Cyber-Physical System Checkpointing and Recovery

Fanxin Kong, Meng Xu, James Weimer, Oleg Sokolsky, Insup Lee
Department of Computer & Information Science, University of Pennsylvania
{*fanxink, mengxu, weimerj, sokolsky, lee*}*@cis.upenn.edu*

***Abstract*—Transitioning to more open architectures has been making Cyber-Physical Systems (CPS) vulnerable to malicious attacks that are beyond the conventional cyber attacks. This paper studies attack-resilience enhancement for a system under emerging attacks in the environment of the controller. An effective way to address this problem is to make system state estimation accurate enough for control regardless of the compromised components. This work follows this way and develops a procedure named CPS checkpointing and recovery, which leverages historical data to recover failed system states. Specially, we first propose a new concept of physical-state recovery. The essential operation is defined as rolling the system forward starting from a consistent historical system state. Second, we design a checkpointing protocol that defines how to record system states for the recovery. The protocol introduces a sliding window that accommodates attack-detection delay to improve the correctness of stored states. Third, we present a use case of CPS checkpointing and recovery that deals with compromised sensor measurements. At last, we evaluate our design through conducting simulator-based experiments and illustrating the use of our design with an unmanned vehicle case study.**

***Keywords*-Cyber-Physical Systems, Security, Resilience, Checkpointing, Recovery**

I. Introduction

Cyber-Physical Systems (CPS) tightly couple computing and communication processes with sensing and actuation components that interact with the physical world. A typical example for such systems is modern vehicles, which demonstrate a complex interaction of many Electric Control Units (ECUs) over different types of networks. The increasing functionalities and network interoperability transition CPS from isolated control systems to more open interacting architectures, which enables various new services and applications such as remote code updates and vehicle-to-vehicle communication. Meanwhile, this transition, however, introduces potential security vulnerabilities that are easily exploitable [1]–[3].

The interaction between information technology and the physical world makes CPS vulnerable to malicious attacks that are beyond the traditional cyber attacks [4]–[6]. For example, the authors in [7] point out that using simple methods can disrupt the operation of a car and even disable the vehicle. Further, the authors in [8] illustrate a case study, where a Denial-of-Service (DoS) attack can be launched to compromise the CAN bus and the functionalities dependent on the bus. This issue is even more emphasized with the rise of vehicle autonomy, because the autonomy will exacerbate the consequences of attacks. Thus, we argue that security is one key prerequisite for large deployment of autonomous systems such as self-driving cars.

Exclusively employing cyber-security techniques to secure CPS is inadequate. This is indicated especially by non-invasive sensor attacks, that is, when the physical environment is compromised to allow injecting malicious signals to sensors [6]. For example, the authors in [9] exploit weaknesses in wheel speed sensors and present non-invasive attacks on Antilock Braking Systems (ABS). The authors in [10] demonstrate attacks on GPS sensors to misguide a yacht off course. In addition, the authors in [11] demonstrate remote attacks on sensors including camera and LiDAR that are usually mounted in autonomous vehicles.

These results have motivated many research efforts that study the problems of intrusion detection (e.g., [12]–[15]) and attack-resilience (e.g., [6], [16]–[18]) under the cases of various attacks on sensors, actuators and communication networks. An effective way for improving attack-resilience is to develop methods that can estimate system states accurate enough for control regardless of the compromised components. One advantage of this way is that it allows a system to use the same controller as in the case without attacks. Existing work such as [6], [16] right follows this way. However, the proposed techniques confine to the setting with sensor redundancy, i.e., multiple sensors (partially) measuring the same physical variables. Further, the techniques are applicable only when the number of compromised sensors is within a threshold. As a result, it is still in question how to deal with attack-resilience for the case that violates these above limitations.

To address this case, we propose to leverage checkpointing and recovery, i.e., using historical data to recover system states. However, to design such a procedure well applicable to CPS is a non-trivial task because of the following two major challenges. Firstly, the conventional roll-back operation is unsuitable for recovering states of the plant. Physically rolling back the plant usually incurs considerable overhead, and further it is even infeasible to roll back some irreversible processes. Secondly, detection mechanisms, such as data-driven methods for sensor attacks [12], [19], usually have substantial detection delay, i.e., the time interval between the occurrence of an attack and the detection of it. States stored during the detection interval may be incorrect and

thus using these states can result in unsuccessful recoveries.

In this paper, we develop a procedure of CPS recovery and checkpointing that well addresses both challenges. Firstly, we propose a new concept of physical-state recovery. The essential operation is defined as rolling the system forward to the current time, starting from a consistent historical physical-state. This roll-forward operation has much lower overhead than the conventional roll-back operation in the context of recovering physical-states. Secondly, we design a checkpointing protocol that defines how to record system states used for the recovery. The protocol employs a sliding window that accommodates the detection delay to improve the correctness of the stored states. Note that the CPS checkpointing and recovery is an reactive procedure that needs to be used with existing detection mechanisms (e.g., [19], [20]), that is, the physical-state recovery is triggered to execute if the mechanisms discover some compromised components; otherwise, the system follows the checkpointing protocol to record states.

In essence, the CPS checkpointing and recovery is a general method that handles failed estimated states. Such states can be created by different kinds of possibilities and favorably, our method is not confined to a certain kind. To be specific, our method is applicable to deal with compromised sensor measurements caused by attacks and faults. In this sense, attack and fault are treated interchangeably in this paper. We then present a use case of the procedure on addressing sensor attacks/faults, e.g., attackers modifying sensor information or preventing the controller from receiving it. Finally, we evaluate our design through conducting simulation-based experiments as well as illustrate the use of our design on an unmanned vehicle case study.

The rest of the paper is organized as follows. Section II defines CPS recovery. Section III presents the checkpointing protocol. Section IV uses CPS recovery and checkpointing to address sensor attacks/faults. Section V validates our design. Section VII concludes the paper.

II. Cyber-Physical System Recovery

Conceptually, CPS recovery can be seen as to restore the controllability and functionality of the system. We study CPS recovery through dividing it into two different operations: recovery of cyber-states and recovery of physical-states. *Cyber-states* are defined as computing information of a controller, such as values of data variables. *Physical-states* are defined as physical information of a plant, such as the velocity of a motor.

A. Cyber-State Recovery

Recovery of cyber-states, or cyber-state recovery, is similar to that of conventional computer systems. Thus, we review only several key concepts and definitions from [21] and adapt them to cyber-physical systems.

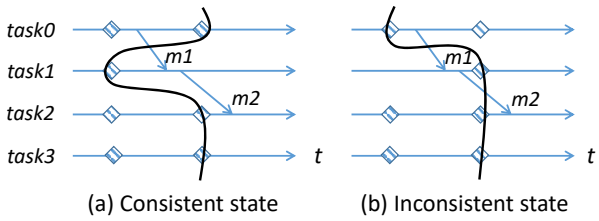


Figure 1. An example of consistent and inconsistent cyber-state.

Definition 1: [Consistent Global Cyber-State] A consistent global cyber-state is one in which, if the state of a task reflects a message receipt, then the state of the corresponding sender task reflects sending that message.

Definition 2: [Cyber-State Recovery] Cyber-state recovery is defined as rolling the cyber component (i.e., the controller) *back* to a consistent global cyber-state.

Fig. 1 depicts an example that illustrates the above two definitions, where the cyber component, i.e., the controller, consisting of four tasks. Fig. 1(a) shows a global consistent cyber-state (marked by the black line), because all the four states satisfy Def. 1. By contrast, Fig. 1(b) is an inconsistent state, because $task1$ is shown to have received message $m1$ but the state of $task0$ does not reflect sending it.

Tasks $task0$, $task1$, and $task2$ are called to be *dependent* on each other due to their mutual communication; while $task3$ is independent on other tasks because it has no communication with them. For an independent task, e.g., $task3$, any of its previously stored correct cyber-states can be used for recovery, and the latest one is usually used for the global consistent state in order to shorten the recovery time as much as possible. The cyber-state recovery can be that the cyber component rolls back to the global consistent state in Fig. 1(a) if some failure occurs. With stored message $m1$, $task0$ does not need to resend it and thus $task1$ can then start to execute from that state.

Another key concept is *roll-back propagation*, which is a phenomenon that upon a failure of one or more tasks, their dependencies may force some of the tasks that do not fail to roll back. Under some situations, rollback propagation may extend back to the initial state, losing all the work executed before the failure.

The above just introduces some key background for cyber-state recovery. There are many existing works that study cyber-state recovery and validate its feasibility [21]. Thus, in the following, we will focus on discussing our newly proposed physical-state recovery.

B. Physical-State Recovery

Unlike CPS, conventional computer systems usually do not have much interaction with the physical space. Thus no clear definition of physical-state recovery has been studied for them. We identify and define physical-state recovery for CPS in this subsection.

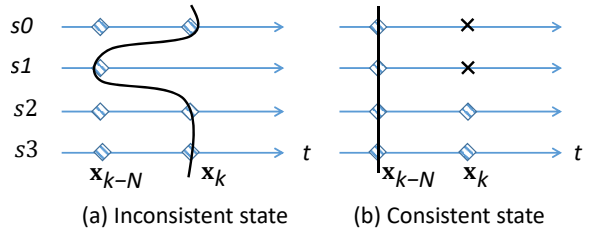


Figure 2. An example of consistent and inconsistent physical-state. ×: failure.

We consider a Linear-Time Invariant (LTI) system given by Eqn. (1)(2), as an example to present physical-state recovery.

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{v}_k, \tag{1}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{w}_k, \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^n$ denotes the plant's physical-state vectors; $\mathbf{u} \in \mathbb{R}^m$ is the control input vectors; $\mathbf{y} \in \mathbb{R}^p$ denotes the plant's output vectors from measurements of sensors; $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^p$ are noise vectors; $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ have suitable dimensions.

Definition 3: [Consistent Global Physical-State] A consistent global physical-state is one that reflects each individual internal element's value of the same time point.

Fig. 2 depicts an example that illustrates the above definition, where $s0$ to $s3$ are four internal elements of the system state. Fig. 2(a) presents an inconsistent physical-state (marked by the black line), because the value of element $s1$ corresponds to a different time point from that of other three elements. By contrast, Fig. 2(b) shows a global consistent physical-state, because the values of all four elements are of the same time t_{k-N}.

The rationale of defining the physical-state consistency based on time is two-fold. First, in real systems, clocks are synchronized within some very small fixed time interval, and we acknowledge that clock synchronization is non-negligible, in general. However, to simplify presentation, here we assume clock synchronization error is small and omit clock synchronization uncertainty in the analysis. Second, internal elements of a system state change separately as to the output of the system.

Definition 4: [Physical-State Recovery] Physical-state recovery is defined as rolling values of internal elements *forward* to the current time, by starting from a consistent global physical-state.

We use $\mathbf{x}_{k,(1,q)} \in \mathbb{R}^q$ to denote the physical-state vector of q failed elements, i.e., $\mathbf{x}_{k,(1,q)} = [x_{k,1}, ..., x_{k,q}]^T$, and use $\mathbf{x}_{k,(q+1,n)} \in \mathbb{R}^{n-q}$ to denote the vector of $n-q$ unfailed elements, i.e., $\mathbf{x}_{k,(q+1,n)} = [x_{k,q+1}, ..., x_{k,n}]^T$. For example, as shown in Fig. 2(b), elements $s0$ and $s1$ fail at current time t_k, i.e., $\mathbf{x}_{k,(1,2)} = [x_{k,1}, x_{k,2}]^T$, while elements $s2$ and $s3$ operate correctly, i.e., $\mathbf{x}_{k,(3,4)} = [x_{k,3}, x_{k,4}]^T$. We consider elements dependent on attacked/faulty sensors as failed. By Def. 4, the physical-state recovery is to perform the *roll-forward state prediction* for q failed elements using the following two steps.

Step (i): use the control inputs between t_{k-N} and t_k to make a prediction about the current system state (of all elements), that is,

$$\hat{\mathbf{x}}_k = \mathbf{A}^N \bar{\mathbf{x}}_{k-N} + \sum_{i=1}^{N} \mathbf{A}^{i-1}\mathbf{B}\mathbf{u}_{k-i}, \tag{3}$$

where $\bar{\mathbf{x}}_{k-N}$ is a consistent global physical-state estimated using $\mathbf{y}_{k-N} = \mathbf{C}\bar{\mathbf{x}}_{k-N}$. We assume that the system is observable, i.e., full states can be estimated by some way.

Step (ii): for the q failed elements, use the value vector extracted from $\hat{\mathbf{x}}_k$ (given by Eqn. (3)), that is, $\hat{\mathbf{x}}_{k,(1,q)}$. This means that this part of state prediction is based on the historical information.

For the $n-q$ unfailed elements, we use the value vector estimated by $\mathbf{y}_k = \mathbf{C}\bar{\mathbf{x}}_k$, which is $\bar{\mathbf{x}}_{k,(q+1,n)}$. This means that this part of state estimation is based on current sensor measurements. Hence, the overall estimated state $\tilde{\mathbf{x}}_k$ after the physical-state recovery is

$$\tilde{\mathbf{x}}_k = \begin{bmatrix} \hat{\mathbf{x}}_{k,(1,q)} \\ \bar{\mathbf{x}}_{k,(q+1,n)} \end{bmatrix}. \tag{4}$$

There is no *roll-forward propagation* phenomenon for the defined physical-state recovery, which means that failed elements will not cause unfailed elements to roll forward. The reason is two-fold. Failed elements depend on which sensors are compromised, instead of depending on other elements. Further, the internal elements of a system state change separately, and the failed and unfailed elements are also treated separately by the defined physical-state recovery.

Rationale of Roll-Forward Recovery. There can be two different ways to perform physical-state recovery: one is rolling the system forward from the consistent global state while the other is rolling the plant back to that state. To realize the latter way, it needs to make the plant's state match the values of the consistent state, which requires physically rolls the plant back. This operation not only comes with high overhead but also sometimes is infeasible, e.g., for irreversible processes. By contrast, the roll-forward recovery is carried out from the other direction, i.e., matching the values to the plant's state, which is thus always feasible and with lower overhead. Therefore, we choose to use the roll-forward operation for the physical-state recovery.

C. Cyber-State vs. Physical-State Recovery

Table I presents a comparison about key features of cyber-state recovery and physical-state recovery.

Firstly, cyber-state recovery defines the state consistency based on logic to ensure the computational correctness of tasks of a controller; while the physical-state recovery defines the consistency based on time to ensure that values of internal elements reflect the plant's state of the same time point.

Table I
MAJOR DIFFERENCES BETWEEN CYBER-STATE AND PHYSICAL-STATE RECOVERY

	Consistency	Direction	Propagation	Applicability
Cyber-state recovery	Logic-based	Roll-back	Yes	Within one sampling period
Physical-state recovery	Time-based	Roll-forward	No	Across sampling periods

Secondly, cyber-state recovery is rolling the cyber-component or the controller back to a consistent global state and the roll-back may propagate among tasks because of their dependence; while the physical-state recovery is rolling internal elements' values forward to the current time and the roll-forward does not propagate among internal elements.

Thirdly, cyber-state recovery confines to the cyber-states within one single sampling period; while the physi-cal-state recovery has a scope of physical-states across multiple sampling periods. The execution of tasks in one sampling period is usually separate from their execution in other periods, and thus the cyber-states are only valid and useful within one single sampling period. By contrast, the physical-states (i.e., values of internal elements) do not change while executing control tasks within one period, and instead they may change across sampling periods. Hence, for a case that needs both recoveries, a system first performs cyber-state recovery and then carries out physical-state recovery.

D. State Prediction and Estimation Errors

The overall error $\mathbf{e}_k$ caused by the physical-state recovery is composed of two parts: (i) the state prediction error $\mathbf{e}_{k,(1,q)}$ of the q failed elements, and (ii) the state estimation error $\mathbf{e}_{k,(q+1,n)}$ of the $n-q$ unfailed elements. That is,

$$\mathbf{e}_k = \left[\begin{array}{c} \mathbf{e}_{k,(1,q)} \\ \mathbf{e}_{k,(q+1,n)} \end{array} \right]. \tag{5}$$

(i) The prediction error $\mathbf{e}_{k,(1,q)}$ is caused by the physical-state recovery, which is thus

$$\begin{aligned} \mathbf{e}_{k,(1,q)} &= [\mathbf{x}_k - \hat{\mathbf{x}}_k]_{(1,q)} \\ &= \left[\mathbf{A}^N \delta_{k-N}\right]_{(1,q)} + \left[\sum_{i=1}^{N} \mathbf{A}^{i-1}\mathbf{v}_{k-i}\right]_{(1,q)}, \end{aligned} \tag{6}$$

where $\delta_{k-N} = \mathbf{x}_{k-N} - \bar{\mathbf{x}}_{k-N}$ can be obtained by solving the following equation set:

$$\left\{ \begin{array}{l} \mathbf{y}_{k-N} = \mathbf{C}\mathbf{x}_{k-N} + \mathbf{w}_{k-N} \\ \mathbf{y}_{k-N} = \mathbf{C}\bar{\mathbf{x}}_{k-N} \end{array} \right. . \tag{7}$$

(ii) The estimation $\mathbf{e}_{k,(q+1,n)}$ is induced by sensor measurements, which is thus

$$\mathbf{e}_{k,(q+1,n)} = \delta_{k,(q+1,n)}, \tag{8}$$

where $\delta_k = \mathbf{x}_k - \bar{\mathbf{x}}_k$ can be obtained by a similar way as in Eqn. (7).

We use $|\mathbf{P}|$ to denote the matrix whose elements are absolute values of the initial matrix $\mathbf{P}$. Further, for matrices $\mathbf{P}$ and $\mathbf{Q}$, $\mathbf{P} \preceq \mathbf{Q}$ means that the matrix $\mathbf{P}$ is element-wise less than or equal to the matrix $\mathbf{Q}$. If we assume that $|\delta_k| \preceq \epsilon_{\delta_k}$ and $|\mathbf{v}_k| \preceq \epsilon_{v_k}$, then we can bound the state prediction and estimation errors as

$$\begin{aligned} &\mathbf{e}_{k,(1,q)} \preceq \mathbf{e}^+_{k,(1,q)} = \left[|\mathbf{A}^N| \epsilon_{\delta_{k-N}}\right]_{(1,q)} + \left[\sum_{i=1}^{N} |\mathbf{A}^{i-1}| \epsilon_{v_{k-i}}\right]_{(1,q)}, \\ &\mathbf{e}_{k,(q+1,n)} \preceq \mathbf{e}^+_{k,(q+1,n)} = \epsilon_{\delta_{k,(q+1,n)}}. \end{aligned} \tag{9}$$

From the above equation, we can see that the bound $\mathbf{e}^+_{k,(1,q)}$ increases as the time interval (t_{k-N}, t_k) or the number N grows; while the bound $\mathbf{e}^+_{k,(q+1,n)}$ only depends on the information at current time t_k. Later on, we will discuss more about the state prediction error in the evaluation section. Note that in the above, we just show an example of how the physical-state recovery works, where the system model is used to carry out state prediction. The prediction can be also preformed by other methods, such as Kalman filter or machine learning. Comparison between these methods is one future work.

III. CHECKPOINTING CYBER-PHYSICAL SYSTEMS

We have introduced the physical-state recovery. This section discusses how to checkpoint a system, i.e., how to record system states that can be used for the recovery.

A. A Sliding Window Based Checkpointing Protocol

Checkpointing a system means occasionally storing the state of the system on safe and secure devices. Note that states in this section include both physical-state information (i.e., estimated physical-states from sensor measurements) and control inputs. The stored states are called checkpoints or recovery points. If no failure has occurred, a checkpoint is regarded as correct and thus a trustworthy state. The checkpoint will be then stored and used for potential recoveries in future. The protocol stipulates that detection is carried out before saving system states. This stipulation will improve the correctness of checkpoints and thus improves the probability of successful recovery.

Protocol Operation. Fig. 3 illustrates the timing diagram of the proposed protocol. The protocol has four major components as follows.

(i) *Detection window*. The window size (of N sampling periods) represents how many historical data are used for detection, for example, N latest sensor measurements used to detect sensor attacks/faults.

(ii) *Buffered states* (states from t_{k-N+1} to t_k). The states within the detection window are first buffered, i.e., pending

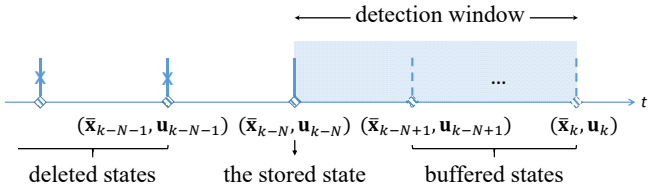


Figure 3. Timing diagram for checkpointing cyber-physical systems.

to be stored, because the detection has not yet given results for them.

(iii) *The stored state/checkpoint* (the state at t_{k-N}). The checkpoint already passes the detection and is stored for potential recoveries.

(iv) *Deleted states* (states from t_0 to t_{k-N-1}). When the most recent correct checkpoint is enough for recovery, the previous stored states or checkpoints are no longer needed and thus can be discarded.

First, the protocol checkpoints the system at every sampling period. To ensure the consistency of physical-states, each checkpoint contains values of internal elements of the same period. As illustrated in Fig. 3, at current time t_k, the protocol first buffers the state $(\mathbf{x}_k, \mathbf{u}_k)$. The reason is that within the detection window, there is no detection result for the state, and whether it is correct is still unknown yet. Consider sensor attacks as an example. Many existing works, such as data-driven [12], [19], [22] and model-based [15], [23] ones, use historical sensor measurements from multiple sampling periods to perform attack detection. Upon the detection of an attack, it is hard for these works to tell the attack occurs at which time point within the detection window, especially for attacks that modify information in an unaggressive way. Note that the window size N, i.e., the number of historical data used for detection, is determined by a specific detection mechanism. For example, the authors in [19] use a window size of 100 seconds for their correlation-based detection mechanism, and thus N is the number of data points within 100 seconds. The time here is also called as detection delay, and N is used to represent this delay.

Second, the buffered states that have moved outside the detection window, are considered to have successfully passed the detection and thus are regarded as correct states if no failure (e.g., failures caused by compromised sensors) is detected so far. As shown in Fig. 3, the checkpoint at time t_{k-N} is outside the detection window, and thus state $(\mathbf{x}_{k-N}, \mathbf{u}_{k-N})$ is saved. Storing after buffering can eliminate the case of saving states between the occurrence of an attack and its detection as well as the corresponding latent failures caused by this case.

Thirdly, the protocol discards those checkpoints that are no longer needed. As shown in Fig. 3, the checkpoint at time t_{k-N-1} is discarded because the checkpoint at time t_{k-N} is newly stored and regarded as correct.

Introducing the detection window with a size of multiple sampling periods makes the proposed checkpointing protocol different from many existing works such as [24], [25], which assume instant detection upon the failures' occurrence.

IV. A Use Case: Addressing Sensor Attacks/Faults

In this section, we present a use case of the CPS checkpointing and recovery, which is the capability of addressing sensor attacks/faults. We assume the existence of some mechanisms that can detect sensor attacks/faults. Our method works with those mechanisms, that is, after they discover some attacks/faults, the physical-state recovery will be triggered to execute.

The threat model is as follows. First, we assume that attackers are able to compromise sensor information, e.g., modifying it or preventing the controller from receiving it, but they cannot compromise actuators. Second, the historical information, including physical-states and control inputs, is securely stored and cannot be compromised by attackers.

A. Handling Compromised Sensor Measurements

Attacks on sensors make them unreliable and thus the estimated plant's states based on their measurements also become untrustworthy. Suppose that at current time t_k, some sensor attacks are detected, which cause values of q internal elements compromised, i.e., there are q failed internal elements. The following two sequential steps are applied to handle the attacks.

Step 1: Physical-State Recovery. The physical-state recovery uses Eqn. (4) to predict and estimate the current system state. After the recovery, the first key question is whether the estimated state $\tilde{\mathbf{x}}_k$ is accurate enough to be used for control. If $\tilde{\mathbf{x}}_k$ is far from the plant's realistic state, the derived control inputs may drive the plant to drift even further away. Let $\mathbf{E}$ be the vector that denotes the maximum state estimation error that the system can tolerate. Given Eqn. (9), if the estimation error satisfies

$$\mathbf{e}_k^+ = \begin{bmatrix} \mathbf{e}_{k,(1,q)}^+ \\ \mathbf{e}_{k,(q+1,n)}^+ \end{bmatrix} \preceq \mathbf{E}, \tag{10}$$

then the estimated state $\tilde{\mathbf{x}}_k$ is regarded acceptable and thus can be used for control. Note that till now, the system only carries out the physical-state recovery, which thus only derives $\tilde{\mathbf{x}}_k$. No new control inputs for actuation have been generated yet.

Step 2: Recovery-Based Resilient Control. With an acceptable estimated state $\tilde{\mathbf{x}}_k$, this step is to control the plant in presence of compromised sensor measurements, where new control inputs will be generated for actuation. Let $t_{k+M}, M \geq 1$ denote the time after Mth sampling period from the current time t_k. Between time t_k and t_{k+M}, based

on the recovered state $\tilde{\mathbf{x}}_k$, we use Eqn. (11) to predict values of the q failed internal elements, which is

$$\hat{\mathbf{x}}_{k+M,(1,q)} = \left[\mathbf{A}^M \tilde{\mathbf{x}}_k + \sum_{i=1}^{M} \mathbf{A}^{i-1} \mathbf{B} \mathbf{u}_{k+M-i} \right]_{(1,q)}. \quad (11)$$

For the $n-q$ unfailed elements, use the value vector estimated by $\mathbf{y}_{k+M} = \mathbf{C}\bar{\mathbf{x}}_{k+M}$ at time t_{k+M}. Thus, at time t_{k+M}, the overall estimated state $\tilde{\mathbf{x}}_{k+M}$ is

$$\tilde{\mathbf{x}}_{k+M} = \begin{bmatrix} \hat{\mathbf{x}}_{k+M,(1,q)} \\ \bar{\mathbf{x}}_{k+M,(q+1,n)} \end{bmatrix}. \quad (12)$$

Note that the proposed method, i.e., the physical-state recovery combined with the resilient control, is triggered to execute after some sensor attacks or faults are detected. In other words, the method is an reactive mechanism that needs to work with some existing detection mechanisms such as [12], [19].

Conceptual Discussion. The defined recovery-based resilient control has some similarities and differences with two established control concepts: open-loop control and closed-loop control. (i) Compared with open-loop control, one similarity is that both methods do not use measurements of compromised sensors as feedback to a controller. One difference is that the defined resilient control uses the predicted state (based on the physical-state recovery) and measurements of uncompromised sensors as feedback, while open-loop control does not. (ii) Compared to closed-loop control, both methods have feedback, but the content of feedback is different. That is, one is based on sensor measurements, while the other one is the predicted state of failed internal elements as well as using measurements of uncompromised sensors.

B. Drift Analysis

Between time t_k and t_{k+M}, the system may continue drifting away, since the control relies on the predicted states for the failed elements. Let $\mathbf{e}_{k+M}$ to denote the overall error accumulated from t_{k-N} to t_{k+M}. Given Eqn. (4)(12), we can have

$$\mathbf{e}_{k+M,(1,q)} = \left[\mathbf{A}^{N+M} \delta_{k-N}\right]_{(1,q)} + \left[\sum_{i=1}^{N+M} \mathbf{A}^{i-1} \mathbf{v}_{k+M-i} \right]_{(1,q)},$$
$$\mathbf{e}_{k+M,(q+1,n)} = \delta_{k+M,(q+1,n)}. \quad (13)$$

And the corresponding bounded errors are thus

$$\mathbf{e}^{+}_{k+M,(1,q)} = \left[|\mathbf{A}^{N+M}| \epsilon_{\delta_{k-N}}\right]_{(1,q)} + \left[\sum_{i=1}^{N+M} |\mathbf{A}^{i-1}| \epsilon_{v_{k+M-i}} \right]_{(1,q)},$$
$$\mathbf{e}^{+}_{k+M,(q+1,n)} = \epsilon_{\delta_{k+M,(q+1,n)}}. \quad (14)$$

Capability of the Recovery and Resilient Control. As the system keeps operating under the recovery-based resilient control, the drift $\mathbf{e}_{k+M}$ can grow larger and larger over time. When the drift cannot satisfy the maximum tolerable estimation error, i.e., $\mathbf{e}^{+}_{k+M} \not\leq \mathbf{E}$, the system ceases the resilient control and may need to reset the attacked sensors or the portion under attack. In other words, the proposed approach, i.e., the physical-state recovery together with the resilient control, is a conservative way to deal with sensor attacks, which is able to postpone resetting the attacked sensors or the attacked portion. Resetting them usually incurs much higher overhead (e.g., possibly pausing the plant) to the system than does the proposed approach.

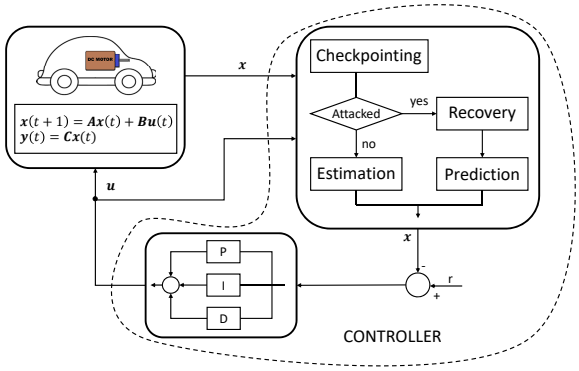


Figure 4. Control system diagram.

Further, different kinds of sensor attacks or faults can affect the system with different time durations. That is, the attacked sensors may become trustworthy again multiple sampling periods later, or after even longer time, or never [12]. Our method is generally applicable to all of these cases, and its advantage is even more noticeable for short-term attacks. That is, for sensor attacks causing transient failures to internal elements or transient attacks, there is even no need to reset them if they can become trustworthy before the condition $\mathbf{e}^{+}_{k+M} \not\leq \mathbf{E}$ occurs. One possible situation for short-term attacks is that attackers just have intermittent physical access to attackees, e.g., an attacker uses some device to attack camera or LiDAR sensors of a passing-by autonomous car [11]. From this perspective, our method is also a more lightweight way to deal with transient attacks.[1]

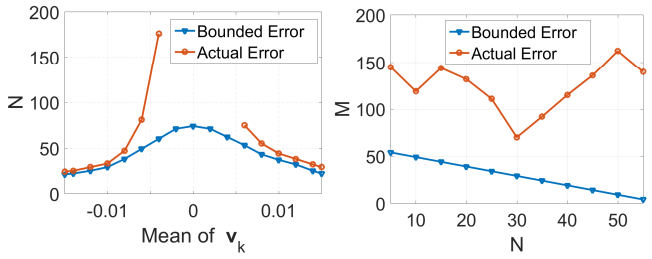
V. Evaluation

To validate the CPS checkpointing and recovery and highlight its benefits on addressing sensor attacks, we conduct two experiments: one is simulator based and the other uses an unmanned vehicle test bed. Both experiments have the same control system diagram as illustrated by Fig. 4. The controller consists of our approach and a PID controller.

A. Simulation-Based Evaluation

Settings and Model. The scenario considered for this experiment is that the operator specifies the desired vehicle speed, and the controller needs to ensure this speed as much as possible even if the speed sensor/encoder is under attack. We use Simulink to build a simulator where a DC motor

[1]Addressing how to reset attacked or faulty sensors is beyond the scope of this paper.



(a) Conservativeness of the error bound (b) The trade-off between N and M

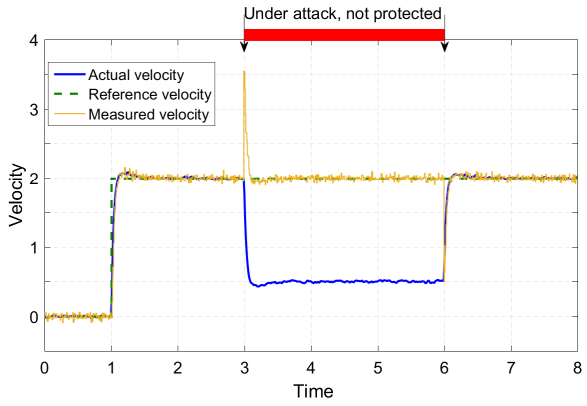
Figure 5. Simulation results about the physical-state recovery and resilient control.

drives an inertial load. DC motors are widely used in electric vehicles and many autonomous car prototypes. We use the dynamic model of the DC motor given by

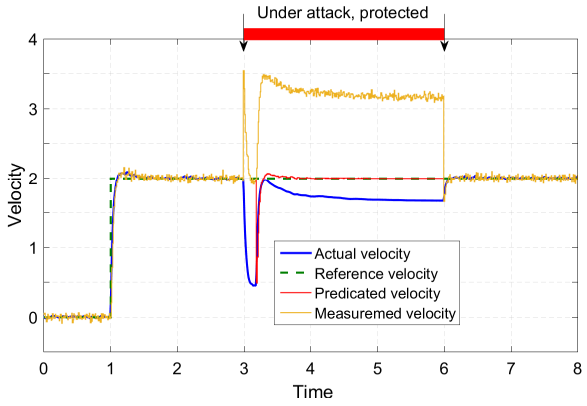
$$\begin{bmatrix} \dot{i} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{K_b}{L} \\ \frac{K_m}{J} & -\frac{K_f}{J} \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} v, \qquad (15)$$

where the current i and the angular velocity ω are considered as the two states of the system. The applied voltage v is the control input, and the angular velocity ω is the output. The parameters of the motor are set as follows. The resistance R and the self-inductance L are set as 1 and 0.5 respectively. Both the armature constant K_m and the EMF constant K_b are set as 0.01. The viscous friction constant K_f is 0.1. The inertial load J is 0.01. Sensor noise (of the speed sensor/encoder) obeys Gaussian distribution with the variance of 0.001. The reference velocity is set as 2, and the required drift from the reference velocity is within $[-0.3, 0.3]$. The sampling period is set as 0.01. We implement the protocol proposed in Section III to checkpoint the angular velocity ω, current i, and applied voltage v. To ensure consistency, each system state records the three parameters' value of the same sampling period.

Simulation Results. Fig. 5(a) illustrates the conservativeness for the error bound of the predicted velocity. For this figure, noise profiles $\mathbf{v}_k$ are uniformly chosen from $[-0.015 + \mu, 0.015 + \mu]$, where μ is the mean. The y-axis N indexes the maximum window size that the system can tolerate, i.e., the furthest consistent state from which the system can start the physical-state recovery as long as the error is within the required range. The figure plots two different cases based on the actual error (by Eqn. (6)) and bounded error (by Eqn. (9)) respectively. We can see that the former case has larger N than does the later case, i.e., by the actual error based case, the system can use a state that is further back for the physical-state recovery. This difference or the error bound's conservativeness increases as the absolute value $|\mu|$ decreases. When $|\mu|$ approaches to zero, the number N of the actual error based case becomes even larger (the corresponding data points are not shown in the figure). On the other hand, when $|\mu|$ is large, the number



(a) Under attack/fault without protection



(b) Under attack/fault with protection

Figure 6. Comparison about actual, predicted and measured velocity among cases with and without the physical-state recovery.

N of the two cases becomes similar and thus the error bound's conservativeness becomes low. Another observation from the figure is that for both cases, the number N increases as the value $|\mu|$ scales down. In addition, the above observations make the following indication. First, they indicate how to choose among detection mechanisms to work with the proposed checkpointing protocol, if different mechanisms have different window sizes. The second indication is about how to determine the window size to work with the proposed checkpointing protocol, if a detection mechanism can use multiple window sizes.

Fig. 5(b) demonstrates the result about the relationship between the window size N and M (i.e., how far to go forward for the recovery-based resilient control), when the overall drift error is within the required range. For this figure, noise profiles $\mathbf{v}_k$ are uniformly chosen from $[-0.01, 0.02]$. The trade-off relationship is clear for the case based on the bounded error (by Eqn. (9)(14)), which is as N increases M decreases and vice versa. By contrast, there is no such relationship for the case based on the actual error (by Eqn. (6)(13)). Further, M of this case is larger than that of the bounded error based case, i.e., it can enable longer time for the recovery-based resilient control than the bounded

error based case. However, to ensure the required range of drift errors, it should leverage the bounded error and its corresponding trade-off relationship to determine N and M for the checkpointing protocol proposed in Section III. In other words, after choosing a detection mechanism (and thus its corresponding window size), the time duration to run the resilient control can be determined accordingly. In addition, detection mechanisms with lower detection delay (i.e., a smaller window size) can allow the system to execute the resilient control for a longer time and thus further postpone resetting the attacked components. This observation further indicates the importance of fast detection. Note that this simulation is not show the performance of a specific detection mechanism, but to demonstrate the impact of detection delay.

Fig. 6 shows comparison between two cases with and without the protection of our approach. The attack on the encoder starts from time 3 and ends at time 6, which adds a constant value of 1.5 to each sensor measurement. Noise profiles $\mathbf{v}_k$ are uniformly chosen from $[-0.01, 0.02]$, i.e., the mean is a positive value of 0.005. As shown by Fig. 6(a), if without physical-state recovery, the actual velocity declines quickly to 0.5 after the attack starts at time 3. The measured velocity is 2 and equal to the reference velocity but the measurement is untrustworthy, because the attack adds 1.5 to each sensor measurement. The controller uses this compromised measurement to derive control inputs, which makes the actual velocity far from the reference value.

By contrast, Fig. 6(b) shows the result with the protection of our approach. Between time 3 and time 3.2, our approach has not been applied yet and thus the actual velocity decreases to 0.5 because of the attack. Note that 0.2 is the time length of the sliding window as to the setting here. At time 3.2, the system first carries out the physical-state recovery to make a prediction on the system state (i.e., velocity). Comparing the red and blue curves, we can see that the prediction of this time point is rather accurate with little error. Then, the system performs the recovery-based resilient control based on the prediction. Between time 3.2 and time 6, the controller depends on the predicted velocity, instead of sensor measurements, to derive control inputs. Although the predicted velocity is equal to the reference value, the actual velocity gradually drifts from the reference value due to the error accumulation caused by noise $\mathbf{v}_k$. We can see that the drift here is much less than that in Fig. 6(a). After the encoder becomes trustworthy again from time 6, the controller switches to use sensor measurements and the actual velocity turns to the reference value. There is no need to rest the encoder, because the system has not drifted too much (e.g., outside the required range) till the time when the encoder comes back trustworthy. Hence, our approach well protects the system from the transient attack. In addition, if the attack continues after time 6 and lasts for longer time, the system needs to cease the resilient control before it drifts outside the required range.

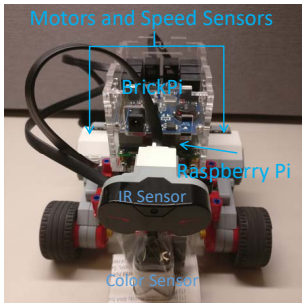


Figure 7. The unmanned vehicle test bed.

B. Case Study

Setup and Implementation. We have considered the speed control in the previous subsection. In this subsection, we consider a different scenario to further demonstrate the capability of our approach, where the controller needs to ensure the vehicle to travel in a straight line.

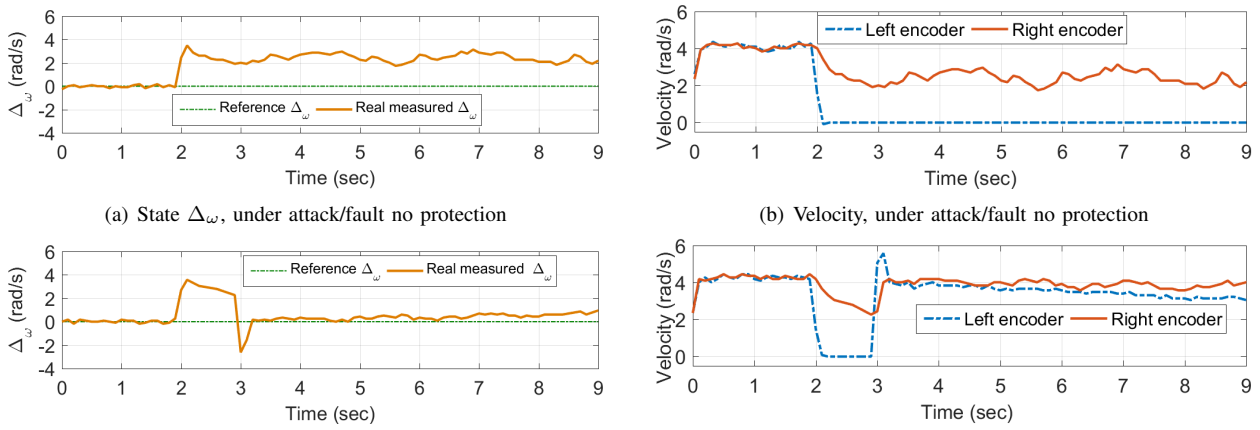
The unmanned vehicle test bed, as shown in Fig. 7, is used for the illustration. The test bed assembles two boards: Raspberry Pi and BrickPi. Raspberry Pi runs Linux OS and interfaces with motors and sensors via drivers implemented on BrickPi. The test bed uses the motors and sensors available in Lego Mindstorms [26]. Each front wheel is driven by a motor and each motor has a built-in speed sensor. The test bed also assembles other two sensors, i.e., IR sensor and color sensor, which can be used to detect and trace objects. However, to illustrate the use of our design, we only use the two speed sensors in this experiment. We implement our design in C language.

Making turns of the vehicle is implemented in a manner that the vehicle turns right (left) when the left (right) front wheel travels at a higher speed than does the right (left) front wheel. Thus, the front two wheels need to have the same speed in order for the vehicle to travel in a straight line. We use Δ_ω to denote the difference between the speeds of the two front wheels/motors, i.e., $\Delta_\omega = \omega_l - \omega_r$, and use Δ_v to denote the difference between the voltages applied to the two motors, i.e., $\Delta_v = v_l - v_r$. We consider Δ_ω and Δ_v as the state and control input of the system, respectively. The state space equation is

$$\dot{\Delta_\omega} = -\frac{25}{3}\Delta_\omega + 5\Delta_v. \tag{16}$$

Experimental Results. Fig. 8 presents the results of a deployment during experiments carried out on a rough carpet floor. The voltage value is bounded, and initially we set the applied voltage to both motors as 50% of the bound. The sensor attack starts at 2 *sec* and modifies each measurement of the left speed sensor to be a constant value of 2π *rad*. The sampling period is set to be 0.1 *sec*.

Fig. 8(a)8(b) plots the result about the system state Δ_ω and motors' velocity for one time of experiment without the protection of our approach. After the attack starts at 2 sec, the measurement of the left motor fed into the controller is



(a) State Δ_ω, under attack/fault no protection

(b) Velocity, under attack/fault no protection

(c) State Δ_ω, $N = 10$, under attack/fault with protection

(d) Velocity, $N = 10$, under attack/fault with protection

Figure 8. Experimental results about a deployment during two experiments with and without protection of our approach.

compromised to be a rather large value (i.e., $2\pi\ rad$). With this compromised value, the applied voltage to the left motor needs to decrease in order to achieve the reference state, i.e., the velocity of both motors is the same. However, this makes the real velocity of the left motor decline to zero after several periods, shown in Fig. 8(b). Thus, the realistic situation is that the velocity difference between the two motors is large, shown in Fig. 8(a). The real system state is far from the reference value, and the vehicle keeps making turns instead of travelling in a straight line.

Fig. 8(c)8(d) demonstrates the result for one time of experiment with the protection of our approach. The window size N is set as 10, which means that the physical-state recovery occurs at 3 sec as to the setting here. The recovery-based resilient control also starts at this time point. Between 2 sec and 3 sec, the system is under attack and the physical-state recovery has not started yet. Thus, during this time interval, the velocity of the left motor declines to zero and the system state is far from the reference state. After 3 sec, the physical-state recovery and resilient control are brought on-line to start control the vehicle using the predicted state. At first, the system state is very close to the reference state, shown in Fig. 8(c), that is, the velocity of the two motors is nearly the same, shown in Fig. 8(d). This indicates that the prediction by the physical-state recovery is rather accurate. As time goes on, under the resilient control, the system state slowly drifts away, and the velocity difference grows little by little. The drift here is much less than that of Fig. 8(a), and grows not so fast.

VI. Related Work

Checkpointing and Recovery. Checkpointing protocols evolve significantly from the earliest works that employ synchronized checkpointing such as [27]–[29], to the recent works that leverage various consistency-guaranteed methods to allow asynchronized checkpointing, such as [21], [30], [31]. These approaches record cyber-states while our approach checkpoints physical-states and control inputs. Further, the stored physical-states and control inputs of an individual checkpoint are all in the same sampling period. From this perspective, our approach can be seen as synchronized checkpointing. Furthermore, our checkpointing protocol explicitly considers detection delay. This makes it different from protocols such as [24], [25], which assumes instant detection upon attacks' or faults' occurrence.

Handling Sensor Attacks. There are many defense techniques that improve system resilience against sensor attacks, such as [6], [16]–[18]. Most of them confine to systems with sensor redundancy, that is, multiple sensors (partially) measure the same physical variables. As the case with no sensor redundancy or when all redundant sensors are compromised, they do not work. By contrast, our approach has no such limitations and thus comes with wider applicability.

VII. Conclusion

In this work, we have studied the problem of enhancing attack-resilience for a CPS system under attacks on the environment of the controller such as attacks on sensors. We have developed a procedure of CPS checkpointing and recovery. Firstly, we propose a new concept of physical-state recovery, which is defined as rolling the system forward to match values of internal elements with the states of the plant. Secondly, we design a slide window based checkpointing protocol that defines how to record system states for the recovery. The proposed procedure possesses several advantages including low overhead to the system, no modification to the existing controller, and no limitations about sensor redundancy. Thirdly, we present a use case of the procedure on handling sensor attacks. Finally, we validate the feasibility of our design using a DC motor simulator and an unmanned vehicle case study.

Acknowledgment

This work was supported in part by NSF CNS-1505799, the Intel-NSF Partnership for Cyber-Physical Systems Security and Privacy, and ONR N000141712012.

REFERENCES

[1] A. M. Wyglinski, X. Huang, T. Padir, L. Lai, T. R. Eisenbarth, and K. Venkatasubramanian, "Security of autonomous systems employing embedded computing and sensors," *IEEE micro*, 2013.

[2] M. Amoozadeh, A. Raghuramu, C.-N. Chuah, D. Ghosal, H. M. Zhang, J. Rowe, and K. Levitt, "Security vulnerabilities of connected vehicle streams and their impact on cooperative driving," *IEEE Communications Magazine*, 2015.

[3] S. Parkinson, P. Ward, K. Wilson, and J. Miller, "Cyber threats facing autonomous and connected vehicles: Future challenges," *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[4] A. A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2008.

[5] N. Adam, "Workshop on future directions in cyber-physical systems security," Department of Homeland Security, Tech. Rep., January 2010.

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

[7] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*. San Francisco, 2011.

[8] K.-T. Cho and K. G. Shin, "Error handling of in-vehicle networks makes them vulnerable," in *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2016.

[9] Y. Shoukry, P. Martin, P. Tabuada, and M. Srivastava, "Non-invasive spoofing attacks for anti-lock braking systems," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2013.

[10] A. H. Rutkin, ""spoofers" use fake gps signals to knock a yacht off course," MIT Technology Review, August 14, 2013.

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

[12] R. Mitchell and I.-R. Chen, "A survey of intrusion detection techniques for cyber-physical systems," *ACM Computing Surveys*, 2014.

[13] F. Pasqualetti, F. Dörfler, and F. Bullo, "Attack detection and identification in cyber-physical systems," *IEEE Transactions on Automatic Control*, 2013.

[14] J. Park, R. Ivanov, J. Weimer, M. Pajic, and I. Lee, "Sensor attack detection in the presence of transient faults," in *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*. ACM, 2015.

[15] Z. Feng, N. Guan, M. Lv, W. Liu, Q. Deng, X. Liu, and W. Yi, "Efficient drone hijacking detection using onboard motion sensors," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017.

[16] H. Fawzi, P. Tabuada, and S. Diggavi, "Secure estimation and control for cyber-physical systems under adversarial attacks," *IEEE Transactions on Automatic Control*, 2014.

[17] R. Ivanov, M. Pajic, and I. Lee, "Attack-resilient sensor fusion for safety-critical cyber-physical systems," *ACM Transactions on Embedded Computing Systems*, 2016.

[18] S. Z. Yong, M. Zhu, and E. Frazzoli, "Resilient state estimation against switching attacks on stochastic cyber-physical systems," in *IEEE 54th Annual Conference on Decision and Control (CDC)*. IEEE, 2015.

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

[20] P. S. Duggirala, T. T. Johnson, A. Zimmerman, and S. Mitra, "Static and dynamic analysis of timed distributed traces," in *IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2012.

[21] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, 2002.

[22] S. N. Narayanan, S. Mittal, and A. Joshi, "Using data analytics to detect anomalous states in vehicles," *arXiv preprint arXiv:1512.08048*, 2015.

[23] P. Uppuluri and R. Sekar, "Experiences with specification-based intrusion detection," in *Recent Advances in Intrusion Detection*. Springer, 2001.

[24] S. Punnekkat, A. Burns, and R. Davis, "Analysis of checkpointing for real-time systems," *Real-Time Systems*, 2001.

[25] E. Gelenbe and D. Derochette, "Performance of rollback recovery systems under intermittent failures," *Communications of the ACM*, 1978.

[26] Lego, "31313 Mindstorms EV3," https://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313.

[27] Y. Tamir and C. H. Sequin, "Error recovery in multicomputers using global checkpoints," in *International Conference on Parallel Processing*. IEEE, 1984.

[28] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Transactions on Computer Systems*, 1985.

[29] K. G. Shin, T.-H. Lin, and Y.-H. Lee, "Optimal checkpointing of real-time tasks," *IEEE Transactions on computers*, 1987.

[30] D. B. Johnson, "Distributed system fault tolerance using message logging and checkpointing," Ph.D. dissertation, Rice University, 1990.

[31] K.-F. Ssu, B. Yao, and W. K. Fuchs, "An adaptive checkpointing protocol to bound recovery time with message logging," in *IEEE Symposium on Reliable Distributed Systems*. IEEE, 1999.