autonomous driving system that utilizes the Stanley method [5] to govern vehicular motion. The system has 4 state variables from x_0 to x_3 and 1 control input. The reference of the system is to ensure that the vehicle travels along the predetermined road.

Other simulation settings are listed in Table. II.

3) 4-wheel testbed: As shown in Fig. 5a, the 4-wheel testbed comprises an STM32 board, a Raspberry Pi Model 4B, a motor, and a servo. The STM32 board includes a UWB positioning module, 9-axis IMU, and speed sensor to monitor position, yaw angle, and speed. Two PID controllers manage vehicle speed and turning. The recovery-guaranteed detector is deployed on the same lab computer we used for numerical simulations. We use the ROS2 framework for the communication between the boards and the desktop. The STM32 reads sensor data, while the Raspberry Pi manages communication and controls.



(a) Structure of the testbed



(b) Circular track

Fig. 5: 4-wheel testbed

The 4-wheel testbed performs lane-keeping on a circular track, shown in Fig. 5b. Its control model, based on the kinematic bicycle framework, mirrors the numerical simulation model with different parameters. In the experiment, the testbed is controlled to run at a constant speed of 0.3 m/s. The target set $\boldsymbol{X_T}$ is defined as within a 0.1-meter range of the center of the track, the safe set $\boldsymbol{X_S}$ is set as within the boundaries of the track, the control inputs $\boldsymbol{u}$ is in $[-1, 1]$, and the noise $\boldsymbol{v}$ for sensors is in $[-0.2, 0.2]$. The control period of this testbed is 100 ms. We inject a bias attack which is -0.8 to the IMU sensor measurement. The recovery starts upon the detector raises an alert. We separately test our recovery-guaranteed and fixed window-based detectors.

4) Metrics: We have two metrics. One is the recoverable time window missing numbers (RWM), which indicates the instances when the system, under attack, cannot maintain a recoverable time window of size K at the time step just before the detector alerts. This means that when the detector triggers an alarm, there is no guarantee that the system can recover safely. The second is the false positive rate (FP), which represents the rate at which the detector issues alarms when the system is not under attack.

5) Baselines: We select two commonly used detectors as baselines for comparison: the *window-based detector* and the *CUSUM detector*, representing short-memory and long-memory detectors, respectively. Neither method can guarantee the system will be able to safely recover when alert. For a fair comparison, the baselines, derived from works [6], [11], [19], have already been well fine-tuned.

6) Detector configurations: In the experiments, the parameter settings for our method are as follows. The desired recoverable time window K for all three simulators and the testbed is set to 3, the window size W of all the detectors is set to 10, and the run interval for the authenticator is set according to the dimensions of the system's state.

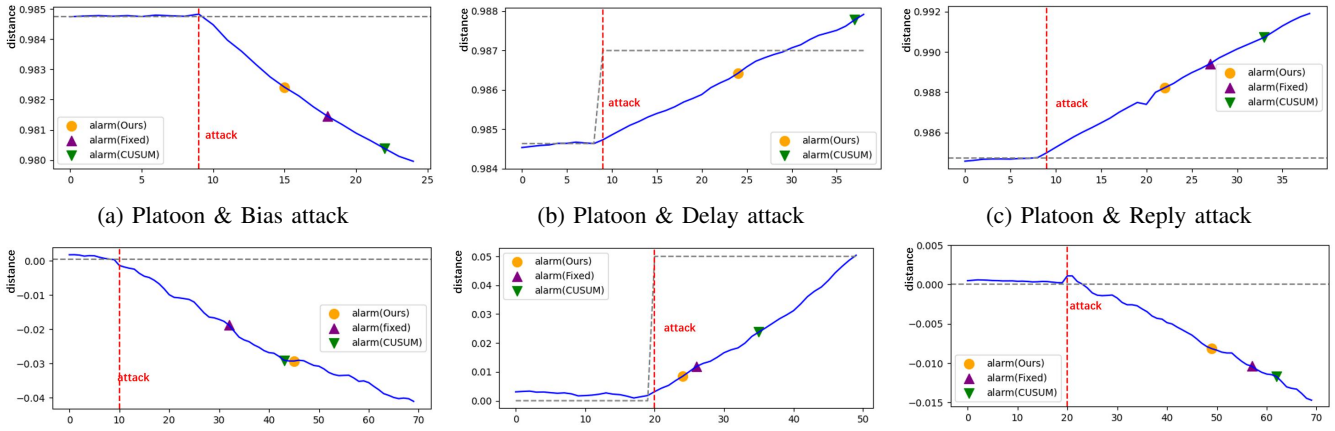
7) Attack scenarios: In our experiments, we consider three types of attack scenarios: bias, delay, and replay attack. A bias attack involves replacing sensor data with arbitrary values. In a delay attack, sensor measurements are delayed before being sent to the controller, extending over a specified time period. A replay attack involves substituting sensor data with previously recorded data. These scenarios help test the robustness of our system under different adversarial conditions.

B. Experimental results

1) Simulation results: Our detector is evaluated under all nine cases, covering every combination of three simulators and three attack scenarios. Fig. 6 shows part of the results using platoon and lane keeping under bias, delay, and reply attacks. For clarity, only the first alarm issued by each detector after an attack is marked in the diagrams. Our recovery-guaranteed detector triggers alarms earlier than others, as it dynamically adjusts the threshold based on the system's recoverable time window. When an attack occurs, the estimated range of the system's real state expands, shifting the state toward the unsafe set. This causes the detector to lower the threshold, prompting an earlier alarm compared to baselines.

Table. III shows the FP and RWM across 100 simulations for each case. Compared to the baseline detectors, our method consistently achieves a zero RWM while maintaining a relatively low FP. For example, in the platoon scenario under bias attack, our detector achieves the lowest FP (1.31) with 0 RWM. This is because the detector dynamically adjusts the threshold: lowering it when the recoverable time window shrinks to maintain a size of K or triggering an alarm if the threshold becomes too low, and increasing it to reduce FP when the window size exceeds K.

2) Testbed results: In the 4-wheel testbed experiment, we select a well-tuned window-based detector as the baseline to



(a) Platoon & Bias attack (b) Platoon & Delay attack (c) Platoon & Reply attack

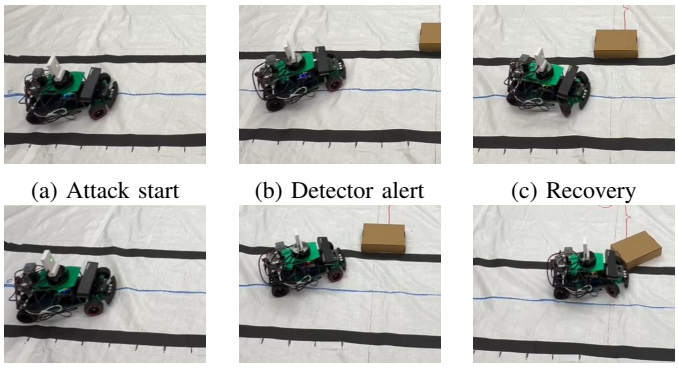
(d) Lane keeping & Bias attack (e) Lane keeping & Delay attack (f) Lane keeping & Reply attack

Fig. 6: Comparison of detector results between recovery-guaranteed detector (Ours), fixed window-based detector (Fixed), and the CUSUM detector (CUSUM). The horizontal axis represents time steps. Blue solid line: system's real state. Grey dashed line: reference value. Red dashed line: attack start time. Orange circle marker: alerts raised by our detector. Purple triangle marker: alerts raised by the window-based detector. Green triangle marker: alerts raised by the CUSUM detector.

Simulator	Attack	Strategy	FP	RWM
Platoon	Bias	Ours	**1.31**	**0**
		Fixed	7.20	59
		CUSUM	3.88	67
	Delay	Ours	**1.20**	**0**
		Fixed	6.37	71
		CUSUM	2.63	77
	Reply	Ours	**1.23**	**0**
		Fixed	4.80	82
		CUSUM	4.42	37
Lane Keeping	Bias	Ours	**7.30**	**0**
		Fixed	14.2	80
		CUSUM	16.1	54
	Delay	Ours	**6.80**	**0**
		Fixed	17.2	75
		CUSUM	11.2	62
	Reply	Ours	**8.30**	**0**
		Fixed	12.6	72
		CUSUM	8.59	63
Aircraft Pitch	Bias	Ours	5.60	**0**
		Fixed	**3.55**	69
		CUSUM	4.10	73
	Delay	Ours	6.13	**0**
		Fixed	5.12	89
		CUSUM	**4.15**	66
	Reply	Ours	**3.28**	**0**
		Fixed	3.95	95
		CUSUM	5.47	71

TABLE III: The experiment results of 3 detectors under 9 scenarios.

compare with our recovery-guaranteed detector. Fig. 7a, 7b, and 7c show the scenes when the testbed using the recovery-guaranteed detector is under attack, when the detector triggers an alarm, and when recovery is successful, respectively. Fig. 7d, 7e, and 7f illustrate the scenes when the testbed using the window-based detector is under attack, when the detector triggers an alarm, and when recovery fails, respectively. In both experiments, the testbed is attacked at the same location, but the testbed using our method raises an alarm earlier and successfully recovers. In contrast, the testbed using the window-based detector raises an alarm later. Although the vehicle remains within the safe set (i.e., within the track) at the time of the alarm, it inevitably enters the unsafe set and collides with an obstacle during the recovery attempt, resulting in a failed recovery. This experiment demonstrates that our method can ensure that the detector raises a timely alarm and guarantees successful recovery on an actual embedded system following a sensor attack.



(a) Attack start (b) Detector alert (c) Recovery

(d) Attack start (e) Detector alert (f) Recovery failed

Fig. 7: The 4-wheel testbed results. Fig. 7a, 7b, and 7c show the testbed using the recovery-guaranteed detector under sensor attack and successfully recovering. Fig. 7d, 7e, and 7f show the testbed using the window-based detector under the same attack but failing to recover.

3) The impact of noise and target set: We use the platoon simulator under bias attacks to demonstrate the impact of noise boundaries and target set sizes. The detector's FP is evaluated across nine scenarios, combining three noise levels and three target set sizes, as shown in Table. IV. For the target set X_T, the sizes of its first three dimensions are adjusted. Results show that as noise increases and the target set size decreases, the FP rate rises. This occurs because higher noise and a smaller target set reduce system recoverability, prompting the detector

	Target set X_T		
Noise v	[0.4, 1.2]	[0.6, 1.2]	[0.8, 1.2]
[-0.001, 0.001]	0.00	0.00	10.6
[-0.002, 0.002]	0.63	6.88	23.1
[-0.004, 0.004]	18.1	35.6	53.1

TABLE IV: The impact of the size of noise and target set on the FP of the recovery-guaranteed detector.

	Platoon	**Lane Keeping**	**Aircraft Pitch**
Authentication	24.4	15.5	11.7
Error estimation	0.11	0.07	0.03
Recoverability Calculation	4.31	1.06	1.01
Threshold Adjustment	0.11	0.22	0.19

TABLE V: The time cost associated with the four computational processes included in our detector. Time unit: ms.

to lower the threshold to maintain the recoverable time window size K, which leads to a higher FP.

4) Overhead analysis: We analyze the time cost of our recovery-guaranteed detector across three simulators, including authentication, error estimation, recoverability calculation, and threshold adjustment times. Table. V displays the time taken by each component, with the times expressed in milliseconds (ms), with a control step size of 40 ms. The table shows that the computations for our recovery-guaranteed detector can be completed within one time step. Note that the authenticator operates in parallel with other components of our detector and does not need to be completed within a single time step.

VIII. Conclusion

We design a recovery-guaranteed sensor attack detection framework for CPS, which gets the conservative range for the system's real state at each time step and calculates the system's recoverable time window using an incremental online recoverability verification method. We develop an efficient algorithm to maintain the recoverable window size by adjusting the threshold of the detector. Implemented on three simulators and a testbed, our framework reduces the false positive rate while ensuring successful recovery.

Acknowledgment

This work was supported in part by NSF CNS-2333980 and the Air Force under PIA FA8750-19-3-1000. The U.S. Government is authorized to reproduce and distribute copies for Governmental purposes notwithstanding any copyright or other restrictive legends. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of National Science Foundation (NSF) or the Air Force.

References

[1] C. L. DeMarco, J. Sariashkar, and F. Alvarado, "The potential for malicious control in a competitive power systems environment," in *Proceeding of the 1996 IEEE International Conference on Control Applications IEEE International Conference on Control Applications held together with IEEE International Symposium on Intelligent Contro*. IEEE, 1996, pp. 462–467.

[2] G. Dán and H. Sandberg, "Stealth attacks and protection schemes for state estimators in power systems," in *2010 first IEEE international conference on smart grid communications*. IEEE, 2010, pp. 214–219.

[3] F. Pasqualetti, F. Dörfler, and F. Bullo, "Cyber-physical attacks in power networks: Models, fundamental limitations and monitor design," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE, 2011, pp. 2195–2201.

[4] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 881–896.

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

[6] 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.

[7] R. Tunga, C. Murguia, and J. Ruths, "Tuning windowed chi-squared detectors for sensor attacks," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 1752–1757.

[8] 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)*. IEEE, 2020, pp. 1–6.

[9] 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.

[10] L. Zhang, X. Chen, F. Kong, and A. A. Cardenas, "Real-time attack-recovery for cyber-physical systems using linear approximations," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 205–217.

[11] 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 Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[12] 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.

[13] L. Zhang, K. Sridhar, M. Liu, P. Lu, X. Chen, F. Kong, O. Sokolsky, and I. Lee, "Real-time data-predictive attack-recovery for complex cyber-physical systems," in *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2023, pp. 209–222.

[14] M. Liu, L. Zhang, V. V. Phoha, and F. Kong, "Learn-to-respond: Sequence-predictive recovery from sensor attacks in cyber-physical systems," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 78–91.

[15] P. Lu, L. Zhang, M. Liu, K. Sridhar, O. Sokolsky, F. Kong, and I. Lee, "Recovery from adversarial attacks in cyber-physical systems: Shallow, deep, and exploratory works," *ACM Computing Surveys*, vol. 56, no. 8, pp. 1–31, 2024.

[16] X. Guo, S. Han, X. S. Hu, X. Jiao, Y. Jin, F. Kong, and M. Lemmon, "Towards scalable, secure, and smart mission-critical iot systems: review and vision," in *Proceedings of the 2021 International Conference on Embedded Software*, 2021, pp. 1–10.

[17] 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.

[18] F. Akowuah, K. Fletcher, and F. Kong, "Variable window and deadline-aware sensor attack detector for automotive cps," in *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2023, pp. 54–63.

[19] F. Akowuah and F. Kong, "Real-time adaptive sensor attack detection in autonomous cyber-physical systems," in *2021 IEEE 27th real-time and embedded technology and applications symposium (RTAS)*. IEEE, 2021, pp. 237–250.

[20] Z. Wang, L. Zhang, Q. Qiu, and F. Kong, "Catch you if pay attention: Temporal sensor attack diagnosis using attention mechanisms for cyber-physical systems," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 64–77.

[21] M. A. Arroyo, M. T. I. Ziad, H. Kobayashi, J. Yang, and S. Sethumadhavan, "Yolo: frequently resetting cyber-physical systems for security,"

in *Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2019*, vol. 11009. SPIE, 2019, pp. 166–183.

[22] L. Niu, D. Sahabandu, A. Clark, and R. Poovendran, "Verifying safety for resilient cyber-physical systems via reactive software restart," in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2022, pp. 104–115.

[23] 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.

[24] H. Choi, W.-C. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Deng, "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.

[25] E. Schulte, "Symmetry of polytopes and polyhe dra," in *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017, pp. 477–503.

[26] A. Girard, "Reachability of uncertain linear systems using zonotopes," in *Proceedings of the 8th International Workshop on Hybrid Systems: Computation and Control (HSCC'05)*, ser. LNCS, vol. 3414. Springer, 2005, pp. 291–305.

[27] X. Cai, K. Han, Y. Li, X. Li, J. Zhang, and Y. Zhang, "A sensor attack detection method based on fusion interval and historical measurement in cps," in *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2020, pp. 1–4.

[28] Y. Mo, R. Chabukswar, and B. Sinopoli, "Detecting integrity attacks on scada systems," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 4, pp. 1396–1407, 2013.

[29] D. Ye, T.-Y. Zhang, and G. Guo, "Stochastic coding detection scheme in cyber-physical systems against replay attack," *Information Sciences*, vol. 481, pp. 432–444, 2019.

[30] Y. Mo and B. Sinopoli, "Secure control against replay attacks," in *2009 47th annual Allerton conference on communication, control, and computing (Allerton)*. IEEE, 2009, pp. 911–918.

[31] H. Jeon and Y. Eun, "A stealthy sensor attack for uncertain cyber-physical systems," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6345–6352, 2019.

[32] S. Kim, Y. Eun, and K.-J. Park, "Stealthy sensor attack detection and real-time performance recovery for resilient cps," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7412–7422, 2021.

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

[34] H. Fawzi, P. Tabuada, and S. Diggavi, "Secure estimation and control for cyber-physical systems under adversarial attacks," *IEEE Transactions on Automatic control*, vol. 59, no. 6, pp. 1454–1467, 2014.

[35] M. Pajic, J. Weimer, N. Bezzo, P. Tabuada, O. Sokolsky, I. Lee, and G. J. Pappas, "Robustness of attack-resilient state estimators," in *2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2014, pp. 163–174.

[36] S. Dadras, R. M. Gerdes, and R. Sharma, "Vehicular platooning in an adversarial environment," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 167–178. [Online]. Available: https://doi.org/10.1145/2714576.2714619

[37] 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)*. USENIX Association, Aug. 2020, pp. 895–912. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/quinonez