2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS) | 979-8-3503-2176-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/RTAS58335.2023.00024

Real-Time Data-Predictive Attack-Recovery for Complex Cyber-Physical Systems

Lin Zhang*, Kaustubh Sridhar†, Mengyu Liu*, Pengyuan Lu†,
Xin Chen‡, Fanxin Kong*, Oleg Sokolsky†, Insup Lee†
*Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse NY
†Department of Computer and Information Science, University of Pennsylvania, Philadelphia PA
‡Department of Computer Science, University of Dayton, Dayton OH
lzhan120@syr.edu, ksridhar@seas.upenn.edu, mliu71@syr.edu, pelu@seas.upenn.edu,
xchen4@udayton.edu, fkong03@syr.edu, sokolsky@cis.upenn.edu, lee@cis.upenn.edu

***Abstract*—Cyber-physical systems (CPSs) leverage computations to operate physical objects in real-world environments, and increasingly more CPS-based applications have been designed for life-critical applications. Therefore, any vulnerability in such a system can lead to severe consequences if exploited by adversaries. In this paper, we present a data predictive recovery system to safeguard the CPS from sensor attacks, assuming that we can identify compromised sensors from data. Our recovery system guarantees that the CPS will never encounter unsafe states and will smoothly recover to a target set within a conservative deadline. It also guarantees that the CPS will remain within the target set for a specified period. Major highlights of our paper include (i) the recovery procedure works on nonlinear systems, (ii) the method leverages uncorrupted sensors to relieve uncertainty accumulation, and (iii) an extensive set of experiments on various nonlinear benchmarks that demonstrate our framework's performance and efficiency.**

***Index Terms*—cyber-physical systems, security, real-time recovery, nonlinear systems**

I. Introduction

Cyber-Physical Systems (CPSs) tightly integrate computational components and physical processes, which interact in a feedback loop with the help of sensors, actuators, and networking. The integration empowers critical applications and services in various domains, such as transportation, health care, the power grid, and industrial control [1]–[3]. Meanwhile, new vulnerabilities in CPSs are emerging due to the transition from isolated control architectures to open and autonomous systems [4]–[9]. Different malicious attacks targeted at these vulnerabilities are springing up, causing severe personal casualties, social harm, and economic losses [10]–[12].

Sensor attacks, which corrupt sensor measurements, 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 [13]. 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 [14]. 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 [15]. 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 [16]. 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.

The urgent need to combat sensor attacks motivates many attack recovery approaches. These recovery approaches mitigate the negative impacts of sensor attacks by correcting deviations of physical states. For example, Fig. 1 illustrates a recovery process for an autonomous vehicle. The vehicle deviates from the center of lanes under a GPS spoofing attack, and even goes towards the oncoming lane. A recovery approach can drive the vehicle back to its own lane, and the green line shows the recovery trajectory. Recovery approaches can be divided into two threads. The first thread is the virtual-sensor-based attack recovery [12], [17], which replaces the corrupted sensor data with the virtual sensor data predicted by the system model. On the basis of such predicted data, the system relies on the original controller to steer the physical states back to the desired states. The other thread is safety-controller-based attack recovery [18], [19], which creates a new dedicated controller to generate a recovery control sequence under attacks. The controller can be formulated as an optimization problem with time and safety constraints.

However, these two existing recovery threads face several challenges when applied to real systems. First, the virtual-

979-8-3503-2176-0/23/$31.00 ©2023 IEEE
DOI 10.1109/RTAS58335.2023.00024

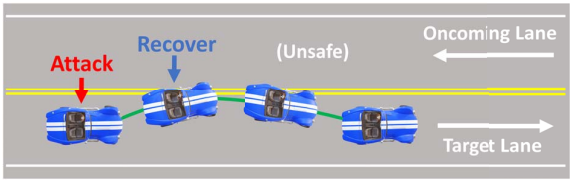


Fig. 1: Attack Recovery Demonstration on a Vehicle

sensor-based attack recovery relies on the original nominal controller to recover the system, but nominal controllers are not designed to handle attacks, lacking timing or safety guarantees. Second, real CPSs are complex and usually nonlinear, so the recovery computational overhead on such systems is extremely large. These methods reduce the complexity by working on a linear model, which can be obtained by the linearization of nonlinear systems around an equilibrium point, or by system identification. Nevertheless, the linear model only works well around the equilibrium point or in a small range. Out of this range, there is a large modeling error, making these methods fail to find a recovery solution. Third, uncertainties, including noise and external disturbances, accumulate over time. This happens because, after detecting the attack, the existing methods assume that all sensors are attacked, and, therefore, stop receiving feedback from physical sensors. Without feedback, uncertainties cannot be rejected anymore and accumulate, affecting the effectiveness of recovery.

To overcome these challenges, this paper proposes a new real-time recovery method against senor attacks for nonlinear CPSs. Once the detector identifies a sensor attack, the proposed method takes over the nominal controller, efficiently removing the negative impact caused by the attack and steering the system to the desired states in real time.

Our main contributions are summarized as follows.

• For nonlinear CPSs, we propose an attack recovery system with four components. The state predictor performs nonlinear reachability analyses, making it possible to reconstruct the initial state set of recovery. Based on that, the time oracle computes a safety deadline online, after which the system states may become unsafe and cause severe consequences under current control inputs. Throughout the recovery, the model adaptor keeps approximating the nonlinear system to linear discrete-time models, enabling the generator to efficiently obtain the recovery control sequence that can drive the CPS to a target state before the safety deadline.

• The proposed method computes timing requirements, online, in terms of recovery time. Sensor attacks deviate the system's physical states away from the desired states, and it is impossible to anticipate this impact in advance. If the actual physical state after an attack is closer to the unsafe set, a shorter deadline should be chosen, and vice versa [20]. Thus, the proposed method computes the safety deadline online using the time oracle to better respond to different attacks.

• The proposed method makes full use of the uncompromised sensor data, if any, to alleviate the accumulation of uncertainty. The recovery control sequence generator uses good sensor measurements as feedback at each activation, which prevents the uncertainty from exploding in the uncorrupted dimensions. Moreover, it leverages good sensor measurements during state reconstruction also, which further relieves the impact caused by uncertainties.

• The proposed method has low computational overhead. First, the state predictor adopts a tool, Flow*, to perform fast nonlinear reachability analysis. Second, the model adaptor approximates the nonlinear and even nonconvex dynamics into linear discrete-time models. Thus, the recovery controller can solve optimization problems for linear systems instead of nonlinear ones, reducing computational overhead. Also, the linear approximation is kept updated with the current state estimate and control input during recovery, and the approximation is quite accurate within a small range. Thus, accuracy is only minimally affected by the approximation.

• 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.

• The paper evaluates the proposed method with several numerical and high-fidelity simulators. The results show the effectiveness of the proposed method, whereas the baseline methods may fail to recover the physical states.

The rest of this paper is organized as follows. Section II provides background and discussion of related work. Section III gives preliminaries of our system's recovery algorithm. Section IV details the design of system components. Section V validates our recovery system. Section VI concludes the paper.

II. Background and Related Work

Current Research Focus. With the rise of CPS autonomy, sensors are increasingly deployed to measure system states and to perceive surroundings. However, new security vulnerabilities in sensors come along, and sensor attacks spring up, such as transduction attacks. Moreover, the launch of such attacks becomes much easier, as the attack requires lower and lower costs [21]–[23]. The urgent defense needs motivated many attack detection works. The detectors leverage the system model [24]–[29] or sensor correlations [11], [30]–[33] to find anomalies compared to expected data. Note that, there usually is a detection delay before the attacks are detected, and the physical states have already considerably deviated from the system's desired states, even will reach the unsafe states in the future. Thus, it is important to extend the benefits of attack detection and remove the negative impact caused by CPS attacks in time. Researchers place great expectations on attack recovery techniques to achieve this goal.

Attack Recovery Works. CPS recovery is a sprouting field that specifically researches how to correct a CPS's behaviors upon adversarial attacks. We divide these recovery techniques into two main categories: shallow and deep recovery, with

the latter providing higher safety guarantees but having few publications in the current state of the art.

We use the term shallow recovery for methods that aim to fix affected CPS behaviors under attacks but have little to no requirement on system states. One basic solution is to simply restart the corrupted component while switching to a substitute component to maintain system stability [34], [35]. Another example of shallow recovery is to leverage redundancy, using multiple components with the same functionality and taking actions collectively, such as fusion on redundant sensors. Then, upon a detected attack on some of these components, the CPS isolates them from decision making [36]. Moreover, specifically upon sensor attacks, the current state measurement is corrupted but some system designs allow state estimation via built-in software. For example, the system can use a checkpointer to memorize the last trustworthy state and roll-forward on its model to predict what the current actual state should be [12], [17], [37]–[40].

On the other hand, deep recovery leverages prerequisite knowledge of system states and aims at guiding the attacked system back to pre-defined target states, which satisfy desirable properties, such as safety, and maintain the system there. For example, [18] designs a linear programming recovery controller to accomplish this goal, while a succeeding work [19] replaces the control method with linear quadratic regulator and achieves smoother trajectories on multiple linear system benchmarks. This approach has been applied to real-world scenarios such as power grids, where safe and unsafe are defined by different levels of thermal loading [41], [42]. More than traditional controllers, machine learning researchers have proposed learned control policies for deep recovery, such as rescuing an attacked drone by control policies trained via reinforcement learning [43]. The paper proposes a deep recovery approach for non-linear systems.

Closest Related Work. The closest related work to this paper is [18] and [19], identified as deep recovery. The recovery system proposed in [18] provides an initial solution to real-time recovery for CPS under sensor attacks. It builds a linear-programming-based recovery controller that generates a recovery control sequence bringing a physical system under sensor attacks back to a target set in real-time. Then, the LQR-based recovery system in [19] improves it by smoothing the recovery state trajectory, maintaining physical states after recovery, and considering a larger computational overhead.

However, several key aspects are overlooked in both works. First, they assume that they recover the physical state of linear time-invariant systems. However, most real-world systems are nonlinear, and it is hard to find a linear approximation during the entire recovery process. Second, they abandon sensor measurements of all sensors once a sensor attack is detected, even when only part of the sensors is corrupted. However, the use of these uncompromised sensors could improve the possibility of successful recovery and recovery performance. This paper addresses these key aspects.

Scope of this Paper. To address the challenges, we pursue real-time attack recovery, which steers a nonlinear physical system back to a target state set before the recovery deadline and maintains the state in the set for a while. Recovery provides a time cushion for compromised parts to be reset, and then the normal controller can retake control of the system.

Note that the recovery addressed in this work is different from the recovery concept in computer systems. The latter is to recover computing tasks, e.g., variables' values, and thus is limited to the cyber part [44]–[49]. In contrast, this paper focuses on recovering the state of the physical system or the physical state, e.g., the speed of a vehicle.

TABLE I: Nomenclature

Symbols	Description
n	dimension of state space
m	dimension of control space
$\boldsymbol{x}_t$	system state estimate at the time t
$\boldsymbol{u}_t$	control input at the time t
X_t	overapproximation of $\boldsymbol{x}_t$
$\mathcal{T}$	target state set $\subset \mathbb{R}^n$
$\mathcal{F}$	unsafe state set $\subset \mathbb{R}^n$, $\mathcal{F} \cap \mathcal{T} = \emptyset$
t_w	time when the last trustworthy state is cached
t_a	unknown time when the attack begins
t_f	time when attack is detected and recovery begins
t_r	time when the first recovery control is implemented
t_d	deadline by which the system is in target set
t_m	time such that the system is maintained in target set in $[t_d, t_m]$
δ	granularity of time; length of one control step
T	number of control steps within which one optimization computation is guaranteed to finish
J	quadratic cost function of each optimization problem

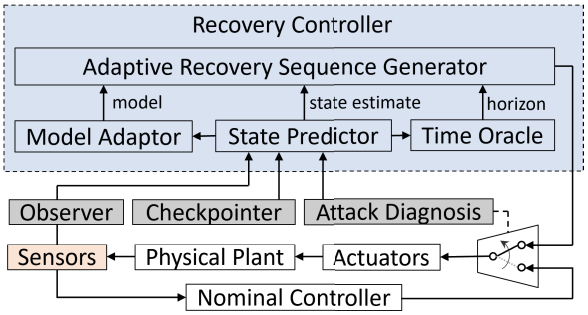


Fig. 2: Real-time Data Predictive Recovery Overview

III. Preliminaries and System Overview

This section explains the preliminaries of the real-time data predictive attack-recovery system, including the system model and the threat model. Then we summarize the overview of system design, which will be discussed further in Section IV, and give the problem statement and assumptions.

A. System Model

We aim to recover cyber-physical systems shown at the bottom of Fig. 2, including a physical plant, a nominal controller, sensors, and actuators. The system works at every control step δ in a periodic manner. First, the sensors measure the physical plant states. The controller then calculates the state estimate based on the sensor measurements. Next,

the controller generates the control inputs $\boldsymbol{u}$ according to a control algorithm and sends them to the actuators. Finally, the actuators implement the control inputs $\boldsymbol{u}$ and drive the system towards the desired or reference states. The state estimate $\boldsymbol{x}$ is a vector of size n determined by the number of physical system state variables (such as velocity, pressure, voltage, etc.); the control input $\boldsymbol{u}$ is a vector of size m determined by the number of control signals (such as throttle, steering angle, etc.).

The system plant follows the physical laws that can be modeled as a continuous nonlinear system using an ordinary differential equation in the form of Equation (1).

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{d} \tag{1}$$

where $\boldsymbol{d}$ is the disturbance term due to uncertainties. 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. We discuss how to extend the system with an observer in Section IV-C1.

B. Threat Model

The CPS suffers from sensor attacks, where sensor attacks aim at compromising the integrity or availability of sensor data. The controller then generates inappropriate control inputs based on malicious sensor data. We list some possible sensor attack scenarios, but are not limited to these. For compromising integrity, (i) bias attacks change the value of sensor data by adding a bias to them; (ii) replay attacks replace the current sensor data with historical state ones. For compromising availability, (iii) delay attacks defer the update of sensor data, leading to stale state estimates. There is no assumption on the number of compromised sensors. Our method, similar to [18], [19], can operate when all sensors are compromised but is the only framework that can leverage measurements from uncompromised sensors. That is, measurements from good sensors, if any, can only help to improve recovery performance. We demonstrate the same in our experimental results in Section V.

C. Problem Statement

We consider a nonlinear CPS described in Section III-A under the sensor attacks shown in Section III-B. 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.

D. Recovery Controller Overview

Our real-time data predictive recovery system is shown in Fig. 2. 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.

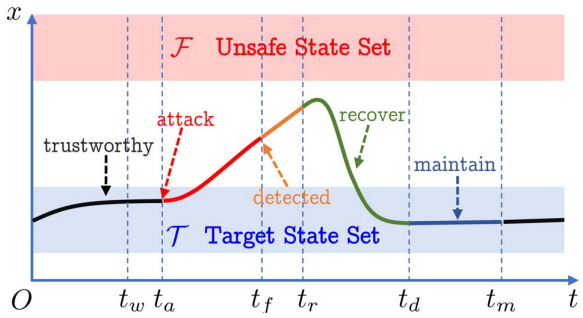


Fig. 3: Illustration of Recovery Timeline

When does the recovery controller take effect? The system runs in two possible modes - normal mode and recovery mode. Fig. 3 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 IV-C) 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.

E. 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 Sec-

tion III-A. Also, sensor attacks make the nominal controller generate inappropriate control inputs that lead to the deviation from desired physical states, detailed in Section III-B.

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 [12], [18], [19], [37]. Under the assumption of fully observable states at the end of Section III-A, 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 [23], [27]. 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 [50]. We assume that the recovery control sequence can be implemented to actuators, or we need a redundant actuator in the systems.

IV. Model Predictive Recovery Controller

In this section, we provide an in-depth explanation of our recovery controller, including the adaptive recovery sequence generator and the supportive components: model adaptor, state predictor, and time oracle.

A. Data Predictive Recovery Algorithm

The blue shaded box in Fig. 2 illustrates the data predictive recovery controller. The controller consists of four components that cooperate with each other. The adaptive recovery sequence generator (Section IV-B), 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 IV-C1) and the time oracle (Section IV-C3), respectively. In addition, the model adapter (Section IV-C2) updates the linear approximation of the nonlinear dynamics before formulating each optimization problem.

Algorithm 1 shows the real-time attack-recovery procedure, and the critical time is reflected in the timeline (Fig. 3):
(i) Lines 3-4: 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 5-11: 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 IV-C1) 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 IV-C3). Then, the model adaptor (Section IV-C2) computes an updated linear approximation around current states and control input from nonlinear dynamics. Based on them, the adaptive recovery sequence generator (Section IV-B) 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 12-13: During period (t_f, t_r), the system runs with the prepared control sequence, meanwhile solving opt_0.
(iv) Lines 14-21: 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.

Algorithm 1 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 $0 \leq t \leq t_m$

1: **while** $t \geq 0$ **do**
2: **switch** t **do**
3: **case** $t < t_f$
4: Run with normal control
5: **case** $t = t_f$
6: $t_d, t_m \leftarrow$ time oracle ▷ deadline computing
7: $X_{t_r} \leftarrow$ state predictor ▷ state reconstruction
8: $sys_t \leftarrow$ model adaptor ▷ model adaptation
9: $opt_0(sys_t, X_{t_r}, t_m - t_r)$ ▷ first opt.
10: $\boldsymbol{u}_t, \ldots, \boldsymbol{u}_{t_r-\delta} \leftarrow \boldsymbol{u}_t$ ▷ from nominal control
11: Run with $\boldsymbol{u}_t$ ▷ implement control input
12: **case** $t_f < t < t_r$
13: Run with $\boldsymbol{u}_{t_f+\delta}, \ldots, \boldsymbol{u}_{t_r-\delta}$ ▷ cached control
14: **case** $t_r \leq t \leq t_m$
15: **if** $t = t_r + KT\delta$ with integer $K \geq 0$ **then**
16: $\boldsymbol{u}_t, \ldots, \boldsymbol{u}_{t+(T-1)\delta} \leftarrow opt_K$ ▷ opt. result
17: $X_{t+T\delta} \leftarrow$ state predictor ▷ reachability
18: $sys_t \leftarrow$ model adaptor ▷ model adapt.
19: $opt_{K+1}(sys_t, X_{t+T\delta}, t_m - (t + T\delta))$
20: **end if**
21: Run with $\boldsymbol{u}_t$ ▷ implement control input
22: **end while**

B. 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 (1). To efficiently compute recovery control inputs, we perform local linearization and discretization online using the model adaptor (Section IV-C2), 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.

*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 steer towards reference state fast and smoothly:

$$\begin{aligned} J = &(\boldsymbol{x}_N - \boldsymbol{x}^*)^T \mathbf{Q}_N (\boldsymbol{x}_N - \boldsymbol{x}^*) \\ &+ \sum_{k=0}^{N-1} (\boldsymbol{x}_k - \boldsymbol{x}^*)^T \mathbf{Q} (\boldsymbol{x}_k - \boldsymbol{x}^*) + \boldsymbol{u}_k^T \mathbf{R} \boldsymbol{u}_k \end{aligned} \tag{2}$$

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.

$$\boldsymbol{x}_{i+1} = \mathbf{A}\boldsymbol{x}_i + \mathbf{B}\boldsymbol{u}_i + \boldsymbol{c} \qquad \forall i \tag{3a}$$

$$\boldsymbol{u}_i \in \mathcal{U} \qquad \forall i \tag{3b}$$

$$X_i \cap \mathcal{F} = \emptyset \qquad \forall i \in [0, M] \tag{3c}$$

$$\boldsymbol{x}_i \in \mathcal{T} \qquad \forall i \in [D, M] \tag{3d}$$

Notice that Eq. (3a) is the discrete linearized dynamic constraint provided by the model adapter as in Equation (7), with parameters $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{c}$. It is used to predict the plant's future evolution; Eq. (3b) limits our control inputs according to the actuator's capacity; Eq. (3c) ensures that all recovery states are safe; Eq. (3d) 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.

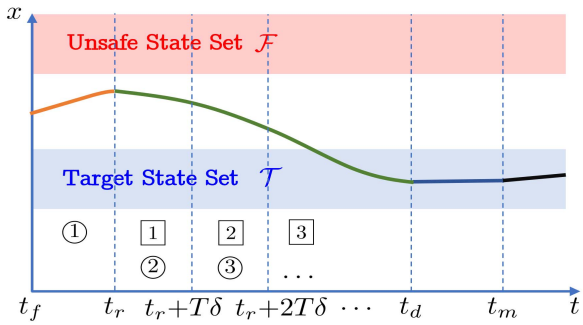


Fig. 4: Illustration of Extended Model Predictive Recovery. ① denotes solving the i^{th} optimization problem, and $\boxed{i}$ denotes implementing the recovery control inputs computed from i^{th} optimization problem.

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 did not consider the uncertainty in the constraints in Eq. (3), such as linearization error and estimation noises due to overapproximated reachable sets. Thus, some constraints are not guaranteed to be met in real applications.

*2) **Extended Data Predictive Recovery Formulation**:* To overcome the limitations above, we extend the basic formulation by considering computational overhead and uncertainties, shown in Fig. 4.

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 IV-C. The constraints of the optimization problem are reformulated as follows.

$$\boldsymbol{x}_0 \in X_0 = \boldsymbol{x}_0 \oplus I_0 \qquad \text{from state predictor} \qquad (4a)$$
$$\boldsymbol{x}_{i+1} = \mathbf{A}\boldsymbol{x}_i + \mathbf{B}\boldsymbol{u}_i + \boldsymbol{c} \qquad \forall i \qquad (4b)$$
$$\boldsymbol{u}_i \in \mathcal{U} \qquad \forall i \qquad (4c)$$
$$\boldsymbol{x}_i \cap (\mathcal{F} \oplus \mathbf{A}^i I_0) = \emptyset \qquad \forall i \in [0, M] \qquad (4d)$$
$$\boldsymbol{x}_i \in \mathcal{T} \ominus \mathbf{A}^i I_0 \qquad \forall i \in [D, M] \qquad (4e)$$

where at Eq. (4a), x_0 is the initial state of recovery with uncertainty interval I_0 that can be obtained from the state predictor; Eq. (4d) and (4e) 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.

C. 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.

*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. Fig. 5 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.

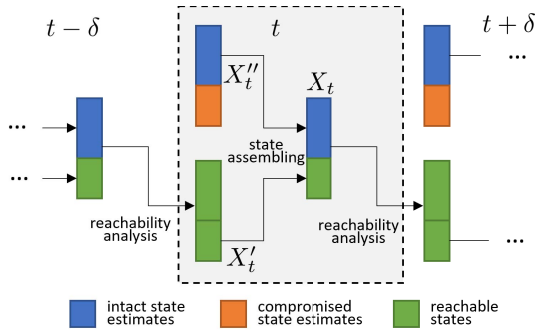


Fig. 5: A fragment of successive calculation of reachable states using the state predictor

Also, current state 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 (4a). 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 [51], and are later used in verified integration of nonlinear ODEs [52] and to compute reachable set overapproximations for hybrid systems [53]–[55]. To do so, we may directly use the state-of-the-art tool, Flow* [56]. Since we only use the tool to compute an interval 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 (1), 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. Fig. 6 illustrates the use of Flow*.

Considering Observer. As stated at the end of Section III-A, 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 Fig. 3, 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

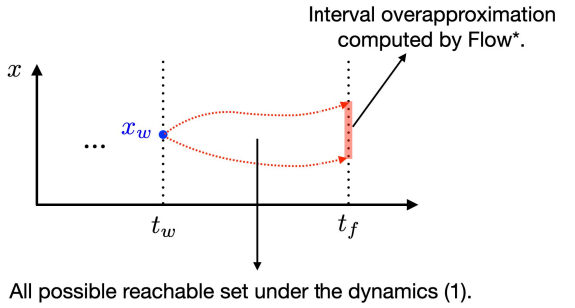


Fig. 6: Illustration of the use of Flow*.

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 Fig. 3), 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 1)

(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 1)

(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.

*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. (4b). Given a continuous nonlinear system described as Eq. (1) 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 $\bar{\boldsymbol{u}}_t$:

$$\begin{aligned}\dot{\boldsymbol{x}}_t &= \varphi^c(\boldsymbol{x}_t, \boldsymbol{u}_t) \\ &\approx \varphi^c(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t) + \frac{\partial \varphi^c}{\partial \boldsymbol{x}_t}\Big|_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t} \cdot (\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t) + \frac{\partial \varphi^c}{\partial \boldsymbol{u}_t}\Big|_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t} \cdot (\boldsymbol{u}_t - \bar{\boldsymbol{u}}_t) \\ &= \mathbf{A}'\boldsymbol{x}_t + \mathbf{B}'\boldsymbol{u}_t + \boldsymbol{c}'\end{aligned} \tag{5}$$

where $\mathbf{A}' = \frac{\partial \varphi^c}{\partial \boldsymbol{x}_t}\big|_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t}$, $\mathbf{B}' = \frac{\partial \varphi^c}{\partial \boldsymbol{u}_t}\big|_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t}$, and $\boldsymbol{c}' = \varphi^c(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t) - \frac{\partial \varphi^c}{\partial \boldsymbol{x}_t}\big|_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t} \cdot \bar{\boldsymbol{x}}_t - \frac{\partial \varphi^c}{\partial \boldsymbol{u}_t}\big|_{\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{u}}_t} \cdot \bar{\boldsymbol{u}}_t$. 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) and is given as follows.

$$\begin{aligned}\boldsymbol{x}_{t+\delta} &= e^{\delta \mathbf{A}'}\boldsymbol{x}_t + (\mathbf{A}')^{-1}(e^{\delta \mathbf{A}'} - \mathbf{I})\mathbf{B}'\boldsymbol{u}_t + \delta \boldsymbol{c}' \\ &\approx (\mathbf{I} + \delta \mathbf{A}')\boldsymbol{x}_t + \delta \mathbf{B}'\boldsymbol{u}_t + \delta \boldsymbol{c}'\end{aligned} \tag{6}$$

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 (7), where $\mathbf{A} = \mathbf{I} + \delta \mathbf{A}'$, $\mathbf{B} = \delta \mathbf{B}'$, and $\boldsymbol{c} = \delta \boldsymbol{c}'$.

$$\boldsymbol{x}_{t+\delta} = \mathbf{A}\boldsymbol{x}_t + \mathbf{B}\boldsymbol{u}_t + \boldsymbol{c} \tag{7}$$

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).

$$\begin{aligned}\dot{\boldsymbol{x}}_t &= \boldsymbol{x}_t^2 + \boldsymbol{u}_t \approx \bar{\boldsymbol{x}}_t + \bar{\boldsymbol{u}}_t + 2\bar{\boldsymbol{x}}_t \cdot (\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t) + 1 \cdot (\boldsymbol{u}_t - \bar{\boldsymbol{u}}_t) \\ &= 2\bar{\boldsymbol{x}}_t\boldsymbol{x}_t + \boldsymbol{u}_t - 2\bar{\boldsymbol{x}}_t^2 + \bar{\boldsymbol{x}}_t\end{aligned} \tag{8}$$

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. (6) with some time step such as $\delta = 0.01$.

*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 [19] 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 $t_m = t_d + M$.

V. Evaluation

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

A. Experiment Environment

We implement a nonlinear system simulation tool using Python. 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.

B. 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 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 [58] 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,

$$\begin{aligned} V\dot{C_A} &= q\left(C_{af} - C_A\right) - k_0 \exp\left(\frac{-E}{RT}\right) VC_A \\ \rho C_p V\dot{T} &= \rho C_p q\left(T_f - T\right) + \Delta H k_0 \exp\left(\frac{-E}{RT}\right) VC_A \\ &\quad + UA\left(T_C - T\right) \end{aligned} \tag{9}$$

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 [59], and the bias parameter is shown in TABLE II.

Quadrotor: The quadrotor dynamics [23], [60]–[62] describe the evolution of its attitude 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,

$$\begin{aligned} \dot{\phi} &= w_\phi, & \dot{w}_\phi &= \tfrac{U_\phi}{I_x} + \dot{\theta}\dot{\psi}\left(\tfrac{I_y - I_z}{I_x}\right) \\ \dot{\theta} &= w_\theta, & \dot{w}_\theta &= \tfrac{U_\theta}{I_y} + \dot{\phi}\dot{\psi}\left(\tfrac{I_z - I_x}{I_y}\right) \\ \dot{\psi} &= w_\psi, & \dot{w}_\psi &= \tfrac{U_\psi}{I_z} + \dot{\phi}\dot{\theta}\left(\tfrac{I_x - I_y}{I_z}\right) \\ \dot{x} &= v_x, & \dot{v}_x &= \tfrac{U_t}{m}(\cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi)) \\ \dot{y} &= v_y, & \dot{v}_y &= \tfrac{U_t}{m}(\cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi)) \\ \dot{z} &= v_z, & \dot{v}_z &= \tfrac{U_t}{m}\cos(\phi)\cos(\theta) - g \end{aligned} \tag{10}$$

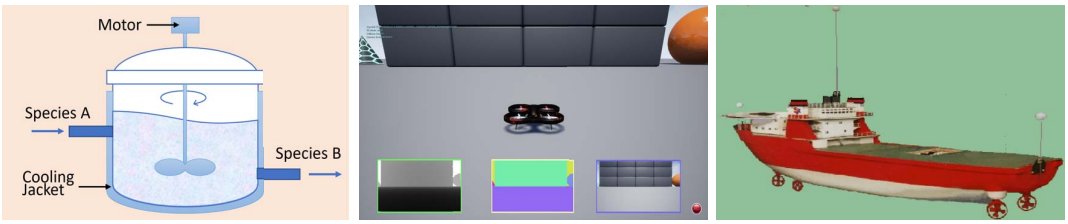
The state-space model is abstracted from a high-fidelity simulator AirSim that Microsoft developed using the Unreal engine as shown in Fig. 7b. A PID controller is used to maintain the quadrotor at a certain height. Here, we consider the sensor bias attack scenario in which the attacker compromises height sensors resulting in incorrect feedback (see [18], [19]).

Vessel: The vessel dynamics describe the generic ship with 3 DoF [57], [63]. 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 Fig. 7c. 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.

C. Experiment Settings

All experiment settings are given in Table II. We chose our CSTR settings, including reference, frequencies (dt, MPC frequency), delay, safe/target sets based on previous work in [58]. We tuned a PID controller to stabilize the model within control limits determined by the physical properties of the CSTR system [58]. For the quadrotor, we build on the settings for the linearized model [19], 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 II's second column). Finally, for the Naval Vessel, we adapt reference, frequency, safe/target set settings from [57], and tune two PID controllers to stabilize the reference speed. Our control limits are again chosen based on the actuator properties of the vessel [57]. In all three benchmarks, we chose asymmetric positive noise to clearly observe the performance of the recovery algorithms. Symmetric noises, on the other



(a) CSTR (b) Drone from AirSim (c) Vessel model 1:70 scale [57]

Fig. 7: Numerical and High-Fidelity CPS Simulators

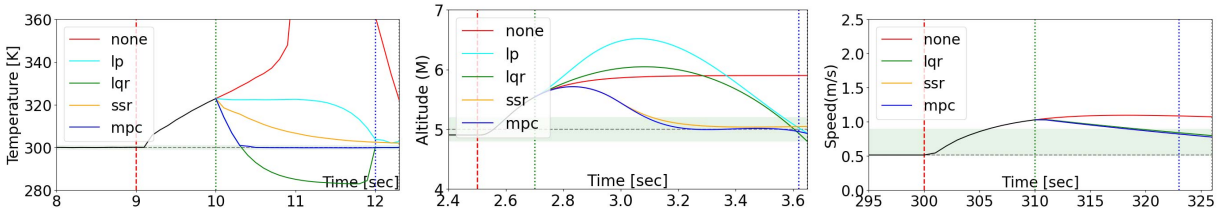


Fig. 8: Performance of our method (MPC recovery control) compared to the baselines (no recovery, LP recovery control, LQR recovery control, and Software Sensor Recovery or SSR) on three benchmarks: continuous stirred tank reactor (CSTR) control **[left]**, quadrotor altitude control **[middle]**, naval vessel control **[right]**.

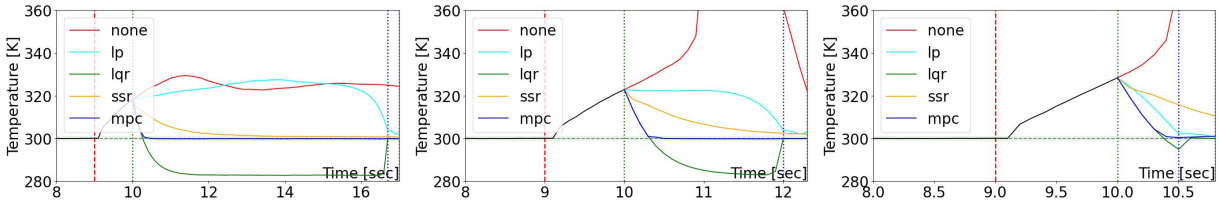


Fig. 9: Sensitivity analysis to bias values of $\{-25, -30, -35\}$ on the CSTR benchmark.

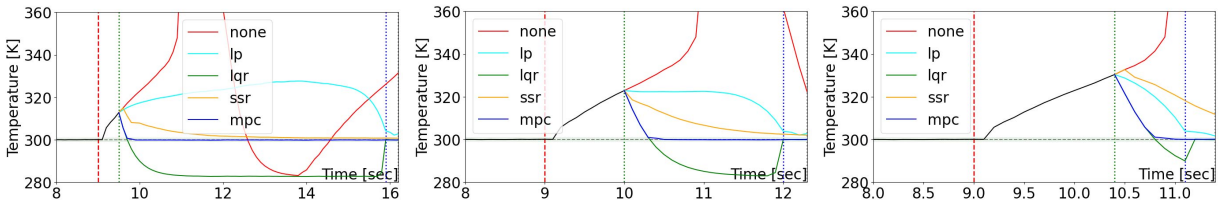


Fig. 10: Sensitivity analysis to detection delay values of $\{0.5, 1.0, 1.4\}$ on the CSTR benchmark.

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.

D. 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)** [12], [17]. 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.

	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 II: Settings used in each benchmark.

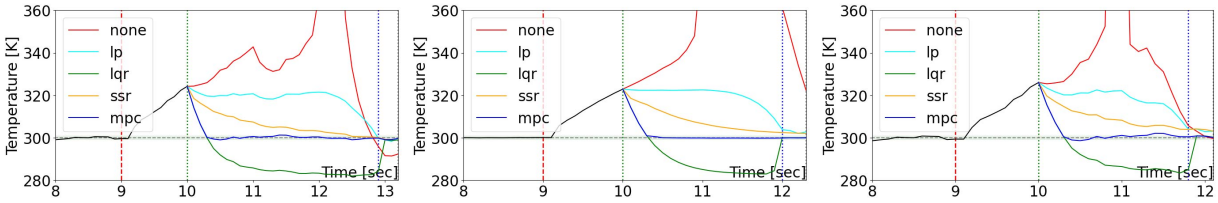


Fig. 11: Sensitivity analysis to noise with upper bounds of $\{0.4, 0.1, 0.6\}$ on the CSTR benchmark.

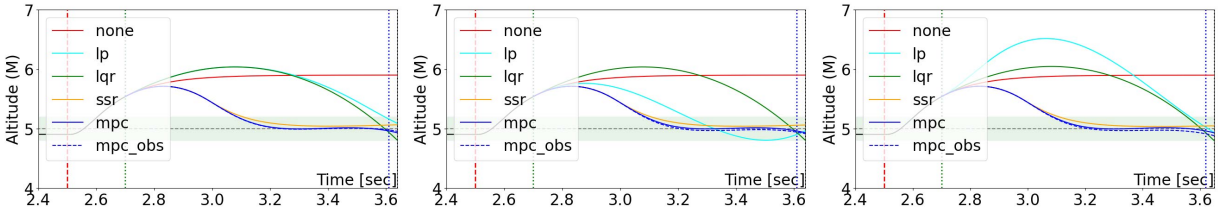


Fig. 12: Sensitivity analysis of recovery with an observer to increasing noise with upper bounds of $\{0.05, 0.1, 0.25\}$ $\times 10^{-3}$ on the Quadrotor benchmark.

(iii) linear-programming-based recovery (lp) [18]. 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) [19]. 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.

E. Recovery Effect

We compared the proposed method with four baselines for the three benchmarks. We plot the actual system physical states in Fig. 8, 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.

F. 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 Fig. 9: 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. Furthermore, 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. Fig. 10 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 Fig. 8 and analyses in Figs. 9, 10, 9, 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 benchmark (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 Fig. 13). Moreover, as seen in the left plot of Fig. 12, the observer does not significantly affect recovery at lower noise thresholds. With increased noise, as seen in the right plot of Fig. 12, 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.

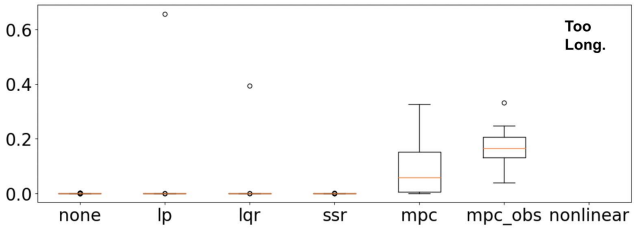


Fig. 13: Computational overhead (in seconds) for all methods on the Quadrotor benchmark. For both LP and LQR, the outlier point on top represents the overhead in formulating and solving the problem at time step t_f.

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. Fig. 11 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. Fig. 13 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, 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 Fig. 13, there is a trade-off between usability and recovery performance; and our proposed method delivers both.

VI. Conclusion

In this work, 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.

ACKNOWLEDGEMENTS

This work was supported in part by NSF CNS-2143256, NSF CNS-2143274, ONR N00014-20-1-2744, and the Air Force under MOU 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 the Air Force, the National Science Foundation (NSF), the Office of Naval Research (ONR), or the U.S. Government.

REFERENCES

[1] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2011, pp. 163–168.

[2] N. H. Motlagh, T. Taleb, and O. Arouk, "Low-altitude unmanned aerial vehicles-based internet of things services: Comprehensive survey and future perspectives," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 899–922, 2016.

[3] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[4] N. Adam, "Workshop on future directions in cyber-physical systems security," in *Report on workshop organized by Department of Homeland Security (DHS)*, 2010.

[5] 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.

[6] S. Chaterji, P. Naghizadeh, M. A. Alam, S. Bagchi, M. Chiang, D. Corman, B. Henz, S. Jana, N. Li, S. Mou *et al.*, "Resilient cyberphysical systems and their application drivers: A technology roadmap," *arXiv preprint arXiv:2001.00090*, 2019.

[7] S. Parkinson, P. Ward, K. Wilson, and J. Miller, "Cyber threats facing autonomous and connected vehicles: Future challenges," *IEEE transactions on intelligent transportation systems*, vol. 18, no. 11, pp. 2898–2915, 2017.

[8] M. Amoozadeh, A. Raghuramu, C.-N. Chuah, D. Ghosal, H. M. Zhang, J. Rowe, and K. Levitt, "Security vulnerabilities of connected vehicle streams and their impact on cooperative driving," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 126–132, 2015.

[9] A. M. Wyglinski, X. Huang, T. Padir, L. Lai, T. R. Eisenbarth, and K. Venkatasubramanian, "Security of autonomous systems employing embedded computing and sensors," *IEEE micro*, vol. 33, no. 1, pp. 80–86, 2013.

[10] 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.

[11] T. He, L. Zhang, F. Kong, and A. Salekin, "Exploring inherent sensor redundancy for automotive anomaly detection," in *57th Design Automation Conference*. ACM, 2020.

[12] 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.

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

[14] 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.

[15] 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.

[16] A. H. Rutkin, "spoofers use fake gps signals to knock a yacht off course," MIT Technology Review, 2013, online; accessed May 2020.

[17] 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.

[18] 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.

[19] 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. [Online]. Available: https://doi.org/10.1145/3477010

[20] 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*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 919–924. [Online]. Available: https://doi.org/10.1145/3489517.3530555

[21] 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.

[22] 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.

[23] 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.

[24] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell, "A survey of physics-based attack detection in cyber-physical systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[25] 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.

[26] 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.

[27] 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.

[28] P. Guo, H. Kim, N. Virani, J. Xu, M. Zhu, and P. Liu, "Roboads: Anomaly detection against sensor and actuator misbehaviors in mobile robots," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 574–585.

[29] ——, "Exploiting physical dynamics to detect actuator and sensor attacks in mobile robots," *arXiv preprint arXiv:1708.01834*, 2017.

[30] A. Taylor, S. Leblanc, and N. Japkowicz, "Anomaly detection in automobile control network data with long short-term memory networks," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2016, pp. 130–139.

[31] A. Ganesan, J. Rao, and K. Shin, "Exploiting consistency among heterogeneous sensors for vehicle anomaly detection," SAE Technical Paper, Tech. Rep., 2017.

[32] M. Müter, A. Groll, and F. C. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *2010 Sixth International Conference on Information Assurance and Security*. IEEE, 2010, pp. 92–98.

[33] 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.

[34] F. A. T. Abad, R. Mancuso, S. Bak, O. Dantsker, and M. Caccamo, "Reset-based recovery for real-time cyber-physical systems with temporal safety constraints," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–8.

[35] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Guaranteed physical security with restart-based design for cyber-physical systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 10–21.

[36] J. Shin, Y. Baek, J. Lee, and S. Lee, "Cyber-physical attack detection and recovery based on rnn in automotive brake systems," *Applied Sciences*, vol. 9, no. 1, 2019.
[37] K. Sridhar, R. Ivanov, V. Lesi, M. Juliato, M. Sastry, L. Yang, J. Weimer, O. Sokolsky, and I. Lee, "A framework for checkpointing and recovery of hierarchical cyber-physical systems," *arXiv preprint arXiv:2205.08650*, 2022.
[38] 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.
[39] N. Nower, Y. Tan, and A. O. Lim, "Efficient temporal and spatial data recovery scheme for stochastic and incomplete feedback data of cyber-physical systems," in *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, 2014, pp. 192–197.
[40] S. Z. Yong, M. Zhu, and E. Frazzoli, "Resilient state estimation against switching attacks on stochastic cyber-physical systems," in *2015 54th IEEE Conference on Decision and Control (CDC)*. IEEE, 2015, pp. 5162–5169.
[41] 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.
[42] ——, "A data-driven model predictive control for alleviating thermal overloads in the presence of possible false data," *IEEE Transactions on Industry Applications*, vol. 57, no. 2, pp. 1872–1881, 2021.
[43] 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.
[44] Y. Tamir and C. H. Sequin, "Error recovery in multicomputers using global checkpoints," in *13th International Conference on Parallel Processing (ICPP)*, 1984, pp. 32–41.
[45] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 1, pp. 63–75, 1985.
[46] K. G. Shin, T.-H. Lin, and Y.-H. Lee, "Optimal checkpointing of real-time tasks," *IEEE Transactions on Computers (TC)*, vol. 100, no. 11, pp. 1328–1341, 1987.
[47] D. B. Johnson, "Distributed system fault tolerance using message logging and checkpointing," Ph.D. dissertation, Rice University, 1990.
[48] K.-F. Ssu, B. Yao, and W. K. Fuchs, "An adaptive checkpointing protocol to bound recovery time with message logging," in *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*. IEEE, 1999, pp. 244–252.
[49] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 375–408, 2002.
[50] 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.
[51] M. Berz, *Modern Map Methods in Particle Beam Physics*, ser. Advances in Imaging and Electron Physics. Academic Press, 1999, vol. 108.
[52] 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.
[53] 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.
[54] X. Chen, "Reachability analysis of non-linear hybrid systems using taylor models," Ph.D. dissertation, RWTH Aachen University, 2015.
[55] X. Chen and S. Sankaranarayanan, "Decomposed reachability analysis for nonlinear systems," in *Proc. of RTSS'16*, 2016, pp. 13–24.
[56] 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.
[57] A. J. Sørensen, "Marine cybernetics, towards autonomous marine operations and systems," *Department of Marine Technology, NTNU*, 2018.
[58] J. D. Hedengren, "A nonlinear model library for dynamics and control," *Yeast*, vol. 7, p. 24, 2008.
[59] 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.
[60] F. Sabatino, "Quadrotor control: modeling, nonlinearcontrol design, and simulation," 2015.
[61] R. Giorgiani do Nascimento, K. Fricke, and F. Viana, "Quadcopter control optimization through machine learning," in *AIAA Scitech 2020 Forum*, 2020, p. 1148.
[62] 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.
[63] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.