Real-time Attack-recovery for Cyber-physical Systems Using Linear-quadratic Regulator

LIN ZHANG, Syracuse University, USA
PENGYUAN LU, University of Pennsylvania, USA
FANXIN KONG, Syracuse University, USA
XIN CHEN, University of Dayton, USA
OLEG SOKOLSKY and INSUP LEE, University of Pennsylvania, USA

The increasing autonomy and connectivity in cyber-physical systems (CPS) come with new security vulnerabilities that are easily exploitable by malicious attackers to spoof a system to perform dangerous actions. While the vast majority of existing works focus on attack prevention and detection, the key question is "what to do after detecting an attack?". This problem attracts fairly rare attention though its significance is emphasized by the need to mitigate or even eliminate attack impacts on a system. In this article, we study this attack response problem and propose novel real-time recovery for securing CPS. First, this work's core component is a recovery control calculator using a Linear-Quadratic Regulator (LQR) with timing and safety constraints. This component can smoothly steer back a physical system under control to a target state set before a safe deadline and maintain the system state in the set once it is driven to it. We further propose an Alternating Direction Method of Multipliers (ADMM) based algorithm that can fast solve the LQR-based recovery problem. Second, supporting components for the attack recovery computation include a checkpointer, a state reconstructor, and a deadline estimator. To realize these components respectively, we propose (i) a sliding-window-based checkpointing protocol that governs sufficient trustworthy data, (ii) a state reconstruction approach that uses the checkpointed data to estimate the current system state, and (iii) a reachability-based approach to conservatively estimate a safe deadline. Finally, we implement our approach and demonstrate its effectiveness in dealing with totally 15 experimental scenarios which are designed based on 5 CPS simulators and 3 types of sensor attacks.

CCS Concepts: • **Security and privacy** → *Security requirements*; • **Computer systems organization** → **Embedded and cyber-physical systems**; **Real-time systems;**

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Embedded Software (EMSOFT), 2021.
This research was supported in part by NSF CCF-2028740, ONR N00014-17-1-2012, ONR N00014-20-1-2744, and AFRL under contract number FA8650-16-C-2642. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF), Office of Naval Research (ONR), U.S. Air Force Research Laboratory (AFRL), the Department of Defense, or the United States Government.
Authors' addresses: L. Zhang and F. Kong, Syracuse University, Syracuse, New York, USA, 13244; emails: {lzhan120, fkong03}@syr.edu; P. Lu, O. Sokolsky, and I. Lee, University of Pennsylvania, Philadelphia, Pennsylvania, USA, 19104; email: pelu@seas.upenn.edu, {sokolsky, lee}@cis.upenn.edu; X. Chen, University of Dayton, Dayton, Ohio, USA, 45469; email: xchen4@udayton.edu.

1539-9087/2021/09-ART79 $15.00
https://doi.org/10.1145/3477010

Additional Key Words and Phrases: Cyber-physical system, security, sensor attack, recovery, real-time, linear-quadratic regulator

ACM Reference format:
Lin Zhang, Pengyuan Lu, Fanxin Kong, Xin Chen, Oleg Sokolsky, and Insup Lee. 2021. Real-time Attack-recovery for Cyber-physical Systems Using Linear-quadratic Regulator. *ACM Trans. Embedd. Comput. Syst.* 20, 5s, Article 79 (September 2021), 24 pages.
https://doi.org/10.1145/3477010

1 INTRODUCTION

Cyber-Physical Systems (CPS) tightly couple computing and communication with physical processes via sensing and actuation components. The integration of new technologies to interact and control physical systems enables applications and services, such as autonomous and connected vehicles, unmanned aerial vehicles, and unmanned delivery, that promise enormous societal and economic benefits. Meanwhile, these benefits come with new security vulnerabilities beyond the classical cyber security domain that mainly focuses on information technology systems [10, 28, 33]. Exploiting them, malicious attackers can spoof the system to perform dangerous actions and further cause catastrophic consequences.

One crucial security risk of CPS is sensor attacks, that is, attacks that alter sensor data to negatively interfere with the physical system. Fed with malicious sensor information, a controller obtains corrupted state estimates and may drive the system to unsafe physical states. Sensor attacks can be launched from cyber attack surfaces such as compromising control software or the network between sensors and the controller. Different from these convectional vectors, an emerging attack, know as transduction attacks, non-invasively affects sensor readings through manipulating a physical properties to allow injection of malicious signals to sensors [46, 58, 59]. For example, an attacker can compromise wheel speed sensors to impact antilock braking systems [51], spoof GPS signals to misguide a yacht [47], or corrupt a LiDAR sensor remotely to make a vehicle perceive non-existent objects [45]. Moreover, exclusively using cyber security is inadequate to secure CPS against these attacks [33, 44, 46]. Further, the effectiveness of these attacks will continue to increase with the rise of CPS autonomy.

These new threats have motivated many research efforts on defending against sensor attacks. Most works fall into the two categories: prevention and detection. The first category prevents attacks from negatively impacting the physical system, for instance, attack-resilient sensor fusion [32] or state estimation [44]. The second category detects sensor attacks by monitoring the differences between observed and expected (or estimated) sensor data. Attack detectors usually utilize system models (i.e., trained models or physical dynamics) [24, 46] or sensor correlation [20, 28].

In spite of the extensive volume of works on attack detection, a key question that remains rarely addressed is what to do after detecting an attack. A recent survey that summarizes 32 CPS security papers also raises the same question and points out the future work of attack response [23]. Malicious sensor information can make a controller generate misleading inputs that drive a physical system away from the desired state. The system may deviate to the unsafe region in the end if no responding measures are taken for an attack. Therefore, we believe that it is important to take advantage of benefits from attack detection, discontinue this deviation, and remove the negative impact caused by the attack to the system [8, 24]. Further, several other surveys on CPS attack detection such as [1, 24] also emphasize the scarcity of attack response work. Thus, this article attempts to fill this gap and propose techniques to recover the physical state of a CPS as the response to sensor attacks.

To achieve this goal is challenging. First, attack detection usually comes with some detection delay, i.e., the time interval between the start of an attack and its detection. During the delay, a CPS might have considerably drifted off and towards the unsafe region. The system should be recovered before touching unsafe states or causing serious consequences, for instance, before an autonomous vehicle hit an obstacle. We call this timing constraint as the safety deadline, after which the system may reach the unsafe set. Second, it is infeasible to offline determine the safety deadline due to two reasons. One is that the safe deadline changes as the physical state varies over time [59]. The other is that it is impossible to forecast when an attack occurs beforehand or the physical state at that time. Hence, an appropriate recovery design needs to accommodate the variety of the safety deadline.

Existing works are incapable of solving this real-time recovery problem. First, one common method to respond to a detected attack is to isolate the compromised sensors, derive state estimates from virtual sensors, and then continue to use the original controller to control the system [3, 12, 19, 33, 38, 39]. However, there are two major issues for this method. One is that the original controller implements a mild policy that is not fast enough to restore the system before the safety deadline. The other is that the original controller may be not sufficiently robust to avoid unsafe states during the recovery. Second, as to the control domain, robust control techniques are focused on how to tolerate bounded disturbance or errors [60]. Thus, they are incapable dealing with sensor attacks where sensor readings may be arbitrarily modified by attackers [8, 43]. In addition, control methods that have static policies are suitable to the real-time recovery problem here because of the need to handle the varying safety deadline.

To solve these issues, this article proposes a new recovery control that can (i) safely recover a system in real time with a smooth control trajectory, and (ii) maintain the system state in a target set (at least for a certain amount of time) once it is driven to the set. To realize such a controller, our work makes the following major contributions.

- The core component of the recovery controller is a control calculator based on a Linear-Quadratic Regulator (LQR) with constraints. The time horizon of LQR is set as a safety deadline plus a conservatively estimated time that the system is maintained in the target set (named maintainable time) when under an attack. This setting, together with appropriate constraints on the system state, can achieve the goals (i)(ii) as above. Then, we propose an Alternating Direction Method of Multipliers (ADMM) based algorithm for the LQR-based recovery problem. The algorithm iteratively solves the problem and fast converges to the optimal solution.
- Supporting components of the recovery controller include a checkpointer, a state reconstructor, and a deadline estimator. First, we propose a sliding-window-based checkpointing protocol that considers a varying detection delay, removes false data and keeps sufficient trustworthy data for attack recovery computations. Second, in the presence of an attack, the information of the sensors is no longer trustworthy. Thus, we present a state reconstruction approach that considers the computational overhead and uses the checkpointed data to estimate the system state when recovery sequence begins to be applied. Third, to determine an appropriate length for the recovery control sequence, we develop a reachability based approach to calculate a safety deadline and an approach to conservatively estimate the maintainable time.
- We implement our framework and evaluate it by using 5 CPS simulators under 3 sensor attack scenarios. The results demonstrate the efficiency and efficacy of our design and techniques.

The rest of the article is organized as follows. Section 2 discusses the background and related work. Section 3 introduces preliminaries and design overview of the recovery framework. Sections 4 and 5 present the design of each component of the framework in detail. Section 6 validates our approach and Section 7 concludes the article.

2 BACKGROUND AND RELATED WORK

CPS Security and Current Research Focus. Before concerning CPS security, researchers have paid much attention to deal with faults in control systems. Extensive studies exist in the literature on identifying bad data, tracking the source of the anomaly, and recovering from faults [31]. For example, many works rely on redundant sensors to cross-validate their readings for detection and remove faulty sensors for recovery. However, these works are inadequate to provide security against anomalies caused by malicious attackers because of two reasons. One reason is that these studies generally assume faults are caused by nature and thus model an independent and non-malicious failure behavior. The other is that an adversary can easily bypass the fault protections by violating these assumptions [8, 9, 18, 24, 41]. Hence, overcoming the limitations needs more powerful defense methods against CPS attacks.

This need has motivated many research works on defending against sensor attacks. These works generally fall into two groups: prevention and detection. In the first group, one area that has drawn much attention is to tolerate sensor attacks by sensor redundancy so as to prevent corrupted sensors from affecting the physical system. The sensor redundancy involves homogeneous sensors (e.g., multiple encoders measuring the vehicle speed) and heterogeneous sensors that can measure the same physical parameter (e.g., encoders and GPS sensors: the speed can also be calculated using the GPS information). The value fed into the controller is produced by fusing readings of the redundant sensors. Existing works usually assume that there is at least triple modular redundancy and less than half of the redundant sensors can be compromised [32, 37, 44]. Given that these assumptions hold, the corrupted sensors can be tolerated and the fused sensor data is treated as trustworthy.

In the second group, one area that has drawn much attention is how to utilize physical invariants to detect sensor attacks. A physical invariant is defined as an invariant guarded by certain physical laws. There are two kinds of physical invariants commonly adopted in the literature. The first kind uses a system model to capture the physical system's dynamics [13, 24, 46]. For example, a set of differential equations [14] or a machine learning model can be used to describe the motion of a quadcopter. The second kind of physical invariant refers to sensor correlation, where multiple (heterogeneous) sensors correlatively respond to the same physical aspect at the same time [2, 28, 54]. For example, pressing the accelerator will increase engine RPM and vehicle speed as well as affect GPS readings. There are usually two phases for attack detection. The offline learning phase is to extract the physical invariant of a system. The online detection phase is to monitor sensor (and actuation) values from physical observations, and identify anomalies between these values and the expected ones given by the physical invariant.

Research Problem and Scope. There is much less attention on recovering CPS from sensor attacks, compared to the large body of the literature on attack detection. As noted above, the need to remove the impacts caused by sensor attacks to a system emphasizes the importance of the attack recovery problem studied in this article. To be specific, the proposed recovery is a responsive procedure that is used with detection methods. This work considers that there is an attack detector already in place, and the objective is to take in the alerts that this detector generates and respond to them for recovering the physical system. The execution of the recovery is tightly tied to the capability of the attack detector. For example, if a false negative occurs, i.e., the detector fails to detect an attack, the recovery will not be triggered and thus cannot help defend against the attack.

The recovery problem studied in this work differs from the recovery concept defined purely for computer systems. The latter, called cyber recovery, is confined to the cyber part because its scope is on recovering computing tasks. The state of the cyber recovery refers to computing information such as variable values in the program or the cyber state. The conventional method is that after a

fault is detected, the computing tasks will be rolled back to a globally consistent state checkpointed in the history [17]. By contrast, this article aims to restore the state of a physical system or the physical state.

Inadequacy of Existing Works. One possible method to react to a detected attack is to restart a CPS. However, a physical system may need a great amount of time to shutdown and reboot [30, 57, 59]. During the time, a CPS is taken offline and serious consequences may occur. Hence, a better method needs to respond to detected attacks in an online manner. For example, as to sensor attacks, we may just restart the attacked sensors instead of the whole CPS. During the sensor reboot time, a recovery method is needed to stop a system from further drifting and bring the system to a safe state. Further, as mentioned, the online recovery needs to meet the timing constraint. That is, this article pursues real-time attack-recovery that brings a CPS back to a safe state before the safety deadline. In the following, we will discuss two major groups of related work that are from two different research communities, respectively, and give the reason why they are inapplicable or inadequate to address this recovery problem.

The first group of related work includes studies from the cyber security community. As noted above, a commonly used attack response approach is to obtain state estimates for the corrupted sensors and continue to use the original controller to restore the physical system [3, 12, 19, 33, 38, 39]. Actually, this approach just performs state prediction using a system model that is pre-known or learnt, which is only one component in the recovery framework in Section 3. Notably, several important aspects for recovering a CPS under attack are missed in these existing works. First, they do not consider the timing constraint, i.e., the safety deadline, and thus cannot guarantee a timely recovery. Second, they still rely on the original controller and thus cannot ensure that no unsafe states will be touched during the recovery.

The second group of related work includes studies from the control community. First, conventional robust control approaches are good at tolerating disturbance and modeling errors that can be bounded [25, 60]. However, they are inadequate to defend against sensor attacks because the impacts caused by sensor attacks are hard to bound as attackers may arbitrarily manipulate sensor readings [8, 9, 35, 43]. Similarly, Kalman filter based approaches are also insufficient to handle such malicious sensor attacks [41, 56]. Further, the timing constraint, i.e., the safety deadline, is not considered by these approaches. Second, as to the hierarchical control architecture, also called the multi-mode architecture sometimes, the control policy for each mode is usually static. Using static control polices is not suitable for the recovery problem here, because as mentioned above, the safety deadline varies at run time and it is infeasible to determine the deadline beforehand. To handle the varying deadline, We need a dynamic control policy that can generate recovery control to adapt to different deadlines on the fly. Lastly, from the concept level, our recovery problem is different from event-driven control. Although the event of attack detection triggers the recovery controller, the recovery control is time-driven because the generated control actions are still applied periodically [29, 50]. More discussion on the recovery framework is in Section 3.2.1.

Closest Related Work. The closest related work to this article is [59], which is regarded as the first work of its kind in online and timely recovery from sensor attacks in CPS. This work develops a recovery controller that is automatically activated to take over a system after an attack is detected. The controller steers back to a safe state in real-time in presence of the attack, and guarantees that no unsafe states are reachable along the recovery process. This work provides an initial solution to the problem, but leaves several important issues unsolved.

First, the work overlooks the control performance during the recovery. More precisely, there is considerable oscillation in the resulted recovery trajectory (as shown in Section 6). Second, the work just focuses on bringing a system back to a target state set, but it is not clear how to maintain the system state in that set afterwards. That is, the system state may move outside the target set

Table 1. Notations and Symbols Used in This Article

Notation	Description
δ	control stepsize/control interval
$\boldsymbol{x}_t$	state estimate at time t
$\boldsymbol{u}_t$	control input to be implemented at time t
$\bar{\boldsymbol{x}}_t$	real/true/actual system state at time t
t_w	the time when the system state is the last trustworthy state
t_0	the time when the sensor attack starts
t_a	the time when the sensor attack is detected
t_r	the time when the recovery control sequence begins to be implemented
t_d	safe deadline, the end time of the recovery period
t_l	the end time of the maintenance period or maintainable time
D	recovery length, $D = t_d - t_r$
M	maintenance length, $M = t_l - t_d$
N	total recovery control length, $N = t_l - t_r$
$\boldsymbol{v}_t$	the uncertainty at time t
v_{max}	the maximum value of the uncertainty
n	the dimension of system state vector
m	the dimension of control input vector
J	the objective function of optimization problem
Q	the state cost
Q_f	the final state cost
R	the control input cost
$\oplus$	Minkowski sum, i.e., $X \oplus Y = \{x + y \mid x \in X, y \in Y\}$
$\ominus$	Minkowski difference, i.e., $X \ominus Y = \bigcap_{y \in Y} \{x - y \mid x \in X\}$

again if using inappropriate control inputs after the recovery. Third, the work assumes the recovery control sequence can be applied immediately once an attack is detected, but the computational overhead of the sequence is non-negligible for complex systems. Fourth, the checkpointing protocol does not consider a varying detection delay, which is common for attack detectors. This article addresses these key issues by proposing new recovery control techniques.

3 PRELIMINARIES AND DESIGN OVERVIEW

In this section, we present the system model, the threat model, and the design overview of the real-time attack-recovery framework. Table 1 summarizes the notations and symbols used in this article.

3.1 System and Threat Model

Under our consideration, the cyber-physical system model is a physical process (or a plant) controlled by a controller. The controller operates at every *control step*, denoted by a positive value δ. When a control step starts, the controller reads sensor measurements and computes the plant's state estimate, denoted by a set of real-valued variables $\{x_1, \ldots, x_n\}$. Base on it, the controller generates the control signals $\{u_1, \ldots, u_m\}$ through the control algorithm and then sends the result to the actuators. The actuator will apply the control inputs to the plant in the current step. For brevity, we collectively represent the variables $\{x_1(t), \ldots, x_n(t)\}$ by $\boldsymbol{x}(t)$, i.e., $\boldsymbol{x}(t) \in \mathbb{R}^n$, and $\{u_1(t), \ldots, u_m(t)\}$

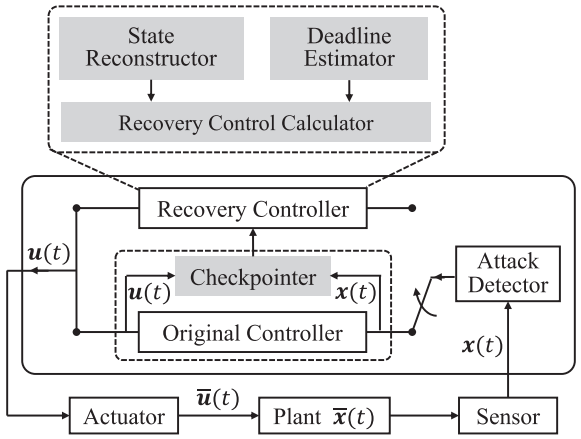


Fig. 1. The Framework of Real-Time Attack-Recovery.

by $\boldsymbol{u}(t)$, i.e., $\boldsymbol{u}(t) \in \mathbb{R}^m$, at time t. For easy presentation of equations, we sometimes use x_t or u_t to denote $x(t)$ or $u(t)$, respectively.

The parameter $\boldsymbol{x}(t)$ denotes the state estimate, while $\bar{\boldsymbol{x}}(t)$ is the real (true) state of the plant. The parameter $\boldsymbol{u}(t)$ denotes the control input computed by the controller, while $\bar{\boldsymbol{u}}(t)$ is the real input to the plant. We assume that the plant is fully observable to the sensor.

Threat Model. We consider a malicious attacker who can launch sensor attacks, that is, to alter the sensor data before sent to the controller so as to compromise the integrity and availability of sensor measurements in CPS.

• *Compromising Integrity of Sensor Data.* The attack scenarios we consider under this category are bias and reply attacks. (i) Bias attacks. This kind of attack modifies sensor data by adding or subtracting certain values. That is, the value $\boldsymbol{x}(t)$ can be set to be $\bar{\boldsymbol{x}}(t) \pm \boldsymbol{e}$ by an attack, where $\boldsymbol{e}$ is the modification value. Our work has no restriction on the number of the system state that can be compromised. For example, it can $\boldsymbol{e} = (e_1, \ldots, e_n)^T, e_i \neq 0, \forall i$, all dimensions compromised, or $\boldsymbol{e} = (0, \ldots, 0, e_j, \ldots, e_n)^T$, partial dimensions compromised, starting from the beginning of the attack. (ii) Replay attacks. This kind of attack sends historical sensor data instead of current ones to the controller. That is, $\boldsymbol{x}(t) = \bar{\boldsymbol{x}}(t - s)$ starting from the attack for some $s > 0$.

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

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

3.2 Overview of the Real-time Attack-recovery Framework

This work follows the novel real-time recovery framework first proposed in [59], and updates each component as mentioned in Section 1. For completeness, the following introduces the framework, which is illustrated in Figure 1. The framework has two operating modes: *normal* and *recovery* mode. The attack detector determines whether the system is under attack. Once the detector

identifies an attack, the system will be switched from the normal mode to the recovery mode. As mentioned in the introduction, the attack detection is outside the scope of our article, and we assume an existing detection method that works with our recovery, such as [13, 24, 28, 40, 46]. Although attack detection is a flourishing track in CPS security, how to extend its main benefits and secure the system is still an open question. This article aims to fill this gap.

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

Recovery Mode. Switching to the recovery mode, the *recovery controller* takes over the system. The recovery controller consists of three components, as shown by the shaded boxes in Figure 1: (i) recovery control calculator, (ii) state reconstructor, and (iii) deadline estimator. The following briefly describes these components, and we will provide their detailed design in Sections 4 and 5.

• *Recovery Control Calculator.* It takes a substantial time for attack detectors to identify the attack after it is launched [24, 33, 46]. During this delay, the attack may drive the system to an undesired state. Thus, this component can compute a Piece-Wise Constant (PWC) control sequence that restores the system from a compromised state into a target state set within a safety deadline and maintain the system in this set before the maintainable time. The control sequence starts from a time point that is close to when an attack is detected. The initial states and two deadlines are obtained from the following two supporting components.

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

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

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

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

Second, this framework can be seen as an extension of the simplex architecture [15, 42, 55], which consists of a "complex" controller and a "safety" controller. Our framework extends this architecture by adding the four new components, checkpointer, state reconstructor, deadline estimator, and recovery control calculator, as shown in Figure 1.

Third, the proposed framework is different from event-driven control. In our framework, although the recovery controller is triggered by an event of attack detection, the controller is periodic or time-triggered, i.e., with equidistant sampling intervals. That is, the event of the detection of an attack only makes the system switches from the original controller to the recovery controller, which is still time-triggered. By contrast, the sampling is event-triggered in the regime of event-driven control [29, 50]. Hence, event-driven control is inapplicable to our recovery problem.

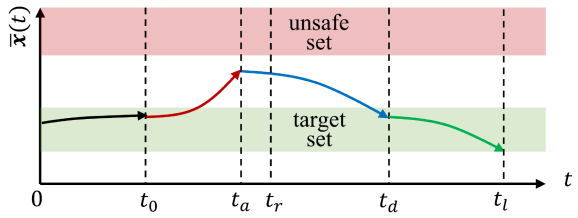


Fig. 2. Recovering a system under an sensor attack.

Fourth, this framework handles multiple types of sensor attacks, which are illustrated in Section 3.1. By contrast, some works require the attack to belong to a particular class. For example, some works from networked control systems (NCS) domain pay more attention to the delay attack. They explore to stabilize NCS in the presence of network-induced delay that is inherent to NCS [7, 21].

3.2.2 Recovery Control Sequence. We illustrate an example of the use of recovery control in Figure 2. The system works normally from the start of the time $t = 0$. A sensor attack starts at the time $t = t_0$ and is detected by the attack detector at the time $t = t_a$. Here, we do not require a particular value for t_a but assume that during the time interval $[t_0, t_a]$, although the system is drifted by the attack, no unsafe state is reached. It can be fulfilled by a well-designed attack detector. Our recovery framework takes over the control of the system at $t = t_a$, it firstly computes a recovery control sequence, and then applies it at the time $t = t_r$. The value of $t_r - t_a$ is the maximum bound for the computational overhead of the recovery control and we assume that it can be conservatively estimated by offline-profiling. The framework is required to obtain the recovery control before t_r.

The recovery control is a piecewise constant sequence which should satisfy the following properties. (i) It consists of two parts: the first D steps, $D = t_d - t_r$, is the *recovery period*, while the second part is the *maintenance period* which has M steps where $M = t_l - t_d$. (ii) The D control steps in the recovery period are guaranteed to steer the system to a state in the target set without reaching any unsafe state. (iii) After the system is recovered, the M control steps in the maintenance period can still keep the system in the target set till the time $t = t_l$. The reason to have the maintenance period is to allow the system to tolerate attacks after the recovery period [33]. For example, to have a time period for resetting the attacked sensors to make them trustworthy again if possible. After the maintenance period, the control of the system is given back to the original controller.

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

4 LQR BASED RECOVERY CONTROL CALCULATOR

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

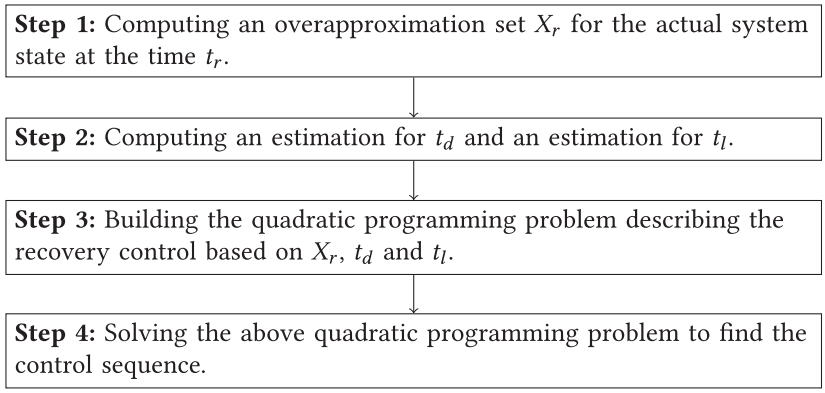


Fig. 3. High-level description of our approach to find a recovery control.

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

4.1 Recovery Problem Formulation

We consider an LTI system and the dynamics is given by Equation (1),

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

where $\boldsymbol{x}_t \in \mathbb{R}^n$ denotes the plant's state vector at time t, $\boldsymbol{u}_t \in \mathbb{R}^m$ is the control input vector, $\boldsymbol{v}_t \in \mathbb{R}^n$ is the uncertainty vector, and A, B have suitable dimensions. We assume that the uncertainty $\boldsymbol{v}_t$ at any time is constrained by a bounded range V, and use v_{max} to denote its magnitude $\sup_{\boldsymbol{v}\in V} \|\boldsymbol{v}\|$ where $\|\cdot\|$ is the euclidean norm. The LTI model has been widely used in both control and security communities [13, 25, 33, 59]. Given the current system state and control input, we can predict next system state from the system model, which represents the system's behavior.

The Linear Quadratic Regulator (LQR) [34] is a well-known optimal feedback controller in control community. It describes the cost as two quadratic terms of states $\boldsymbol{x}$ and control input $\boldsymbol{u}$, which jointly considers the control performance and actuator effort.

As in Figure 2, the (absolute) safe deadline is t_d, estimated by the deadline estimator, and there are $D = t_d - t_r$ control steps before this deadline. In order to stabilize the system states, $M = t_l - t_d$ control steps are added after t_d, and the physical states are restrained within target set X_T during the M control steps. We use LQR with discrete-time finite horizon to formulate our problem. This recovery process is an optimization problem, and we define the objective function as

$$J(\boldsymbol{x}_r, \ldots, \boldsymbol{x}_l, \boldsymbol{u}_r, \ldots, \boldsymbol{u}_{l-1}) = \sum_{i=r}^{l-1} \left(\boldsymbol{x}_i^T Q \boldsymbol{x}_i + \boldsymbol{u}_i^T R \boldsymbol{u}_i\right) + \boldsymbol{x}_l^T Q_f \boldsymbol{x}_l \tag{2}$$

where $\boldsymbol{x}_i$ is the states of LTI system during recovery, $\boldsymbol{u}_i$ is the input used in the i-th step in the recovery control, optimization horizon $N = D + M$ is the number of recovery steps from $t = t_r$ to t_l, and $Q, Q_f \in \mathbb{R}^{n\times n}, R \in \mathbb{R}^{m\times m}$ are semi-definite symmetric matrices that define the state cost,

final state cost, and input cost respectively. With properly designed Q, Q_f and R, minimizing J is to minimize the deviation of the states from the target set as well as the size of control signals required within the horizon.

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

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

$$\phi(\boldsymbol{x}_r, \ldots, \boldsymbol{x}_l, \boldsymbol{u}_r, \ldots, \boldsymbol{u}_{l-1}) := (\boldsymbol{x}_r = \text{center}(X_r)) \tag{3a}$$

$$\wedge \bigwedge_{i=r}^{l} (\boldsymbol{x}_i \oplus \mathcal{B}_i \subseteq X_S \ominus A^{i-r} \mathcal{I}_R) \tag{3b}$$

$$\wedge \bigwedge_{i=d}^{l} (\boldsymbol{x}_i \oplus \mathcal{B}_i \subseteq X_T \ominus A^{i-r} \mathcal{I}_R) \tag{3c}$$

$$\wedge \bigwedge_{i=r}^{l-1} (\boldsymbol{u}_i \in U) \wedge \bigwedge_{i=r}^{l-1} (\boldsymbol{x}_{i+1} = A\boldsymbol{x}_i + \boldsymbol{B}u_i) \tag{3d}$$

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

Referring to Figure 2, the sub-constraint (3a) denotes that the first state at $t = t_r$, $\boldsymbol{x}_r$ is the center of X_r. (3b) requires that the state $\boldsymbol{x}_i$ is always inside the safe set X_S, considering the accumulation of the uncertainty $\mathcal{B}_i$. (3c) requires that the state $\boldsymbol{x}_i$ is always within target set after the deadline t_d, still considering the uncertainty. Finally, (3d) defines the control input range and system dynamics, by which the evolution follows. The correctness of the formulation is proved in [59].

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

$$\min J(\boldsymbol{x}_r, \ldots, \boldsymbol{x}_l, \boldsymbol{u}_r, \ldots, \boldsymbol{u}_{l-1}) \qquad s.t. \quad \phi(\boldsymbol{x}_r, \ldots, \boldsymbol{x}_l, \boldsymbol{u}_r, \ldots, \boldsymbol{u}_{l-1}) \tag{4}$$

4.2 ADMM Algorithm

The alternating direction method of multipliers (ADMM) is an optimization algorithm that decomposes a large global problem into small local subproblems and coordinates to find the global solution [6, 27]. With ADMM, we can solve minimization problems with separable objectives and constraints in the form of:

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) + g(\boldsymbol{z}) \\ \text{s.t.} \quad & \boldsymbol{x} \in C_x, \ \boldsymbol{z} \in C_z, \ A\boldsymbol{x} + B\boldsymbol{z} = \boldsymbol{c} \end{aligned} \tag{5}$$

with variables $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{z} \in \mathbb{R}^m$, parameters $A \in \mathbb{R}^{p \times n}$, $B \in \mathbb{R}^{p \times m}$, and $\boldsymbol{c} \in \mathbb{R}^p$, and convex functions f and g. We form the augmented Lagrangian function for (5) as

$$L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \boldsymbol{y}^T (A\boldsymbol{x} + B\boldsymbol{z} - \boldsymbol{c}) + (\rho/2) \|A\boldsymbol{x} + B\boldsymbol{z} - \boldsymbol{c}\|_2^2 \tag{6}$$

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

ADMM solves problem using the following iterations:

$$\boldsymbol{x}^{(k+1)} := \underset{\boldsymbol{x}\in C_x}{\operatorname{argmin}} L_\rho\left(\boldsymbol{x}, \boldsymbol{z}^{(k)}, \boldsymbol{y}^{(k)}\right) \tag{7a}$$

$$\boldsymbol{z}^{(k+1)} := \underset{\boldsymbol{z}\in C_z}{\operatorname{argmin}} L_\rho\left(\boldsymbol{x}^{(k+1)}, \boldsymbol{z}, \boldsymbol{y}^{(k)}\right) \tag{7b}$$

$$\boldsymbol{y}^{(k+1)} := \boldsymbol{y}^{(k)} + \rho\left(A\boldsymbol{x}^{(k+1)} + B\boldsymbol{z}^{(k+1)} - \boldsymbol{c}\right) \tag{7c}$$

Each iteration includes an $\boldsymbol{x}$-minimization step (7a), a $\boldsymbol{z}$-minimization step (7b), and a dual update step (7c). Since the objective function is separable, (6) can be minimized over f and g separately in step (7a) and (7b). Usually, it is easier to solve these two sub-problems than to solve the global problem directly, and the total solving time can be reduced.

4.3 ADMM-Based Control Sequence Calculation

Our recovery problem (4) can be solved by ADMM method, because the optimization variables $\boldsymbol{x}$ and $\boldsymbol{u}$ are decoupled and thus separable. To construct a separable objective function and equality constraint, we define the following matrices:

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

Therefore, we obtain the matrix form of (2) as

$$J(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{x}^T \bar{Q}\boldsymbol{x} + \boldsymbol{u}^T \bar{R}\boldsymbol{u} \tag{8}$$

where $\boldsymbol{x} = [\boldsymbol{x}_0^T\ \boldsymbol{x}_1^T\ \cdots\ \boldsymbol{x}_N^T]^T \in \mathbb{R}^{n(N+1)}$, $\boldsymbol{u} = [\boldsymbol{u}_0^T\ \boldsymbol{u}_1^T\ \cdots\ \boldsymbol{u}_{N-1}^T]^T \in \mathbb{R}^{mN}$. The equality constraint (3d) becomes $\bar{A}\boldsymbol{x} + \bar{B}\boldsymbol{u} = 0$. Now, we can reformulate our problem in the form of (5) for ADMM method. Formally:

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) + g(\boldsymbol{u}) \\ \text{s.t.} \quad & (3a)(3b)(3c), \quad \bigwedge_{i=0}^{N-1}(u_i \in U), \quad \bar{A}\boldsymbol{x} + \bar{B}\boldsymbol{u} = 0 \end{aligned} \tag{9}$$

The objective function $J(\boldsymbol{x}, \boldsymbol{u})$ can be split into $f(\boldsymbol{x}) + g(\boldsymbol{u})$, where $f(\boldsymbol{x}) = \boldsymbol{x}^T\bar{Q}\boldsymbol{x}$ and $g(\boldsymbol{u}) = \boldsymbol{u}^T\bar{R}\boldsymbol{u}$. Then, we find the augmented Lagrangian function for (9):

$$L_\rho(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + g(\boldsymbol{u}) + \boldsymbol{\lambda}^T(\bar{A}\boldsymbol{x} + \bar{B}\boldsymbol{u}) + \frac{\rho}{2}\|\bar{A}\boldsymbol{x} + \bar{B}\boldsymbol{u}\|_2^2 \tag{10}$$

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

$$\boldsymbol{x}^{(k+1)} := \underset{\psi}{\operatorname{argmin}}\ L_\rho\left(\boldsymbol{x}, \boldsymbol{u}^{(k)}, \boldsymbol{\lambda}^{(k)}\right), \qquad \text{where } \psi = \ (3a)(3b)(3c) \tag{11}$$

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

$$\boldsymbol{u}^{(k+1)} := \underset{\gamma}{\operatorname{argmin}}\ L_\rho\left(\boldsymbol{x}^{(k+1)}, \boldsymbol{u}, \boldsymbol{\lambda}^{(k)}\right), \qquad \text{where } \gamma = \bigwedge_{i=0}^{N-1}(u_i \in U) \tag{12}$$

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

$$\boldsymbol{\lambda}^{(k+1)} := \boldsymbol{\lambda}^{(k)} + \rho\left(\bar{A}\boldsymbol{x}^{(k+1)} + \bar{B}\boldsymbol{u}^{(k+1)}\right) \tag{13}$$

Algorithm 1 shows the procedure to solve our recovery problem. First, at Line 1, we compute the coefficients of the separable objective function and the equality constraints, i.e., $\bar{Q}$, $\bar{R}$, $\bar{A}$ and

$\bar{B}$. Line 3–10 are the iterations of ADMM, which solve the two sub-problems (11) and (12) and do the dual update. Within every iteration, a residual term r is computed to estimate the convergence. When the $||r||_2^2$ is not greater than a tolerance ϵ, the algorithm converges and halts. We output the final control sequence $\boldsymbol{u}$.

ALGORITHM 1: ADMM-based Control Input Calculation

Input: Q, Q_f, R, ψ, γ, LTI model, N
/* Q: state cost matrix Q_f: final state matrix R: input cost matrix */
/* ψ: constrains on state $\mathbf{x}$ γ: constrain on control input $\mathbf{u}$ */
/* LTI model: state space model N: the number of recovery step */
Output: u
/* $u \in \mathbb{R}^{m \times N}$: control input in the last iteration */
1 Compute $\bar{Q}$ and $\bar{R}$ from Q and R, Compute $\bar{A}$ and $\bar{B}$ from LTI model and N
2 Initiate proper $\boldsymbol{u}^{(0)}$ and $\boldsymbol{\lambda}^{(0)}$
3 **for** $k \leftarrow 0$ *to MAX_ITERS* **do**
4 $\boldsymbol{x}^{(k+1)} \leftarrow \underset{\psi}{\operatorname{argmin}}\, L_\rho\left(\boldsymbol{x}, \boldsymbol{u}^{(k)}, \boldsymbol{\lambda}^{(k)}\right)$ ▷ $\boldsymbol{x}$-minimization
5 $\boldsymbol{u}^{(k+1)} \leftarrow \underset{\gamma}{\operatorname{argmin}}\, L_\rho\left(\boldsymbol{x}^{(k+1)}, \boldsymbol{u}, \boldsymbol{\lambda}^{(k)}\right)$ ▷ $\boldsymbol{u}$-minimization
6 $r \leftarrow \bar{A}\boldsymbol{x}^{(k+1)} + \bar{B}\boldsymbol{u}^{(k+1)}$ ▷ calculate residual
7 **if** $\|r\|_2^2 \leq \epsilon$ **then**
8 break ▷ stop condition
9 **else**
10 $\boldsymbol{\lambda}^{(k+1)} \leftarrow \boldsymbol{\lambda}^{(k)} + \rho\, r$ ▷ dual update

Theorem 4.1. *Starting from time $t = t_r$, the control input sequence $\boldsymbol{u}$ obtained from our LQR approach is able to recover the system to a state in the target set by the time $t = t_d$, and keep the system inside the target set till the time $t = t_l$.*

Remark 4.1. Due to the formal modeling and methods used to find the recovery control, the obtained control sequence is guaranteed to recover the LTI dynamics if it is applied at the time $t = t_r$. Therefore, our LQR approach is *sound*. On the other hand, in order to achieve good performance, conservative methods are used in overestimating the recovery initial state and the deadlines, which will be described in detail in Section 5. It is not guaranteed that our approach can always find a recovery control even there exists one. Hence, our approach is *not complete*.

Remark 4.2. The $O(\frac{1}{\epsilon})$ iteration complexity of ADMM algorithm is shown in [26], where ϵ is the tolerance.

5 SUPPORTING COMPONENTS FOR RECOVERY CONTROL

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

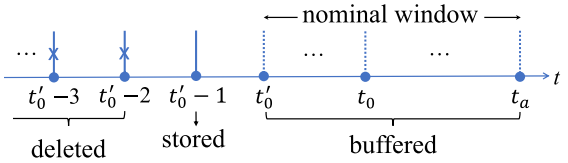


Fig. 4. Illustration of the Sliding Window Based Checkpointing Protocol.

5.1 A Sliding Window Based Checkpointing Protocol

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

The protocol proposed in the work [33] assumes a constant detection delay. However, as indicated in the attack detection work [24, 46, 53], the detection delay of approaches based on the cumulative sum of residuals (CUSUM) varies with factors such as attack scale and a drift parameter. To address these limitations, the new protocol can be viewed as a generalization that is applicable to the methods with both constant and varying detection delays.

This new protocol uses a nominal window to accommodate a varying attack detection delay. The length of this window equals the maximum delay of a detection approach. For example, the maximum delay can be analyzed by assuming the worst-case attack (e.g., a stealthy attack) given a drift parameter for CUSUM based detection [24, 46]. The real delay of the detection of an attack can be less than the nominal window. As shown in Figure 4, the interval $[t_0', t_a]$ denotes the maximum delay and $[t_0, t_a]$ is the true delay, where t_0 and t_a are the start and detection of an attack, respectively. The window slides forward as the time ticks. The protocol records estimate $\boldsymbol{x}(t)$ and control input $\boldsymbol{u}(t)$ by the following three steps: buffer, store, and delete.

- *Buffer.* Estimates and control inputs within the window, i.e., $\{(\boldsymbol{x}(t_0'), \boldsymbol{u}(t_0')), \ldots, (\boldsymbol{x}(t_a'), \boldsymbol{u}(t_a'))\}$ in $[t_0', t_a]$, are first buffered, because they may be already corrupted and it is still in question whether they are correct. Note that the recovery controller starts to run from the time t_a when the attack is detected. Thus, these data cannot be used for reconstructing $\boldsymbol{x}(t_a)$.
- *Store.* Estimates and control inputs that have moved outside the window are considered to be trustworthy. Thus, $(\boldsymbol{x}(t_0'-1), \boldsymbol{u}(t_0'-1))$ is stored.
- *Delete.* The stored data that is no longer needed will be deleted, e.g., $(\boldsymbol{x}(t_0'-2), \boldsymbol{u}(t_0'-2))$ is discarded.

When the detector raises an alarm, it gives us the actual time t_0 when the attack started (this can be done by finding the time of a breakpoint of the time series of the residuals [24, 46]). Since $t_0 \geq t_0'$, the data in $[t_0', t_0]$ has already passed the detection and is also considered as trustworthy. Hence, the data point $(\boldsymbol{x}(t_0-1), \boldsymbol{u}(t_0-1))$ will be used to rebuild $\boldsymbol{x}(t_a)$, instead of that of $t_0'-1$. Using data closer to the time of the detection of an attack can result in better reconstructed estimates. Further, it is worth noting that using the maximum delay as the nominal window guarantees that there is always trustworthy data for state reconstruction.

5.2 State Reconstruction

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

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

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

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

We then extend the expression of X_a to obtain still a linear expression for overapproximating the reachable state at the recovery start time $t = t_r$ based on the fact that the control input will be fixed at $\boldsymbol{u}(t_a)$ after an attack is detected:

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

where $N_r = t_r - t_w$, and $\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N_r-1}$ are the variable representations for the uncertainty from the time t_w to t_r. X_r is also derived through the evolution of the system states based on the LTI dynamics (1).

5.3 Deadline Estimation

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

Estimation of t_d. The purpose of using t_d is to set up a deadline for recovering the system. Since the computation of the optimal deadline requires to solve an optimization problem which is too costly, we use a heuristic to find a feasible control sequence for keeping the system (1) safe from any state in X_r, and set $t_d = t_r + D$ where D is the length of the control sequence. To do so, we assume that all of the control inputs used in the future steps from the time t_r are fixed at $\boldsymbol{u}(t_a)$, i.e., we use the constant inputs at the time t_a from the time t_r to estimate the system safety. Then we repeatedly verify the safety for the reachable set overapproximation X_k:

$$\begin{aligned} X_k &= A^k \boldsymbol{x}_0 + \sum_{i=0}^{k-1} A^i\, B\, \boldsymbol{u}(t_a) + \sum_{i=0}^{k-1} A^i\, \boldsymbol{v}_{i+N_r} \\ \boldsymbol{x}_0 &= X_r(\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N_r-1}),\ \text{and}\ \boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N_r+k-1} \in V \end{aligned} \tag{14}$$

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

Estimation of t_l. The purpose to use a maintenance deadline t_l which is no earlier than t_d is to make the recovery trajectories smoother than those without a maintenance period [59]. Unlike the safe deadline t_d, a later t_l might make the recovery problem harder to solve since it requires the recovery control to keep the system in the target set for a longer time. To obtain a reasonable estimate for t_l, we consider using an approach which is similar to the real-time monitoring technique described in [11]. We require that all of the reachable state at the time from $t_d + 1$ to $t_d + N$ which is t_l should be able to be kept in the target set. Hence, the reachable set overapproximation

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

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

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

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

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

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

Lemma 5.1 ([11]). *If the intersection $E_j \cap D_j$ is* nonempty *then there exists a control sequence to keep X_j in the target set.*

Since a Minkowski difference is often hard to compute, we resort to a more efficient but also conservative way to verify the emptiness of the intersection. We use the interval hull of the second operand in the Minkowski difference, and the result $\hat{D}_j$ is an underapproximation of the actual difference. Therefore, we may conservatively check the emptiness of $E_j \cap \hat{D}_j$ to verify whether a recovery control can maintain the system in the target set. We may do so using linear programming. We repeatedly check the intersection for $j = D + 1, \ldots$ until we meet an empty intersection or j reaches a given maximum bound $j_{\max}$. In the first case, t_l is set to be $t_r + j - 1$, and in the second case we set $t_l = t_r + j_{\max}$.

Remark 5.1. Solving the recovery control problem as a whole is difficult even the dynamics is linear, since we need to find all of the parameters such as control sequence length, safe deadline and control inputs by solving a single optimization problem in order to strictly keep their dependencies to obtain the optimal solution. To avoid the high computational cost, we decompose the problem to first estimate the deadline t_d and the "latest" maintainable time t_l, and then use them in the LP modeling of the recovery control problem. Although our method is not complete, i.e., it may not find a feasible recovery control in some cases, but the result is always sound, that is the control sequence found by our method is guaranteed to steer the system to the target set at the time t_d and maintain it there till the time t_l.

6 EVALUATION

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

6.1 Simulation Settings

6.1.1 Simulation Scenarios. We consider the following CPS models: vehicle turning, series RLC circuit, DC motor position, aircraft pitch, and quadrotor. The LTI models for these systems are obtained by linearization and discretization. All these simulators are LTI models, which are representative and used in both security and control works, such as [4, 13, 22, 33, 36, 48, 49, 52].

Vehicle Turning. The following ODE models the turning of a vehicle, which changes the speed difference between two wheels to steer [33]. The state x denotes the speed difference between two wheels, and control input u is the voltage difference applied to motors controlling the two wheels.

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

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

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

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

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

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

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

Quadrotor. A linear quadrotor model is described in [49]. The system consists of 12 state variables: $[x, y, z]^T$ and $[\phi, \theta, \psi]^T$ are linear and angular positions of the quadrotor in the earth frame. $[u, v, w]^T$ and $[p, q, r]^T$ are the linear and angular velocities in the body frame. The control input is denoted by u, in which f_t is total thrust and $[\tau_x, \tau_y, \tau_z]^T$ are the control torques caused by differences of rotor speeds.

Other simulation settings are listed in Table 2. For example, we use a PI controller ($K_P = 5, K_I = 5$) to update the control input u of series RLC circuit every 0.02 seconds. The attacks start at time $t_0 = 3s$, and are detected at time $t_a = 4.3s$. The system is safe when the capacitor voltage x_1 is in [0,7], and the target set of this scenario is [2.9, 3.1]. The voltage of source can be adjusted between

Table 2. Simulation Scenarios

Simulation Scenarios	Cyber-physical System Properties				Recovery-related Parameters				
	δ	X_S	U	PID	t_0	t_a	X_T	Q_i	R
Vehicle Turning	0.02	$x \in [-2.7, 2.7]$	$[-5, 5]$	0.5, 7, 0	4	4.5	$x \in [0.9, 1.1]$	$Q_1 = 10$	5
Series RLC Circuit	0.02	$x_1 \in [0, 7]$	$[-15, 15]$	5, 5, 0	3	4.3	$x_1 \in [2.9, 3.1]$	$Q_1 = 1$	0.1
DC Motor Position	0.1	$x_1 \in [-4, 4]$	$[-20, 20]$	11, 0, 5	6	9.9	$x_1 \in [-1.67, -1.47]$	$Q_1 = 10$	0.01
Aircraft Pitch	0.02	$x_3 \in [0, 2]$	$[-20, 20]$	14, 0.8, 5.7	3	4.3	$x_3 \in [0.68, 0.72]$	$Q_3 = 1$	1
Quadrotor	0.02	$z \in [-1, 8]$	$[-50, 50]$	0.1, 0, 0.6	12	13.1	$z \in [3.9, 4.1]$	$Q_9 = 1$	1

(Time unit: second) Cyber-physical System Properties - δ: control stepsize, X_S: safe state set, U: control input limits, PID: PID parameters of original controller. Recovery-related Parameters - t_0: attack launched time, t_a: attack detected time, X_T: target state set, Q_i: state cost corresponding to the i^{th} state (other costs set to 1), R: control input cost.

Table 3. Attack Scenarios

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

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

$[-15, 15]$. We concern the first state, so the first diagonal element of Q is Q_i, others are all 1. We choose control input cost as 0.1.

6.1.2 Attack Scenarios. We consider three attack scenarios mentioned in Section 3.1, i.e., bias attacks, replay attacks and delay attacks. These attack scenarios are combined with each simulation scenario, which contributes to fifteen situations in total. We list the simulation parameters in Table 3. For each simulation scenario, we set up an attack parameter.

6.2 Baselines

We compare our proposed method with three baselines:

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

Non-real-time recovery: the system performs the recovery method in [33] after a sensor attack is detected. This method cannot guarantee the system is recovered before a deadline.

Linear-programming recovery: the system runs under a real-time recovery method in [59]. This method is formulated as an linear programming problem.

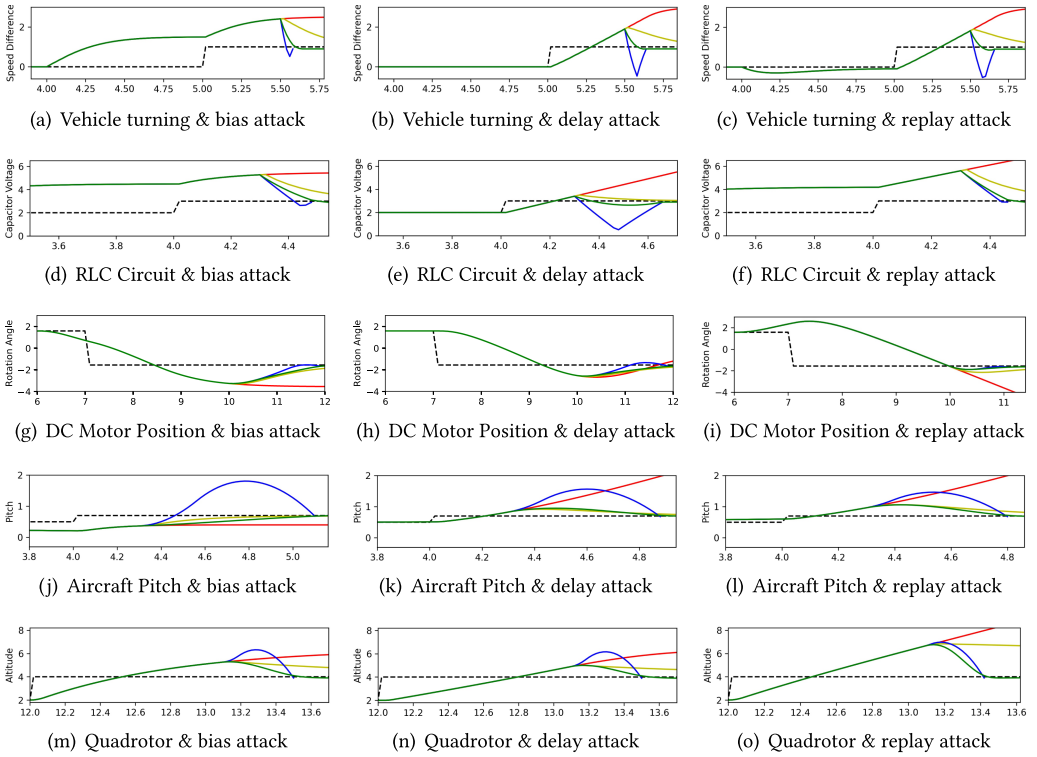
LQR-based recovery: the system run under the proposed real-time recovery method. This article formulates the recovery method as a LQR-based optimization problem.

6.3 Simulation Results

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

6.3.1 Recovery Effect. We plot the actual system states in Figure 5, which demonstrates the recovery effect against three types of sensor attacks in five CPS simulations. The following observations are obtained from these figures.

Recovery mechanism is needed during sensor attacks. The red lines represent the baseline without recovery. The results in Figure 5(f)(i)(k)(l)(o) shows the system states reach the unsafe set, which causes catastrophic consequences in CPS system. This indicates that a recovery mechanism is needed to secure the CPS in presence of sensor attacks.



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

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

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

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

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

Fig. 5. Comparison of the system executions under three situations for each attack scenario. RED = No recovery. YELLOW = Non-real-time recovery (previous work [33]). BLUE = Linear-Programming recovery (previous work [59]). GREEN = LQR-based recovery (our proposal). Dotted Black Line = Reference state.

Non-real-time recovery cannot steer system state back to normal within a deadline. The yellow lines show the state recovered using non-real-time recovery method. From Figure 5(a)(b)(c)(d)(f)(m)(n)(o), the recovery processes take a long time, and they failed to pull the system states back to target set before the deadline. There is no guarantee on deadline and safety.

LP recovery may make the system oscillate before the deadline. The blue lines are powered by LP recovery method. They can recover the system within deadline, because this method formulates the safety deadline as a constraint. However, the recovery trajectories are circuitous, shown in Figure 5(a)(b)(c)(e)(j)(k)(l)(m)(n), because of the linear objective function.

LQR-based recovery can recover systems smoothly within the deadline. The Green lines represent the recovery process of our LQR-based method. The proposed method can recover the system within the deadline, and the recovery trajectories are straightforward. The quadratic objective function adds penalty on states and control inputs, thus, play a key role in smooth trajectories. Moreover, our method makes the state maintain within the target set for a while, which is helpful to return the system to original controller or buy time to restart the original controller.

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

Table 4. Time Cost of Computing the Recovery Controls

	Vehicle turning			RLC Circuit			DC Motor Position		
	bias	delay	replay	bias	delay	replay	bias	delay	replay
T_{LP}	0.83	1.04	1.03	1.31	2.51	1.17	3.92	3.48	1.80
$\%_{LP}$	4.15%	5.20%	5.15%	6.55%	12.55%	5.85%	3.92%	3.48%	1.80%
T_{solver}	24.05	24.54	26.80	27.10	35.21	22.82	82.78	67.92	81.19
$\%_{solver}$	120.25%	122.70%	134.00%	135.50%	176.05%	114.10%	82.78%	67.92%	81.19%
T_{ADMM}	1.69	1.97	1.85	2.09	3.82	2.00	4.48	4.32	3.20
$\%_{ADMM}$	8.45%	9.85%	9.25%	10.45%	19.10%	10.00%	4.48%	4.32%	3.20%

	Aircraft Pitch			Quadrotor		
	bias	delay	replay	bias	delay	replay
T_{LP}	10.79	6.10	4.78	10.91	11.34	8.10
$\%_{LP}$	53.95%	30.50%	23.90%	54.55%	56.70%	40.50%
T_{solver}	81.10	62.02	73.42	44.15	57.09	80.28
$\%_{solver}$	405.50%	310.10%	367.10%	220.75%	285.45%	401.40%
T_{ADMM}	27.37	22.80	19.76	38.52	25.18	35.17
$\%_{ADMM}$	136.85%	114.00%	98.80%	192.60%	125.90%	175.85%

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

method to generate the control input signals. For LQR-based recovery, we first use CVXPY with ECOS solver [16], and then use OSQP solver [5], a standard implementation of ADMM algorithm, to accelerate the solving process. There are following observations from Table 4.

We considered the larger overhead of LQR-based recovery in the framework. The first two rows show the time cost of LP recovery method, and all recovery can be done in one control stepsize. This work assumes that the recovery sequence can be applied immediately after an attack is detected, but this may not be true for complex system. The middle two rows are the time cost of our LQR-based recovery without ADMM algorithm, and the cost is more than one stepsize for most scenarios. A larger overhead is because we use a quadratic objective function instead of a linear one. However, we considered a small computational time ($t_r - t_a$) to get the recovery sequence in our framework, shown as Figure 2, and apply our recovery control sequence at time t_r, which makes our method practical.

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

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

6.3.3 The Impact of Load Disturbance. We take the quadrotor simulator under bias attacks as an example to demonstrate the impact of load disturbance. In this experiment, the safe set is set to be $z \in [-1, 5.7]$, and other parameters remain the same as those in Table 2. In the Equation (1), we consider that the uncertainty $\boldsymbol{\nu}(t)$ includes both sensing noise and load disturbance, and is bounded by v_{max}. We change the load disturbance by varying v_{max} in this experiment. We perform multiple simulations with increasing v_{max}, and record the safe deadline and maintainable time in Table 5.

Table 5. The Impact of Load Disturbance on Quadrotor Simulator under Bias Attack

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

Legends: v_{max}: the maximum load disturbance in each control step, D: recovery length ($D = t_d - t_r$), N: total recovery control length ($N = t_l - t_r$), $T_{solving}$: the solving time of OSQP solver in millisecond.

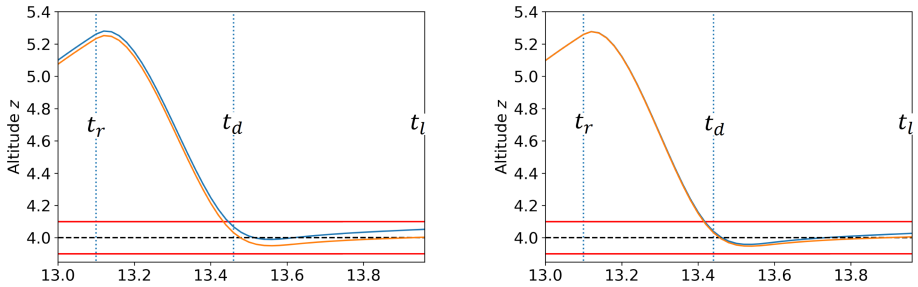
In the table, v_{max} denotes the maximum uncertainty/load disturbance that at each control step; D means the recovery length, i.e., the number of control steps from t_r to t_d; N denotes total recovery control length, i.e., the number of control steps from t_r to t_l; the safe deadline t_d and maintainable time t_l are estimated using our deadline estimator illustrated in the Section 5.3. We also plot the quadrotor's vertical position z in Figure 6. In the figure, the blue solid line is the real states; the orange solid line is the desired recovery states predicted by the recovery controller; the red solid line indicates the boundary of target state set. In the Figure 6(a), the load disturbance begins from time t_0, and in the Figure 6(b), the load disturbance begins from time t_r. Based on the results, we have several observations as follows.

Larger load disturbance leads to earlier safe deadline and earlier maintainable time. The load disturbance exists at each control step and is bounded by v_{max}. If v_{max} is larger, the reachable set overapproximation X_k is bigger according to the Equation (14). Then, the reachable set is more likely to intersect with the unsafe state set, which leads to a shorter recovery length N. Likewise, the constrained safe set with the accumulation of uncertainty, i.e., D_j, is smaller, which results in an earlier maintainable time t_l.

Larger total recovery control length N requires more computational overhead. The state constraints, i.e., Equation (3b) and Equation (3c), cover all states in the recovery and maintenance period. Thus, a larger N means more variables in the optimization problem, which comes with more computational overhead. When v_{max} is smaller,the system state can be maintained within the target set for a longer time once recovered into the set. We may choose a smaller maintenance length to reduce the computational overhead according to different application needs.

The system can still steer the system state back to the target state set and maintain it in the set in presence of load disturbance. In the Figure 6(a), the load disturbance starts from time t_0. At time t_r, the reconstructed state is slightly different from the real state because of the load disturbance. From t_d to t_l, the real state (marked in blue solid line) is within the target state set, although there exists difference between predicted states and the real states. This is because we considered the accumulation of load disturbance in our state constraint in the Equation (3c). In the Figure 6(b), the load disturbance starts from time t_r. At time t_r, the reconstructed state is almost the same as the real state. By comparing the two figures, we can see that the real state at time t_l is closer to the reference state in Figure 6(b) than the Figure 6(a). The reason is that the load disturbance in Figure 6(b) starts later than that in Figure 6(a), and thus affects the recovery for a shorter time. Further, as long as the load disturbance can be bounded to a certain range, the system can still be recovered by our method.

Our recovery method is sound but not complete. The right most column in Table 5 shows a case that our optimization problem is infeasible, when the total recovery control length $N = 16$ is less than the recovery length $D = 17$. This is because $X_T \ominus A^i \mathcal{I}_R$ in the Equation (3c) becomes empty because of the accumulation of load disturbance. Our method cannot guarantee finding a recovery



(a) load disturbance starts from t_0 (b) load disturbance starts from t_r

Fig. 6. The recovery of vertical position z of quadrotor under bias attacks with $v_{max} = 5 \times 10^{-4}$. The solid blue line represents real system states; the orange solid line shows the desired recovery states predicted by recovery controller; black dashed line is the reference or target states; red solid line marks the boundary of target state set.

control sequence to steer the system back to target state set under such a large load disturbance. Further, if our optimization is feasible, then the solution can guarantee that the recovery process will be successful.

7 CONCLUSION

Two fundamental elements for the operation of safe and resilient cyber-physical systems are attack detection and recovery. While the vast majority of existing works focus on attack detection, while little attention has been paid to attack-recovery. In this article, we study this problem and novel techniques on real-time recovery for securing CPS. These techniques include i) an LQR based recovery control calculator that can smoothly and safely recover the system before a safety deadline and maintain the recovered system for a certain amount of time, ii) a checkpointer that keeps enough trustworthy data for the recovery computation, iii) a state reconstructor that rebuilds the current system state, and iv) a deadline estimator that uses reachability analysis to conservatively compute a safety deadline. Using multiple CPS simulators, we show that our methods can recover the attacked-system in a timely and safe manner, outperforming previous related work in terms of smoothness and maintainability.

REFERENCES

[1] Francis Akowuah and Fanxin Kong. 2021. Physical invariant based attack detection for autonomous vehicles: Survey, vision, and challenges. In *4th International Conference on Connected and Autonomous Driving (MetroCAD)*. IEEE.

[2] Francis Akowuah and Fanxin Kong. 2021. Real-time adaptive sensor attack detection in autonomous cyber-physical systems. In *27th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS)*. IEEE.

[3] Francis Akowuah and Fanxin Kong. 2021. Recovery-by-learning: Restoring autonomous cyber-physical systems from sensor attacks. In *The 27th IEEE International Conference on Embedded and Real-time Computing Systems and Applications (RTCSA)*.

[4] Karl Johan Åström and Richard M Murray. 2010. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton university press.

[5] P. Goulart A. Bemporad B. Stellato, G. Banjac, and S. Boyd5. 2020. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation* 12 (2020), 637–672. https://doi.org/10.1007/s12532-020-00179-2

[6] Stephen Boyd, Neal Parikh, and Eric Chu. 2011. *Distributed Optimization and Statistical Learning Via the Alternating Direction Method of Multipliers*. Now Publishers Inc.

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

[8] Alvaro A. Cardenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. 2011. Attacks against process control systems: Risk assessment, detection, and response. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. 355–366.

[9] Alvaro A. Cardenas, Saurabh Amin, and Shankar Sastry. 2008. Secure control: Towards survivable cyber-physical systems. In *The 28th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 495–500.

[10] Somali Chaterji, Parinaz Naghizadeh, Muhammad Ashraful Alam, Saurabh Bagchi, Mung Chiang, David Corman, Brian Henz, Suman Jana, Na Li, Shaoshuai Mou, et al. 2019. Resilient cyberphysical systems and their application drivers: A technology roadmap. *Arxiv Preprint arXiv:2001.00090* (2019).

[11] Xin Chen and Sriram Sankaranarayanan. 2017. Model-predictive real-time monitoring of linear systems. In *IEEE Real-time Systems Symposium (RTSS)*. IEEE Press, 297–306.

[12] Hongjun Choi, Sayali Kate, Yousra Aafer, Xiangyu Zhang, and Dongyan Xu. 2020. Software-based realtime recovery from sensor attacks on robotic vehicles. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*. 349–364.

[13] Hongjun Choi, Wen-Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. 2018. Detecting attacks against robotic vehicles: A control invariant approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 801–816.

[14] Anežka Chovancová, Tomáš Fico, L'uboš Chovanec, and Peter Hubinsk. 2014. Mathematical modelling and parameter identification of quadrotor (a survey). *Procedia Engineering* 96 (2014), 172–181.

[15] Tanya L. Crenshaw, Elsa Gunter, Craig L. Robinson, Lui Sha, and P. R. Kumar. 2007. The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures. In *28th IEEE International Real-time Systems Symposium (RTSS)*. IEEE, 400–412.

[16] A. Domahidi, E. Chu, and S. Boyd. 2013. ECOS: an SOCP solver for embedded systems. In *European Control Conference (ECC)*. 3071–3076.

[17] Elmootazbellah Nabil Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. 2002. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)* 34, 3 (2002), 375–408.

[18] Hamza Fawzi, Paulo Tabuada, and Suhas Diggavi. 2014. Secure estimation and control for cyber-physical systems under adversarial attacks. *IEEE Transactions on Automatic Control* 59, 6 (2014), 1454–1467.

[19] Fan Fei, Zhan Tu, Dongyan Xu, and Xinyan Deng. 2020. 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, 7358–7364.

[20] Arun Ganesan, Jayanthi Rao, and Kang Shin. 2017. *Exploiting Consistency Among Heterogeneous Sensors for Vehicle Anomaly Detection*. Technical Report. SAE Technical Paper.

[21] H. Gao, X. Meng, and T. Chen. 2008. Stabilization of networked control systems with a new delay characterization. *IEEE Trans. Automat. Control* 53, 9 (2008), 2142–2148. https://doi.org/10.1109/TAC.2008.930190

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

[23] Jairo Giraldo, Esha Sarkar, Alvaro A Cardenas, Michail Maniatakos, and Murat Kantarcioglu. 2017. Security and privacy in cyber-physical systems: A survey of surveys. *IEEE Design & Test* 34, 4 (2017), 7–17.

[24] Jairo Giraldo, David Urbina, Alvaro Cardenas, Junia Valente, Mustafa Faisal, Justin Ruths, Nils Ole Tippenhauer, Henrik Sandberg, and Richard Candell. 2018. A survey of physics-based attack detection in cyber-physical systems. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–36.

[25] Michael Green and David JN Limebeer. 2012. *Linear Robust Control*. Courier Corporation.

[26] Bingsheng He and Xiaoming Yuan. 2012. On the $O(1/n)$ Convergence Rate of the Douglas–Rachford Alternating Direction Method. *SIAM J. Numer. Anal.* 50, 2 (2012), 700–709. https://doi.org/10.1137/110836936 arXiv:https://doi.org/10.1137/110836936

[27] Gaoqi He, Zhifu Chai, Xingjian Lu, Fanxin Kong, and Bing Sheng. 2019. ADMM-Based decentralized electric vehicle charging with trip duration limits. In *2019 IEEE Real-time Systems Symposium (RTSS)*. IEEE, 107–119.

[28] Tianjia He, Lin Zhang, Fanxin Kong, and Asif Salekin. 2020. Exploring inherent sensor redundancy for automotive anomaly detection. In *57th Design Automation Conference*. ACM.

[29] WPMH Heemels, JH Sandee, and PPJ Van Den Bosch. 2008. Analysis of event-driven controllers for linear systems. *International Journal of Control* 81, 4 (2008), 571–590.

[30] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. 2017. Cyber-physical systems security—a survey. *IEEE Internet of Things Journal* 4, 6 (2017), 1802–1831.

[31] Inseok Hwang, Sungwan Kim, Youdan Kim, and Chze Eng Seah. 2009. A survey of fault detection, isolation, and reconfiguration methods. *IEEE Transactions on Control Systems Technology* 18, 3 (2009), 636–653.

[32] Radoslav Ivanov, Miroslav Pajic, and Insup Lee. 2016. Attack-resilient sensor fusion for safety-critical cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)* 15, 1 (2016), 21.

[33] Fanxin Kong, Meng Xu, James Weimer, Oleg Sokolsky, and Insup Lee. 2018. Cyber-physical system checkpointing and recovery. In *2018 ACM/IEEE 9th International Conference on Cyber-physical Systems (ICCPS)*. IEEE, 22–31.
[34] Huibert Kwakernaak and Raphael Sivan. 1972. *Linear Optimal Control Systems*. Vol. 1. Wiley-interscience New York.
[35] Yao Liu, Peng Ning, and Michael K Reiter. 2011. False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security (TISSEC)* 14, 1 (2011), 1–33.
[36] Bei Lu et al. 2005. Linear parameter-varying control of an F-16 aircraft at high angle of attack. (2005).
[37] Pengyuan Lu, Limin Zhang, B. Brian Park, and Lu Feng. 2018. Attack-resilient sensor fusion for cooperative adaptive cruise control. In *21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 3955–3960.
[38] Rui Ma, Sagnik Basumallik, Sara Eftekharnejad, and Fanxin Kong. 2020. Recovery-based model predictive control for cascade mitigation under cyber-physical attacks. In *2020 IEEE Texas Power and Energy Conference (TPEC)*. IEEE, 1–6.
[39] Rui Ma, Sagnik Basumallik, Sara Eftekharnejad, and Fanxin Kong. 2021. A data-driven model predictive control for alleviating thermal overloads in presence of possible false data. *IEEE Transactions on Industry Applications* (2021).
[40] Robert Mitchell and Ing-Ray Chen. 2014. A survey of intrusion detection techniques for cyber-physical systems. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 1–29.
[41] Yilin Mo, Emanuele Garone, Alessandro Casavola, and Bruno Sinopoli. 2010. False data injection attacks against state estimation in wireless sensor networks. In *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 5967–5972.
[42] Sibin Mohan, Stanley Bak, Emiliano Betti, Heechul Yun, Lui Sha, and Marco Caccamo. 2012. S3a: Secure system simplex architecture for enhanced security of cyber-physical systems. *Arxiv Preprint arXiv:1202.5722* (2012).
[43] Miroslav Pajic, James Weimer, Nicola Bezzo, Oleg Sokolsky, George J. Pappas, and Insup Lee. 2017. Design and implementation of attack-resilient cyberphysical systems: With a focus on attack-resilient state estimators. *IEEE Control Systems Magazine* 37, 2 (2017), 66–81.
[44] Miroslav Pajic, James Weimer, Nicola Bezzo, Paulo Tabuada, Oleg Sokolsky, Insup Lee, and George J. Pappas. 2014. Robustness of attack-resilient state estimators. In *ACM/IEEE 5th International Conference on Cyber-physical Systems (ICCPS)*. IEEE Computer Society, 163–174.
[45] Jonathan Petit, Bas Stottelaar, Michael Feiri, and Frank Kargl. 2015. Remote attacks on automated vehicles sensors: Experiments on camera and lidar. *Black Hat Europe* 11 (2015), 2015.
[46] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. 2020. SAVIOR: Securing autonomous vehicles with robust physical invariants. In *29th USENIX Security Symposium (USENIX Security 20)*.
[47] Aviva Hope Rutkin. 2013. spoofers use fake GPS signals to knock a yacht off course. MIT Technology Review. Online; accessed May 2020.
[48] Francesco Sabatino. 2015. Quadrotor control: modeling, nonlinear control design, and simulation.
[49] F. Sabatino. 2015. *Quadrotor control: modeling, nonlinear control design, and simulation*. Master's thesis. KTH Royal Institute of Technology.
[50] JH Sandee, WPMH Heemels, and PPJ Van Den Bosch. 2005. Event-driven control as an opportunity in the multidisciplinary development of embedded controllers. In *American Control Conference*. IEEE, 1776–1781.
[51] Yasser Shoukry, Paul Martin, Paulo Tabuada, and Mani Srivastava. 2013. Non-invasive spoofing attacks for anti-lock braking systems. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 55–72.
[52] K. C. Tan and Y. Li. 2001. Performance-based control system design automation via evolutionary computing. *Engineering Applications of Artificial Intelligence* 14, 4 (2001), 473–486.
[53] David I. Urbina, Jairo A. Giraldo, Alvaro A. Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg. 2016. Limiting the impact of stealthy attacks on industrial control systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1092–1105.
[54] Ruixuan Wang, Fanxin Kong, Hasshi Sudler, and Xun Jiao. 2021. HDAD: Hyperdimensional computing-based anomaly detection for automotive sensor attacks. In *27th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS), Brief Industry Paper Track*. IEEE.
[55] Xiaofeng Wang, Naira Hovakimyan, and Lui Sha. 2013. L1Simplex: Fault-tolerant control of cyber-physical systems. In *2013 ACM/IEEE International Conference on Cyber-physical Systems (ICCPS)*. IEEE, 41–50.
[56] Greg Welch, Gary Bishop, et al. 1995. An introduction to the kalman filter. (1995).
[57] Marilyn Wolf and Dimitrios Serpanos. 2017. Safety and security in cyber-physical systems and internet-of-things systems. *Proc. IEEE* 106, 1 (2017), 9–20.
[58] Chen Yan, Hocheol Shin, Connor Bolton, Wenyuan Xu, Yongdae Kim, and Kevin Fu. 2020. Sok: A minimalist approach to formalizing analog sensor security. In *2020 IEEE Symposium on Security and Privacy (SP)*. 480–495.
[59] Lin Zhang, Xin Chen, Fanxin Kong, and Alvaro A. Cardenas. 2020. Real-time recovery for cyber-physical systems using linear approximations. In *41st IEEE Real-time Systems Symposium (RTSS)*. IEEE.
[60] Kemin Zhou and John Comstock Doyle. 1998. *Essentials of Robust Control*. Vol. 104. Prentice hall Upper Saddle River, NJ.

Received April 2021; revised June 2021; accepted July 2021