Summer 7-16-2021

Real-time Adaptive Sensor Attack Detection and Recovery in Autonomous Cyber-physical Systems

Francis Akowuah
Syracuse University

ABSTRACT

Cyber-Physical Systems (CPS) tightly couple information technology with physical processes, which rises new vulnerabilities such as physical attacks that are beyond conventional cyber attacks. Attackers may non-invasively compromise sensors and spoof the controller to perform unsafe actions. This issue is even emphasized with the increasing autonomy in CPS. While this fact has motivated many defense mechanisms against sensor attacks, a clear vision of the timing and usability (or the false alarm rate) of attack detection still remains elusive. Existing works tend to pursue an unachievable goal of minimizing the detection delay and false alarm rate at the same time, while there is a clear trade-off between the two metrics. Instead, this dissertation argues that attack detection should bias different metrics (detection delay and false alarm) when a system sits in different states. For example, if the system is close to unsafe states, reducing the detection delay is preferable to lowering the false alarm rate, and vice versa. This dissertation proposes two real-time adaptive sensor attack detection frameworks. The frameworks can dynamically adapt the detection delay and false alarm rate so as to meet a detection deadline and improve usability according to different system statuses. We design and implement the proposed frameworks and validate them using realistic sensor data of automotive CPS to demonstrate its efficiency and efficacy.

Further, this dissertation proposes *Recovery-by-Learning*, a data-driven attack recovery framework that restores CPS from sensor attacks. The importance of attack recovery is emphasized by the need to mitigate the attack's impact on a system and restore it to continue functioning. We propose a double sliding window-based checkpointing protocol to remove compromised data and keep trustful data for state estimation.

Together, the proposed solutions enable a holistic attack resilient solution for automotive cyber-physical systems.

REAL-TIME ADAPTIVE SENSOR ATTACK DETECTION AND RECOVERY IN AUTONOMOUS CYBER-PHYSICAL SYSTEMS

by

Francis Enoch Akowuah

B.S., Kwame Nkrumah University of Science and Technology, 2006
M.S., North Carolina A & T State University, 2013

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

Syracuse University
July 2021

To my wife, Esther.

ACKNOWLEDGMENTS

First and foremost, I would like to thank the Lord God Almighty for without His grace, mercy, blessings and strength, this study would not have been possible.

I am proud to acknowledge my wife Esther and daughters Fosuah, Kyerewaa, and Aseda for all of the support and encouragement that they have given me during my doctoral studies. I cannot begin to express how excited I am to spend more time with them after many years of living with them but not being there for them. Without their help to make the doctoral process more tenable, I doubt I would have made it through. For this, I will forever be grateful. I am looking forward to more trips to the park and all the exciting places that have remained a dream so far. My dad Christian and siblings Gifty, Patience, Christina, Emmanuel, Mary, and Samuel have all supported this pursuit with their prayers and encouragements. I am also thankful for the prayer support from all my in-laws Ernest, Mary, Abigail, Dorcas, Isaac, Daniel, and Deborah.

My advisor, Dr. Fanxin Kong, took a chance on me when no one would. He has been my mentor in both research and technical pursuits. He has patiently taught me more than I ever imagined about the fundamentals of research, writing, and technical presentation. All of these aspects will serve me very well in the years to come. I am very appreciative of his ungrudging support and care, understanding, and pieces of advice on my career.

Dr. Jae Oh has been an incredible help to me during my course of study. His help enabled me to access funding opportunities that made it possible to keep food on the table

for my family. Without his care, support, and encouragement, quitting the program was the only option for me.

I am also grateful to Dr. Wenliang Du who gave me a variety of opportunities to sharpen my technical skills, explore mobile platform security and teach in his renowned SEED workshop. He spent his fruitful moments in the morning teaching us the art of finding good research problems to think about and I will remain forever grateful for that. This skill will serve me well as a principal investigator on my own research projects in the future.

The help and friendship of Kenneth Fletcher, Amit Ahlawat, Mengyu Liu, Lin Zhang, Romesh Prasad, and Carlos Espinoza have made the long doctoral process possible. I am grateful to them all for the encouragement, technical assistance, and friendship.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Overview of Cyber-physical Systems

Cyber-physical systems (CPS) are systems that monitor or control a physical mechanism using computer-based algorithms. These systems are enabled by a deep integration of software and hardware components. The software components, usually referred to as *cyber* components, consist of the computing and communication aspects of the system. The hardware component or the *physical* components refer to sensors and actuators. Sensors measure the state of system whereas actuators control and move the mechanism or system such as increasing throttle, opening a valve etc. Note that sensors and actuators can also be referred to as the interface between the physical and the cyber world. The basis of CPS includes embedded systems, computers, and software embedded in devices whose primary focus is not computation including but not limited to medical devices, cars, and scientific instruments. CPSs are feedback systems with a feedback loop where the computations affect the physical process and vice versa. Examples of CPS include smart grid systems, medical monitoring devices, industrial control systems, building controls, autonomous vehicles etc.

Increasingly, CPSs play important roles in government, critical infrastructure and daily lives. While the modern society has reaped many benefits from these systems, the full

economic and societal potential has not been realized. Hence, huge investments are being made globally to develop the technology.

Autonomous Cyber-Physical Systems (CPS), such as self-driving cars and unmanned aerial vehicles (UAV), are becoming an integral part of our daily lives. For example, Amazon's Prime Air service seeks to use drones to deliver orders up to five pounds in 30 minutes or less and has already demonstrated its feasibility in [7]. UAVs have also been seen in applications such as aerial photography [20], policing and surveillance [19] [36], infrastructure inspections [35], construction site management [21] and many others. Self-driving cars continue to attract huge investments from big companies and they are expected to be in common use in the near future [39] [77].

1.2 Problem Overview

Due to the safety-critical roles that they play, autonomous CPS security continues to be an essential requirement for its safe functioning. However, the deep intertwinement of physical processes, software, and hardware has increased the attack surface of systems that once had closed architectures. At the cyber level, CPS now suffer attacks similar to what traditional computer software and network face such as buffer overflow, network eavesdropping (sniffing), packet spoofing, data modification attacks, etc. At the physical level, an adversary is able to compromise the physical environment of a system that leads to the injection of malicious signals which impairs the function and behavior of the system. This type of attack is referred to as physical attacks and examples include dazzling cameras with light, injecting false radar signals, injecting false GPS signals, etc.

Comparatively, defense solutions for cyber attacks are more advanced than physical attacks because traditional cyber-security solutions are viable defense against cyber attacks. For instance, firmware hardening [81], control-flow integrity [16], memory isolation [48], etc, can defend CPS cyber attacks. These solutions are, however, weak against physical attacks since these attacks do not directly target the software components. This is especially emphasized by non-invasive sensor attacks. These attacks do not require physical access to the target component and have been shown to be easy (requiring a modicum of knowledge) and inexpensive (requiring cheap equipment to execute). Rutkin [80] showed how non-invasive attacks enabled malicious signals to be injected into GPS sensors, and in the end misguided a yacht off course. Similarly, Shoukry et al. [86] demonstrated how non-invasive attacks on wheel speed sensors influenced Anti-lock Braking Systems (ABS) of a vehicle to malfunction. Petit et al. [76] also showed how an automotive CPS camera and LiDAR can be attacked remotely. In addition, the consequences of sensor attacks will be even exaggerated as the autonomy increases. Therefore, there is the urgent need for physical attack defense solutions.

The urgent need to protect autonomous CPS from physical sensor attacks has motivated a lot of research efforts which can be can be categorized into (1) attack detection and (2) attack recovery.

1.2.1 Attack Detection Limitations

Attack detection is one of the important strategies for securing CPS from malicious attacks. The ability to detect malicious behaviors early sets the stage to provide

countermeasures that either prevent further attacks or mitigates the damaging effects of the attack. Many research efforts have proposed detection solutions such as attack-resilient sensor fusion [41, 60, 65], model-based attack detection [14, 29, 78], and data-based detection [2, 32, 37, 43, 75, 84]. However, the timing and usability of attack detection have not been adequately addressed in existing works. This timing constraint is the detection deadline, before which attacks must be detected. The usability refers to the false alarm rate, and a lower (higher) rate means a better (worse) usability. Existing works tend to minimize the detection delay and false alarm rate at the same time. However, the goal is deemed to be unachievable because of the clear trade-off between the two metrics, i.e., lower delay coming with higher false alarm rate, and vice versa [29, 89, 91]. Hence, we believe that attack detection should have a preference on different metrics when a system runs in different states.

1.2.2 Attack Recovery Limitations

The new CPS threats have motivated many research efforts to defend against sensor attacks. However, most of them have focused on attack detection rather than recovery measures. Raising attack alerts, usually done in attack detection works, do not ensure the continuous functioning of the CPS. Hence, to respond to attack alerts, recovery measures are required to mitigate the effect of an attack on a system and continue the system's operation with minimum interruption.

Only a few existing works have addressed attack-recovery in any form. As the pioneer work in this research thread, Kong et al. [51, 102] assumes full observability of the system

and replaces measurements of compromised sensors by model-predicted values. Fei et al. [23] follow the idea of [51] and proposes a redundant controller for attack recovery that is trained also based on the system model. Although these works validate the performance under their own settings, they require prior knowledge of system dynamics that builds the system model.

1.3 Proposed Solution

1.3.1 Detection

This dissertation proposes two *real-time adaptive* sensor attack detection frameworks that can dynamically adjust detection delay and false alarms. The key rationale behind this framework is as follows.

(i) Why real-time? Given safety-critical CPS, timing is important, as untimely defense, that is, detection of an attack after consequences occur, is just as damaging. For example, consider the cruise control function under a speed sensor spoofing attack that changes the true measurement to a smaller value. Then the vehicle is misled to accelerate so that the real speed can be much higher than the desired. This attack needs to be detected before the vehicle crashes into the front car. This timing constraint is referred to as the *detection deadline*, before which attacks must be detected.

(ii) Why adaptive? On the one hand, a shorter detection delay is not always favorable. In the end, we can have an attack detector that raises an alert at every control period. The detector will discover an attack once it occurs, and thus the detector has the shortest detection delay. However, this will give an unmanageable number of false alarms and thus

unacceptably low usability. On the other hand, an alert can be raised after monitoring multiple control periods to ascertain the occurrence of an attack. However, this can lead to increased detection delay. Hence, we argue that there is a need to adapt the attack detection so that it can make the appropriate trade-off. For example, if the system is already close to unsafe states and thus the detection deadline is stringent, reducing the detection delay will be preferable to lowering the false alarm rate, and vice versa. To enable real-time adaptive detection, this dissertation proposes two frameworks: a CUSUM-based framework and a variable window-based framework.

The CUSUM-based real-time adaptive attack detection framework consists of three necessary components: attack detector, behavior predictor, and drift adaptor, as shown in Fig. 3.1. The technical contribution for each component is as follows.

(i) Attack Detector. As the core of our framework, this component detects anomalies using a CUSUM algorithm that monitors the cumulative sum of residuals between the nominal (estimated by the behavior predictor) and observed sensor values. The algorithm will raise an alarm when the cumulative sum of the residuals is greater than a predefined threshold. Importantly, we augment this algorithm with a drift parameter that governs both the detection delay and false alarms. That is, the algorithm can adjust the two metrics by changing the drift parameter.

(ii) Behavior Predictor. This component estimates nominal sensor values that are fed to the core component. It uses a deep learning (DL) model that is offline extracted through uncovering and exploiting both the local and complex long-term dependencies in multivariate sequential sensor measurements. Thus this model depends on little knowledge of the physical system (e.g., dynamics). Further, this model leverages

convolutional neural network (CNN) and recurrent neural network (RNN) to capture non-linear aspects in sensor data and uses autoregressive models to capture linear aspects. This combination results in high robustness and scalability in handling the sequential sensor data.

(iii) Drift Adaptor. The third component is a drift adaptor that estimates a detection deadline and then determines the drift parameter. The detector component uses this parameter for adjusting the detection delay to ensure timely detection as the detection deadline varies over time.

The variable window-based real-time adaptive attack detection framework also consists of three components: attack detector, state predictor, and window adaptor. The technical contribution for each component is as follows.

(i) Attack Detector. The attack detector uses a stateful detection strategy to monitor the residual sequence for a period of time called the time window. It raises an alert whenever the accumulated residual sequence within the time window exceeds a predefined threshold. An essential parameter to the detection algorithm is the window length, which controls or adjusts the detection delay and false alarms metrics.

(ii) State Predictor. The second component of the framework, state predictor, uses a temporal convolutional network (TCN) model to estimate the nominal sensor values which are used in the attack detection algorithm. The TCN model captures the relationship among correlated sensors to make predictions.

(iii) Window Adaptor. This two-phased component estimates a detection deadline and then determines the window length parameter. This is the parameter that is passed to the attack detector component for adjusting the detection delay to ensure timely detection.

We implement our frameworks and validate them using realistic sensor data of automotive CPS from the AEGIS Big Data Project [44]. The results demonstrate that our frameworks can detect attacks in a real-time manner.

1.3.2 Recovery

To address the limitations of existing attack recovery, this dissertation proposes Recovery-by-Learning, a data-driven attack recovery framework that restores automotive cyber-physical systems from sensor attacks. The framework requires little knowledge of the system's dynamics, but leverages natural redundancy among heterogeneous sensors and historical data for attack recovery. Specially, the framework consists of two major components: state predictor and data checkpointer. At the core of this solution are novel techniques to realize these components.

First, the state predictor is activated to estimate system states when an attack is detected. The predicted states are forwarded to the controller to calculate and issue appropriate control commands to bring the system back to normalcy. The predictor is built on a deep learning model that captures the nominal system behavior. The model exploits the natural redundancy as well as the short and long term temporal correlation among heterogeneous sensors on an autonomous CPS, through combining convolutional neural network (CNN) and recurrent neural network (RNN).

Second, the data checkpointer executes in the normal mode when no attack is detected. It employs a checkpointing protocol to remove corrupted data and keep valid historical data as input to the state predictor to make state estimation. The protocol uses double

sliding windows: detection window and logging window. The former accommodates the substantial detection delay (i.e., the time interval between the start of an attack and the detection of it), during which the correctness of the sensor data is still in question and thus using them may result in unsuccessful recovery. The logging window governs sufficient trustful data for the state prediction.

We implement and evaluate the effectiveness of our framework using a real-world data set, AEGIS Big Data Project [44], and a ground vehicle simulator, Ardupilot SITL Rover [1]. The results show that the proposed framework is capable of ensuring continuous functionality in presence of sensor attacks.

1.4 Overview of dissertation

This dissertation asks three questions (1) *how can we detect when an attack occurs?*, (2) *how can we adapt the behavior of the detector so that it can meet a detection deadline?* and (3) *how can an attacked automotive CPS continue to function during an attack?*. All these questions are aimed at securing the autonomous CPS against physical attacks. Cyber attacks are out of scope since many traditional software security solutions are able to defend them. The proposed solutions together with cyber attack defense solutions can enable a holistic attack resilient solution for autonomous CPS.

The dissertation describes the design, implementation and evaluations of the proposed frameworks mentioned in the previous section. It is presented in the following manner. Chapter 1 is the introduction of the thesis and an overview of the material presented within the dissertation. Chapter 2 is a survey of background material that presents a taxonomy of

the current state of the art of CPS attack detection and attack recovery. Chapter 3 presents the real-time adaptive sensor attack detection frameworks by explaining the novel contributions of their design, implementation and evaluation. Chapter 4 presents the design, implementation and evaluation details of the attack recovery framework. Chapter 5 provides a summary of all findings, conclusions and future work.

1.5 Previous publications

This dissertation is composed of works that have been published in peer-reviewed conferences and manuscripts that are currently under submission as a peer-reviewed conference paper. A portion of the attack detection work is published in the proceedings of 27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2021) [5]. The window-based attack detection portion of the dissertation is currently under submission to the 40th edition of International Conference On Computer Aided Design (ICCAD 2021). The content of chapter 2 has been published in the Fourth International Conference on Connected and Autonomous Driving (MetroCAD 2021). Chapter 4 is based a work which has been accepted for publication in the 27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2021).

2. BACKGROUND

This chapter presents a survey of existing sensor attack detection and attack recovery proposed solutions. Understanding the techniques, the capabilities and limitations of these prior works provides an understanding of how the state of the art in attack detection has advanced and provides an avenue to explore future research effort.

2.1 Physical Invariant Based Attack Detection

This section studies how to detect attacks on autonomous vehicles, and specially focus on physical invariant-based attack detection. A physical invariant (PI) is defined as a property that a physical system always holds, i.e., the evolution of system states (usually measured by sensors) follows immutable physical laws. We first discuss existing research efforts of PI-based attack detection and classify them according to the knowledge of physical invariants and sensor redundancy. While the autonomous vehicle faces both cyber attacks and physical attacks we only focus on the latter. As mentioned in Chapter 1, defenses against cyber attacks are relatively advanced due to the many traditional cybersecurity techniques already available. Comparatively, proposed solutions for defending against physical attacks are few and more challenging. We survey research efforts that seek to detect physical attacks in autonomous vehicles. Since the physical

properties are often measured with sensors, we focus on research works that address the detection of false sensor attacks.

2.1.1 Classification based on physical invariant knowledge

Physical systems have properties that are guarded by immutable physical laws. When attacks are successfully launched, they violate these laws. In order to determine such a violation, it is essential to have a model that accurately approximates the nominal system behavior. The observed behavior can then be compared with the expected behavior (based on the model) to determine a violation of the physical invariant. This has been the general idea behind many attack detection publications in recent years.

Building an accurate model to approximate the nominal system behavior requires knowledge about the system and its dynamics. Modeling the complete system dynamics requires in-depth knowledge and expertise which may not always be available. Hence, recent publications have used two approaches to learn system dynamics. We group publications into two groups namely (1) black box and (2) grey box based on how they capture the system dynamics or physical invariants in their model. Further, we discuss how the model is used for attack detection.

Black-box approach

Publications in this category treat the system as a black box and build a model from the system data, such as sensor readings, control input and output, and system logs. The insight of this approach is that, when the system operating in a normal state, the data or

readings captured by the sensor are directly proportional to the system obeying physical laws. Therefore, the data model that is built from the system data reflects the physical invariant of the system. The popular tools that have been employed in publications to learn system behavior from system data are machine and deep learning techniques. The techniques mine for relevant information and/or relationships among nominal system data.

The black box attack detection approach often has two phases: online and offline phases. The offline and online phases are summarized in Fig. 2.1 and Fig. 2.3 respectively. The offline phase or the *model training* phase starts with collecting data about the system usually consisting of sensor or actuator data. The data collected is pre-processed in order to improve the quality of the data as well as transform it into a form that is required by the chosen machine or deep learning model. The data pre-processing step may include one or more of the following: handling null values, handling categorical values, standardization, and one-hot encoding. The pre-processed data is fed into the machine or deep learning model such as a convolutional neural network (CNN), recurrent neural network (RNN), autoencoder, regression model, etc. The researchers in the papers we selected make different contributions at this stage. Some combine two or more DL/ML models so their trained models can learn certain patterns of interest. Others also reuse existing DL/ML architectures or make simple changes to existing ones. During model training, the output data of the DL/ML model is compared with ground truth data and a loss function calculates a score such as the reconstruction error, prediction error or assigns a label. The training process continues by optimizing and updating the model using the score obtained in the previous step. The output of the offline phase is a trained model that is capable of predicting or classifying observed system behavior.

Table 2.1: Taxonomy based on physical invariant.

Black Box	Grey Box	Correlation
He et. al [37] Li et. al [58] Van et. al [92] Javed et. al [42] Shin et. al [85]	Quinonez et. al [78] Choi et. al [14]	He et. al [37] Ganesan et. al [25] Li et. al [58] Parker et. al [75] Guo et. al [31]

The online phase deploys the trained model so that it can make predictions or classifications when the system is running. The anomaly detection algorithm, in most of the papers surveyed, compares the output of the trained model with the observed signals and then calculates an anomaly score using time-window approaches or statistical methods such as cumulative sum (CUSUM), chi-square, etc. The detector raises an alert whenever the anomaly score exceeds a certain pre-determined threshold.

Grey-box approach

Attack detection solutions in this category have some knowledge about the system and even know the physical invariant. Instead of learning the structure of the model, such papers make their contributions by learning the parameters of the invariants utilizing techniques such as system identification (SI). Such solution is provided in [78] and [14]. Generally, these solutions also have two phases: offline and online phases as shown in Fig. 2.2. The offline phase extracts the physical invariants that are used to build a model that captures the underlying or expected relationships between the sensors and actuators. In other words, the model captures the expected inputs and outputs of the system. The techniques used at this phase may also capture the expected relationship among sensors.

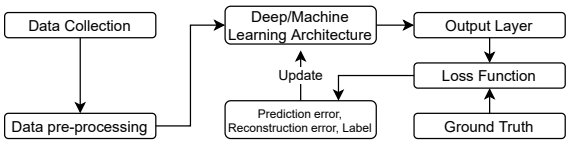


Fig. 2.1.: The offline phase

The solutions in this category have explored both linear [14] and non-linear approaches [78] to describe the physical invariants of AVs. The linear approach assumes a Linear Dynamical State-space (LDS) system which is widely used in system dynamics and control. LDS is given as:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{2.1}$$

where $x_k \in \mathbb{R}^n$ denotes the autonomous vehicle's physical-state vectors; $u_k \in \mathbb{R}^m$ is the control input vectors; $y_k \in \mathbb{R}^p$ denotes the AV's output vectors from measurements of sensors. A, B, and C are the system matrices that are unique for each physical process. Hence, each AV has unique values for A, B and C. The proposed solutions in this category use various techniques to learn these system matrices' parameters, popular among them is system identification.

System identification (SI) is a control system engineering methodology that is used to learn the parameters for the system matrices. The two inputs to the SI method are (1) a control invariant template i.e. equation of a certain degree/form with unknown coefficients/parameters and (2) a vehicle profiling measurement data set including the system inputs, outputs, and states. The vehicle profiling measurement data set is obtained by letting the subject autonomous vehicle perform a set of missions or rides. The runtime

inputs (target states) and system states are measured and recorded during the execution of the missions. When the needed inputs are provided, the SI method then performs computations that instantiate the unknown system matrices (A, B, and C). The resultant equation, therefore, becomes the model for the system which is used in the online phase to predict the behaviors of the autonomous vehicle based on inputs and states. Essentially, the resultant equation serves as the control invariants of the vehicle [14].

Although the linear invariant approach works for a wide number of dynamical systems, autonomous vehicles tend to follow a non-linear invariant as noted in [15, 28, 61, 78]. This approach requires more complex equations than LDS. Authors in [78] indicate that the physical invariants of the quad-copter used in their experiment can be described with 12 non-linear differential equations "that exploit Newton and Euler equations for the 3D motion of a rigid body". The equations oversee the position, speed, angles, and angular speed of the quad-copter. Note here that each type of autonomous vehicle will have its own set of non-linear differential equations that describe its physical invariants. The parameters of these equations are learned using the SI method discussed above for linear systems. Besides the non-linear equations' input, the parameters are

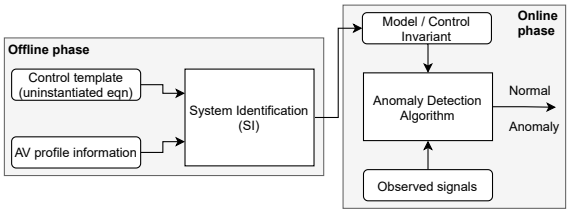


Fig. 2.2.: The general workflow for the grey-box approach. It consists of (1) offline phase where parameters of the control template are learned and (2) online phase where the anomaly detection algorithm uses model predictions and observed signals to determine presence or absence of anomaly

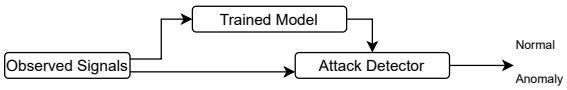


Fig. 2.3.: The online phase

learned in the same way. Particularly, the learning of non-linear parameters is formulated as an optimization problem which is given as [78]:

$$\min_{\mathcal{P}} \sum_{t=1}^{T} (H_t(\mathcal{P}, U_t) - Y_t)^2 \tag{2.2}$$

where U and Y are the input and output data respectively; $\mathcal{P}$ refer to the set of unknown parameters $\{p_1, p_2...\}$. $H_t(\mathcal{P}, U_t)$ denote the estimated output at each sampling instant t for the given parameters $\mathcal{P}$ and the input U_t. Note that $H_t(\mathcal{P}, U_t)$ is the solution the differential equations $F(\cdot)$. The goal of Eqn. 2.2 is to find the parameters $\mathcal{P}$ that better fit the data. In other words, Eqn. 2.2 seeks to find the set of parameters $\mathcal{P}$ that minimize the least square error between the estimated output $H_t(\mathcal{P}, U_t)$ and the measured output Y. Once the unknown parameters are computed, the resultant equations, therefore, become the model for the system which is used in the online phase to predict the behaviors of the autonomous vehicle based on inputs and states.

The online phase consists of an anomaly detection mechanism or algorithm that simply compares the predictions of the model that was built during the offline phase with observed signals or states. The difference between the predictions and the observed states, also called the *residual error*, are accumulated in two ways. The first approach accumulates the residual error as long as no attack has been detected as was done in [78]. The accumulation is reset whenever an attack is detected. The second approach

accumulates the residual error for a set period of time (window) and then resets whenever the time window expires [14]. Either way, an alarm is raised whenever the accumulated residual error exceeds a predetermined threshold.

While the solutions in this category are robust in their attack detection role, they remain weak against stealthy attacks mainly due to perturbations and uncertainties in the model. Stealthy attacks create small deviations over time by spoofing or creating malicious data that allow the system to behave seemingly normally. Stealthy attacks are hard to defend against and remain an open problem in autonomous vehicle attack detection. Researchers in [78] are the first to propose a solution to stealthy attacks in autonomous vehicles.

2.1.2 Classification based on correlation

Immutable physical laws cause multiple sensors to exhibit correlations that can be exploited for attack detection. The multiple sensors could be measuring the same system state or not. Multiple sensors measuring the same physical state are called *homogeneous sensors* whereas those measuring different physical system states are referred to as *heterogeneous sensors*. We classify publications that exploit correlation that naturally exists among sensors for attack detection purposes into two groups: (1) homogeneous sensors and (2) heterogeneous sensors attack detection.

Homogeneous sensors attack detection

Multiple sensors measuring the same physical phenomenon are expected to have their measurements correlating. When this natural redundancy is not observed, it could be an indication of a possible attack, and this has been the basis for publications in this category. For instance, when four wheel speed sensors are used to monitor the speed of a vehicle's wheel, they should all report similar readings under normal operation.

Researchers in [49] propose a switching algorithm that searches for a combination of sensors that have not been compromised and generates estimates that are insensitive to sparse malicious attacks. The algorithm assumes that some of the redundant sensors have been compromised.

Although this is a good approach to attack detection, it has some limitations. First, it increases the cost of production as multiple sensors of the same type have to be deployed. This leads to increased power consumption. Also, more space will be required to accommodate the multiple sensors leading to increased weight. In applications where a lighter weight is desired, this approach may be impractical. On the other hand, fooling the attack detection may be easier since the same attack strategy and equipment can be used to attack the multiple sensors simultaneously. For example, the attacker may successfully cause all speed sensors to report 5mph thereby preserving the correlation.

Heterogeneous sensors attack detection

The attack detection solutions in this category hinge on the observation that some set of sensors within an autonomous vehicle are correlated in terms of their

readings [25, 31, 37, 58]. Remember that this observation is guided by physical laws. For instance, as a car moves faster, naturally, the wheels spin faster, the engine speed increases, and the pressure applied to the pedals also increases. Therefore, this natural phenomenon causes effects on sensors that monitor the wheel speed, engine speed, and pedal. Given that this natural redundancy holds all the time due to physical laws, a violation of the observation to some degree could be an indication of an attack. Hence, the proposed solutions in this category capture this physical invariant by using various methods that exploit the correlation or the natural redundancy that exists among the different sensors. Generally, the detector raises an alert whenever the natural redundancy no longer holds due to attacks.

The methods used to exploit the correlation are varied including cluster analysis [25], Pearson correlation analysis [82], autoencoders [37], regression [58]. In cluster analysis, researchers first build tools to determine the context and the cluster that the identified context belongs to. This is done for each time window. Then, a pairwise cross-correlation is performed and the results are compared with the expected correlation values for that cluster. The calculated deviation from the cluster's mean correlation value is reported as standard deviation from the mean.

In the regression method, the authors formulate the problem as a machine learning regression problem. The regression model uses statistical processes to estimate the relationships among correlated sensors. The model predicts sensor values which are then compared with the observed sensor values. A deviation is calculated and if it exceeds a threshold, an alert is raised.

EVAD [31] utilizes the frequency domain to detect attacked sensors after Fourier transform. They also organize the correlations of sensors into a ring architecture in order to reduce the computation overhead. EVAD exploits both the time domain and the frequency domain property of sensor data as the criterion to detect anomalies.

Researchers in [75] also considered a system where multiple sensors measure the same physical variable. The solution assumes that some of the redundant sensors are attacked. The work develops a resilient sensor fusion algorithm for attack detection.

Unlike the homogeneous approach discussed above, this approach does not increase the cost of production since no extra sensors are needed. Therefore, the power consumption, space, and weight remain the same for these solutions. Also, this approach tends to be more robust to attack since, to fool the detection and maintain the correlation, the attacker has to launch attacks against multiple types of sensors. Based on the fact that each type of sensor relies on different physical principles to operate, the attacker needs multiple strategies, equipment, and varying proximity to the sensor in order to launch a successful attack simultaneously.

2.1.3 Classification based on techniques employed

Existing works on cyber-physical systems attack detection can be categorized based on techniques that are used.

Redundancy-based: Works that use this approach [24, 75, 100, 101] detect attacks by using multiple system components. The duplicated system component may be software (e.g controller) or hardware (e.g. sensors). The states or outputs of each of the duplicated

system components are cross-checked at runtime. In spite of its effectiveness, this approach leads to increased cost, weight, power, space requirement and system complexity.

Signature-based: The works that use this approach [26, 47] monitor runtime patterns and compare them with a pre-maintained dictionary that contains known attack types or attack patterns. For it to be effective, the dictionary needs to have the latest attack patterns. These methods are known to be fast and have low false positives rates, however, they are not effective in handling zero-day attacks [14, 43]

Behavioral rule-based: Behavioral rule-based techniques [9, 67, 68] models the normal system operations by using a specification. The program state transitions or execution time constraints are usually modeled in this approach.

Physics-based: This approach detects attacks by monitoring the physics of cyber-physical system. It is an area of research that is attracting a lot of attention. [29] surveys works that use this approach for cyber-physical systems in general, whereas [4] surveys works that use this approach specifically for autonomous vehicles. Recently, [14] and [78] applied this technique to detect physical sensor attacks. During the first of its two steps, Physics-Based Attack Detection (PBAD) approaches extract the physical invariant of the system and use it to model the system. Although we do not extract the physical invariant directly, we indirectly use deep learning to learn about them. Like other approaches, in the second step, PBAD compares the model predictions with the observed values and raise alarm when observed states exceed a threshold.

Machine/Deep Learning: Machine Learning (ML) and Deep learning (DL) techniques have been employed in many CPS attack detection works

lately [2, 6, 30, 37, 40, 43, 52, 58, 63, 70, 74, 83]. These solutions build a data model by using the system data to train a machine or deep learning model. The models are often used to make predictions of CPS measurements such as sensors. ML and DL methods require a large amount of data to build accurate models. While supervised ML methods require both labeled normal and attack training data, unsupervised methods often process only normal training data. Our solution uses an unsupervised deep learning approach. Our work is distinguished by the incorporation of attack detection deadline estimation and adaptive attack detection mechanism.

2.2 Attack Recovery Solutions

Attack recovery measures are meant to improve the attack-resilience of the autonomous CPS. That is, measures that enable the CPS to function continuously in spite of the attack. Compared with attack detection solutions, we can say that attack recovery solutions are proactive attack resilience measure.

One of the effective ways of improving attack-resilience is to develop methods that can estimate system states accurate enough for control regardless of the compromised components. This way has the merit of allowing the system to use the same controller as in the case without attacks. [73] and [22] are existing works that follow this approach. Note that these proposed solutions are confined to sensor redundancy setting, i.e., multiple sensors are employed to measure the same physical variables. This limitation also follows that the proposed solutions are only applicable when the number of compromised sensors is within a certain threshold. Hence, dealing with attack resilience in the cases that violate

above limitations is a question that still remains. Kong [51] addresses these limitations by proposing a solution that leverages checkpointing and recovery. That is, instead of using redundant sensors, they use the historical data to recover the system states.

Various techniques have also been employed for state estimation. A popular technique is sensor fusion algorithm. As noted in [41], the meaning or interpretation of the term "sensor fusion" vary across different fields of research. While some consider the term to mean the collection and combination of data from homogeneous redundant sensors, others equate the meaning to the "state estimation" of different sensors that measure different aspects of the system's state. The works that use sensor fusion can be categorized based on the sensor model that is used. The first approach leverages a probabilistic model to compute the expected results. Kalman filter [45] falls in this category where assumptions about sensor precisions are combined with the known system dynamics model to achieve a linear estimation of the true state. Many variations of the Kalman model have span including [18] and [97]. These approached have the goal of achieving average performance of a system and therefore [41] assert that they might not be suitable for low-probability analysis of rare events. The second category of sensor fusion works leverages an abstract sensor model instead for worst-case analysis. A pioneer work of this approach is [65] where an assumption is made that sensors provide one-dimensional intervals. Variations of this model includes [66], [13], [103] and [11]. The third category of sensor fusion works treat sensor measurements a decision. All the sensor decisions are combined in some form of a voting scheme [12, 46].

2.3 Challenges

In this section, we discuss some of the challenges that researchers proposing attack detection methods face. We do not discuss the challenges in any particular order of importance.

2.3.1 Training data for data models

From our discussion above, we see that machine and deep learning techniques are valuable for building attack detection solutions. These tools, however, require enormous training data. The first challenge is that the publicly-available datasets are sparse and they contain no or very few attack datapoints. One of the reasons for this is that, especially for real-life datasets, attacks rarely occurred in the past because vehicles were then closed system [37]. Even with modern-day vehicles that are becoming open systems, successful attacks do not happen often. Hence, with such limited attack scenarios in the dataset, the machine and deep learning models are constrained in learning the attack patterns as expected to build robust models that are able to recognize attacks. In other words, CPS attack monitoring models that are trained with insufficient data tend to respond unfavorably to events or scenarios that they have not been seen before [95]. This data sparsity problem was one of the causes of the 2016 Tesla crash [3].

It is worth mentioning that some proposed methods [37] have responded to this data sparsity challenge by leveraging unsupervised machine/deep learning techniques. The models are trained to learn the nominal behavior of the plant under study from only normal data. Then using the principle of inclusion-exclusion, an alarm is raised whenever

the sensor under scrutiny does not produce data that are indications of normal activity. However, the false positive and false negative rates are not promising for practical applications.

Further, the normal data available are not sufficient since they usually do not contain all the normal behavior scenarios. For instance, during the data collection stage, if the autonomous vehicle does not perform certain activities, maneuvers, or tasks, the data associated with these normal behaviors will not be captured in the dataset. Hence, unsupervised learning techniques/models which only learn from normal data are misled to classify even normal autonomous vehicle activities as abnormal.

Lastly, the sensor data obtained from autonomous vehicles can be corrupted, noisy, faulty, missing, and may contain redundant data [95, 98]. Sensors tend to be sensitive to interference in their environment which can lead to data corruption. In most situations or applications, such interference is inevitable and in others, some measures can be taken to reduce the noise. Data may also be corrupted due to the interactions occurring among system components. Lossy communication channels especially those between the sensor and data collection point contribute to data corruption. Identifying that a dataset is corrupted may require some system expertise and can be challenging. The consequences of building an attack monitor on corrupt data are quite obvious.

2.3.2 Testbeds

The availability or access to rich/practical autonomous vehicle testbed is another challenge that researchers face. In most of the papers reviewed, evaluations are not

performed on systems that mimic the resources that are available on real autonomous vehicles thereby reducing the practicality of the proposed solutions. Rather, experiments are carried out using simulated data that were run on computing resources that differ a lot from resources available on autonomous vehicles. For instance, the operating systems that the experiments are simulated are not a real-time OS. Also, the CPU/GPU capabilities and memory capacity available on experimental systems are higher compared with what is available on autonomous vehicles.

In part, high-end autonomous vehicle testbeds are expensive to acquire, limiting research groups, especially those in developing countries, from testing out their novel ideas and designs. Although cheaper testbeds are available, usually, they do not possess all the sensors that may be required for the particular research. It is also possible to custom-build autonomous vehicle testbeds, however, assembling all the components requires expertise that may not be available in the research group or the university at large. Even in instances where the expertise is available, the process of building the testbed can be time-consuming. From our own experience, it has taken more than a year to build an autonomous vehicle testbed. Further, the sharing of testbeds amongst research groups especially those whose physical geography is farther apart may be hampered by travel restrictions by governments, a pandemic, or other factors.

2.3.3 Benchmark for comparing related work

It is difficult to fairly and accurately compare the effectiveness and efficiency of the various proposed attack detection methods due to the absence of “standardized”

benchmark data. Given that each research effort evaluates their work on the data that the researchers generate or simulate, it is difficult to tell if the proposed solutions are applicable to only their data or work with other new data. A common benchmark can facilitate result comparison as well inspire research proposals that perform better than existing solutions.

Also, many researchers who have access to good testbeds or even simulate good autonomous vehicle data often do not make their data and source code publicly available. Such availability to the public not only aids the repeatability of the research method but also allows others to use the data and compare the results.

2.3.4 Standard evaluation metrics

Another challenge regarding research result comparison is the lack of standard evaluation metrics. Usually, different metrics are used for evaluating the proposed attack detection method. This makes it difficult to know which proposal is better and even how an existing solution should be improved based on a metric. A standard evaluation metric can guide the current as well as the future development of evaluative metrics for attack detection methods in autonomous vehicles. A common metric can also help the peer review process so that reviewers can make a better judgment of papers under review and/or make suggestions that improve research efforts.

3. REAL-TIME ADAPTIVE SENSOR DETECTION

As noted in Chapter 1, cyber-physical systems face many new threats as a result of the deep integration of information technology with physical process. While this fact has motivated many defense mechanisms against sensor attacks, a clear vision on the timing and usability (or the false alarm rate) of attack detection still remains elusive. Existing works tend to pursue an unachievable goal of minimizing the detection delay and false alarm rate at the same time, while there is a clear trade-off between the two metrics. Instead, we argue that attack detection should bias different metrics when a system sits in different states. For example, if the system is close to unsafe states, reducing the detection delay is preferable to lowering the false alarm rate, and vice versa. To achieve this, we make the following contributions.

3.1 Cumulative Sum (CUSUM) based attack detection

3.1.1 Contributions

This dissertation proposes a real-time adaptive sensor attack detection framework. The framework can dynamically adapt the detection delay and false alarm rate so as to meet a detection deadline and improve the usability according to different system status.

The core component of this framework is an attack detector that identifies anomalies based on a CUSUM algorithm through monitoring the cumulative sum of difference (or

residuals) between the nominal (predicted) and observed sensor values. We augment this algorithm with a drift parameter that can govern the detection delay and false alarm. The second component is a behavior predictor that estimates nominal sensor values fed to the core component for calculating the residuals. The predictor uses a deep learning model that is offline extracted from sensor data through leveraging convolutional neural network (CNN) and recurrent neural network (RNN). The model relies on little knowledge of the system (e.g., dynamics), but uncovers and exploits both the local and complex long-term dependencies in multivariate sequential sensor measurements. The third component is a drift adaptor that estimates a detection deadline and then determines the drift parameter fed to the detector component for adjusting the detection delay and false alarms. Finally, we implement the proposed framework and validate it using realistic sensor data of automotive CPS to demonstrate its efficiency and efficacy.

System and Threat Model

The CPS model we consider in this work is a physical system, also called a plant, controlled by a controller. The controller operates at every δ unit of time, where $\delta > 0$ is called a control period. At the beginning of every control period, the controller first reads the output of the plant or sensor measurements. Then using a control algorithm, the controller computes the control signals or inputs that are sent to the actuators. The actuators will apply the control inputs to the plant in the current step.

We consider attack scenarios, where the attacker is able to compromise the integrity and availability of sensor data of autonomous CPS, as shown in Fig. 3.1.

(i) Integrity of Sensor Measurements. The adversary is able to modify the sensor measurements by launching spoofing attacks in the CPS's physical environment such as introducing noise or interference in the signals that the sensor is perceiving. The attacker may also undertake replay attacks to compromise data integrity. A successful replay attack enables an attacker to send previously captured data to the CPS. While the replayed data was valid data at a particular point in the past, it does not reflect the current state of the CPS.

(ii) Availability of Sensor Measurements. The adversary is able to delay the controller from receiving the sensor values. The received values are out-of-date and reflect a historical state of the system. Denial of service (DOS) attacks belong to this kind of attack, where the delay is infinite. Signal jamming is one typical DOS attack that the attacker can execute in the CPS's physical environment.

This work is focused only on sensor the attacks mentioned above. We thus assume that the adversary does not compromise the controller, the actuator, or other cyber components of the system (cyber attacks). We do not restrict the maximum number of sensors that can be compromised by an attacker but assume that the attacker has no knowledge of our attack detector.

Overview of System Design

Fig. 3.1 shows the overview of the proposed adaptive real-time attack detection framework. It has two phases: an offline training phase and an online detection phase.

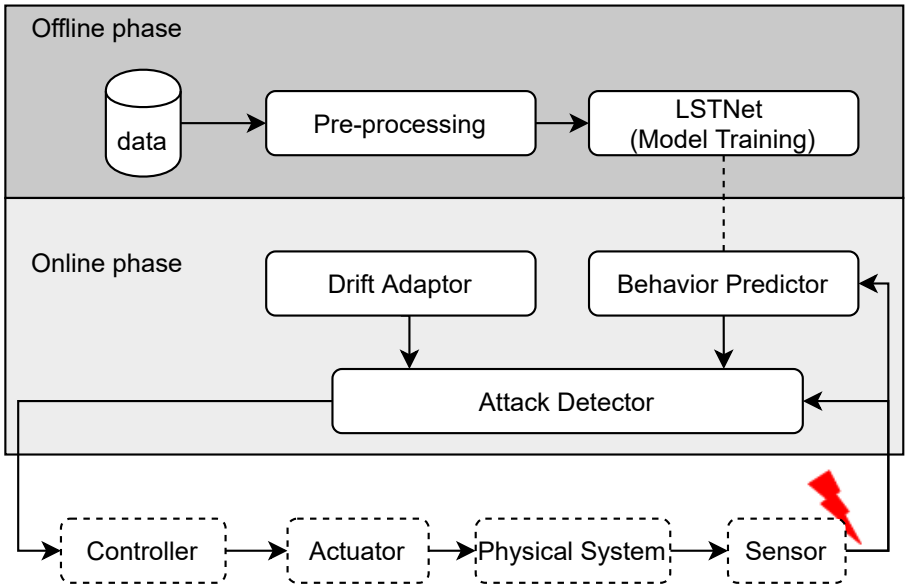


Fig. 3.1.: Design overview of the real-time adaptive sensor attack detection framework.

The offline phase consists of components that function together to learn the nominal behavior of the system through training a deep learning model. It leverages both the local and complex long-term dependencies that exist among sensor data. To achieve this, the pre-processing component first screens out sensors that are correlated with each other by calculating their pairwise correlations. Then, the Long- and Short-Term Time-Series Network (LSTNet) component captures a consistent pattern among the correlated sensors, which is referred to as the nominal behavior.

The online phase handles the real-time attack detection and is made up of three components. The Behavior Predictor uses the learned model to predict nominal sensor values. In the presence of attacks, sensor measurements (observed) will be different from the predicted values. This difference, called the residual, is tracked by the Attack Detector to identify anomalies. It will raise an alarm when the cumulative sum of residuals becomes larger than a pre-defined threshold. The Drift Adaptor ensures a usable detection

result before the detection deadline. The deadline may vary over time as the physical environment changes. This component can dynamically adjust the detection delay to meet the deadline via the drift parameter. To be clear, we state the workflow of the online phase as follows. At each control period, the Behavior Predictor and Drift Adaptor first produce nominal sensor values and the drift value, respectively. Then the Attack Detector uses these values to identify anomalies.

3.1.2 Design of Attack Detector

In this section, we present the detailed design of the core component, Attack Detector, in our framework. This component needs predicted sensor values and the drift parameter from Behavior Predictor and Drift Adaptor respectively. The latter two components will be detailed in the subsequent sections.

Problem Formulation

We formulate the attack detection problem as follows. Given the predicted nominal sensor value $\hat{y}_t \in \mathbb{R}^n$, observed sensor value $y_t \in \mathbb{R}^n$ and the drift parameter λ, the problem is to determine the appropriate time to raise an attack alert t_{alarm} when the observed sensor values deviate from the expected values such that it exceeds a threshold τ:

$$t_{alarm} = \mathcal{C}(y_t, \hat{y}_t, \lambda) > \tau, \tag{3.1}$$

where $\mathcal{C}$ is a change detection mechanism.

Attack Detection

There are two main strategies that can be used to realize Eq. (3.1), that is, to determine the appropriate time to raise alarm: stateless and stateful. (i) In a stateless strategy, it is confined to monitor every single period's residual, and an alarm is raised for every single deviation, that is, if the residual exceeds a pre-determined threshold τ i.e, $r_t > \tau$. This kind of strategy has been shown to have increased false positives [29]. (ii) A stateful strategy, on the other hand, calculates the statistic S_t that keeps track of the historical changes of r_t. It raises alarm when there is a *persistent* deviation over time, i.e. $S_t > \tau$. This kind of strategy has been demonstrated to have decreased false positives [29].

We thus choose to develop a stateful strategy in our framework due to its lower false positive rates. There are usually two kinds of stateful strategies: time window and cumulative sum (CUSUM). (i) In a time-window-based method, the detector looks at the residuals within a time window of multiple control periods. (ii) A CUSUM-based method, on the other hand, efficiently tracks the cumulative sum of residuals of the whole history. The authors in [78] demonstrate that a CUSUM-based approach tends to be faster and more accurate than a time-window-based approach. Further, the former is more robust to attacks that are hard to be detected by other approaches such as attacks hidden in-between time windows and other stealthy attacks.

Hence, we present a CUSUM-based attack detection approach. The algorithm is augmented with a drift parameter, by tuning which the detection delay and false alarms can be changed. The algorithm outline is shown in Algorithm 1. We briefly explain the algorithm as follows.

Algorithm 1: The CUSUM Algorithm.

Input: threshold τ, drift λ, observed sensor value y_t, predicted sensor value $\hat{y}_t$
Output: alarm time t_{alarm}

1 **Initialize:** $S_0 = 0$;
2 **while** $t > 0$ **do**
3 $r_t = y_t - \hat{y}_t$; // the residual of control period t.
4 $S_t = [S_{t-1} + |r_t| - \lambda]^+$; // the cumulative sum;
5 // $[a]^+ = max\{a, 0\}$.
6 **if** $S_t > \tau$ **then**
7 $t_{alarm} = t$;
8 $S_t = 0$;
9 **end**
10 // the cumulative sum is greater than the threshold; at period t an alarm is raised; reset the sum.
11 $t = t + 1$
12 **end**

Line 1 initializes the cumulative sum to zero. Line 2 calculates the residual between the observed sensor value y_t and the predicted sensor value $\hat{y}_t$ obtained from the Behavior Predictor. That is, this difference indicates how deviated the observed value is from the nominal estimate. Line 3 calculates the cumulative sum S_t at control period t, which is a non-negative value. Basically, it equals the cumulative sum at period $t - 1$ plus the absolute value of the residual at t minus the drift parameter. The drift parameter is decided by the Drift Adaptor. As mentioned, selecting the appropriate drift parameter is an important aspect of the algorithm. It can impact both the detection delay and the number of false positives. Line 4-7 checks if the cumulative sum is larger than the pre-defined threshold. If yes, an alert t_{alarm} is raised, and S_t is reset to zero.

3.1.3 Design of Behavior Predictor

In this section, we present the detailed design of behavior predictor. This component builds a data model of the system that captures physical invariants for the purpose of predicting sensor measurements.

Physical invariants are properties of the physical system that should always hold. The invariants are guarded by physical laws. One method to capture physical invariants is to use a physical system model. One disadvantage of this method is the requirement of adequate knowledge of accurate system dynamics, which may not be easy to attain.

In this work, we approximate physical invariants using a deep learning technique instead. The approximated physical invariant will be used as the nominal behavior of the system. This technique treats the system as a black box and explores the correlation of multivariate sensor data. Our insight is that if the system operates normally and obeys physical laws, then the sensor data obtained from the CPS also indirectly obeys physical laws. Hence, with little knowledge of the system dynamics, our deep learning approach enables us to learn the behavior of the system in order to make accurate predictions.

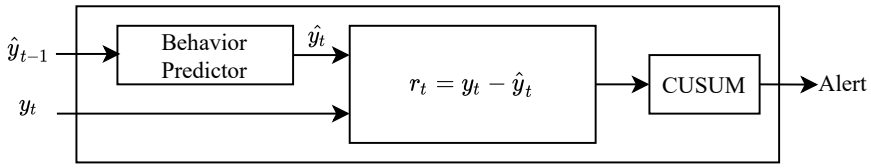


Fig. 3.2.: Dataflow in attack detector.

Problem Formulation

In order to perform the non-trivial task of predicting nominal system behavior, we formulate the problem as a multivariate time series forecasting problem.

Given a fully observable system with n correlated sensors $Y = \{y_1, y_2, ..., y_T\}$ where $y_t \in \mathbb{R}^n$, we want to extract the natural redundancy that exists among the correlated sensors using a deep learning model $\mathcal{D}$, so that we can learn the nominal behavior of the system such that we can predict future sensor values $\hat{y}_{T+1}$. It is assumed that $\{y_1, y_2, ..., y_T\}$ will always be available whenever we predict $\hat{y}_{T+1}$. The input to the behavior predictor at time step T is formulated as $X_T = \{y_1, y_2, ..., y_T\} \in \mathbb{R}^{n \times T}$

$$\hat{y}_{T+1} = \mathcal{D}(X_T) \tag{3.2}$$

Pre-processing

Sensors on automotive CPS exhibit physical sensor correlation or natural redundancy [37]. We need to ensure the DL model is trained using only sensor data that are correlated. This component uses a statistical method to observe the natural redundancy in the dataset and also finds sensor data that are correlated but may not be obvious from domain knowledge.

The pre-processing component builds a correlation matrix based on Pearson's Correlation Coefficient (PCC) algorithm. Data variables or features are said to have a positive correlation when both variables move in tandem. That is, if one variable increases, the other variable also increases. A positive correlation also holds when one

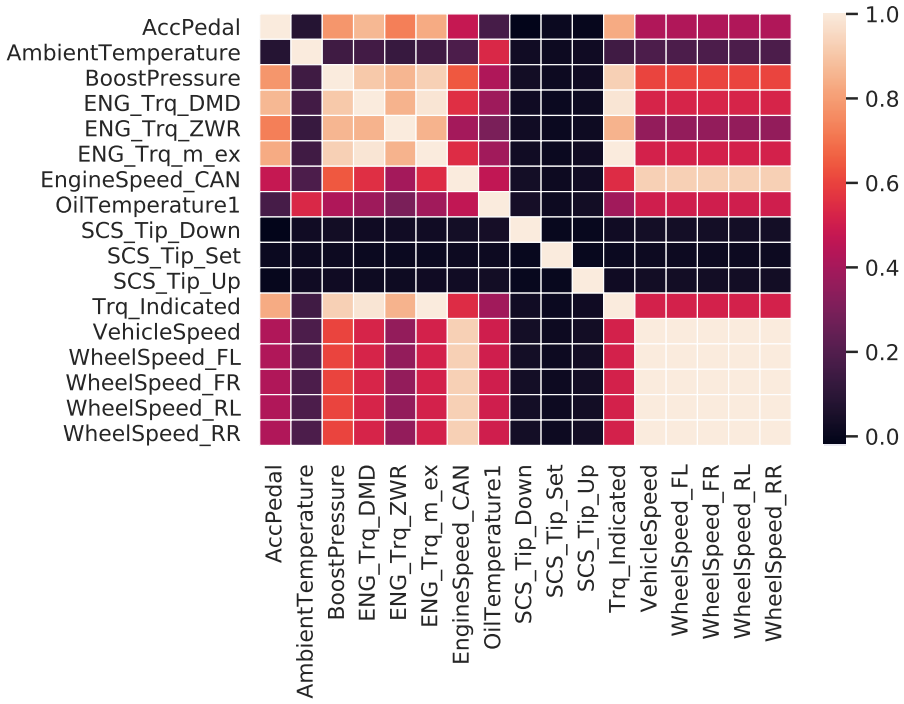


Fig. 3.3.: Example confirming the wheel speed sensors in the dataset has strong correlation with the wheel speed, engine speed and boost pressure sensors.Table 3.1 shows the available sensors in the dataset.

variable decreases and the other variable decreases as well. Conversely, two variables have a negative correlation when one increases and the other variable decreases, and vice versa. PCC indicates a strong positive correlation with coefficient values that are close to +1.0 whereas a strong negative correlation has coefficients that are close to -1.0. Coefficient values close to 0 signifies that the two variables do not have any correlation. We select dataset features whose PCC values are either greater than 0.5 or less than -0.5 as input to model training. For example, to observe the sensors that have natural redundancy with the wheel speed sensor in the AEGIS CAN dataset (we describe this dataset in section 3.1.5), we created the heatmap shown in Fig. 3.3 based on the PCC values. In the

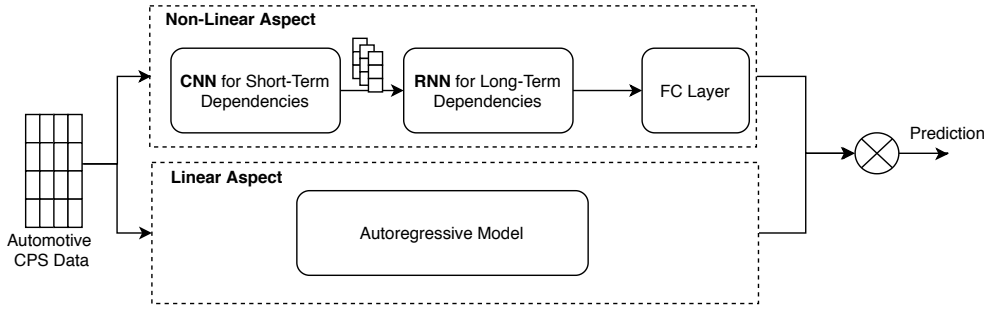


Fig. 3.4.: Architecture of LSTNet model that learns both local and complex long-term dependencies in automotive CPS sensor values for attack detection.

figure, we observe the wheel speed sensors have a strong positive correlation with vehicle speed, engine speed, boost pressure, engine torque and oil temperature sensors.

Long- and Short-Term Time-Series Network (LSTNet)

Fig. 3.4 is an overview of the deep learning architecture used which is based on [54]. The interested reader is referred to [54] for details, here, we briefly describe each component. Mainly, the architecture consists of a convolutional neural network (CNN), a recurrent neural networks (RNN) as well as an autoregressive linear model.

CNN Component. The first layer of the deep learning framework is a CNN without pooling. It is tasked to extract the temporal patterns and the local relationship between the correlated sensor variables. This CNN layer is made up of a number of filters of width w and height n (the number of correlated sensor variables) with each k-th filter passing through the input matrix X to output a vector h_k:

$$h_k = RELU(W_k * X + b_k) \tag{3.3}$$

where $*$ is the convolution operation, W_k and b_k denote the weight parameter and bias respectively. $RELU$ activation function ensures values stay between 0 and 1. Each vector h_k is zero-padded on the left of the input matrix X to have a length of T. In the end, the convolutional layer outputs a matrix of size $d_c \times T$, where d_c is the number of filters. This output matrix is inputted into the recurrent component.

Recurrent Component. The recurrent component has two sub-components namely, gated recurrent unit (GRU) and recurrent-skip.

GRU is a specialized recurrent neural network (RNN) that is suited for modeling sequential data such as sensor readings [53]. Unlike artificial neural networks (ANN), GRU is able to store past information in addition to current inputs in order to determine current outputs. The ability to store past information in GRU is enabled by the state variables that it introduces i.e. the update and reset gates. At a time t, given the input minibatch $x_t \in \mathbb{R}^{m \times l}$ (where m is the number of examples in the minibatch and l is the number of inputs) and the previous hidden state $h_{t-1} \in \mathbb{R}^{m \times s}$ (where s is the number of hidden states), the reset gate $z_t \in \mathbb{R}^{m \times s}$ and update gate $u_t \in \mathbb{R}^{m \times s}$, candidate hidden state c_t and final state h_t are computed as,

$$\begin{aligned} z_t &= \sigma(x_t W_{xr} + h_{t-1} W_{hr} + b_r) \\ u_t &= \sigma(x_t W_{xu} + h_{t-1} W_{hu} + b_u) \\ c_t &= RELU(x_t W_{xc} + r_t \odot (h_{t-1} W_{hc}) + b_c) \\ h_t &= (1 - u_t) \odot c_t + u_t \odot h_{t-1} \end{aligned} \tag{3.4}$$

where $\odot$ is the element-wise (Hadamard) product, σ is the sigmoid function, W_{xr}, W_{xu}, W_{xc}, W_{hr}, W_{hu}, W_{hc} are the weight parameters, and b_r, b_u, b_c are bias parameters.

The output of the GRU layer is the hidden state h_t at each time step. Note that the use of GRU in the recurrent component allows the deep learning model to discard irrelevant previous sensor information and extract only the important ones that help to learn the nominal system behavior.

The second sub-component of the recurrent component is the recurrent skip component. This feature enables the architecture to memorize the repeated historic periodic pattern (such as daily, weekly patterns) in time series data. However, since the automotive CPS sensor data do not exhibit this periodic pattern, we do not turn it on in our experiment.

The output of the recurrent component is passed to a fully connected (FC) layer as shown in Fig. 3.4. FC combines its input to make a prediction result h_t^D is at time step t.

Autoregressive Component. This component addresses a deficiency found in the non-linear neural network components: convolutional and recurrent components. The scale of output in neural networks is known to be insensitive to the scale of its inputs [54]. Hence, given the non-periodic nature of sensor data, this deficiency diminishes the forecasting accuracy of the neural networks. This is solved by decomposing the final prediction into a linear component by using an autoregressive (AR) model which is formulated as,

$$h_t^L = \sum_{k=0}^{q^{ar}-1} W_k^{ar} v_{t-k} + b^{ar} \tag{3.5}$$

where $h_t^L \in \mathbb{R}^n$ is the forecasting result of the AR component, $W^{ar} \in \mathbb{R}^{q^{ar}}$ and $b^{ar} \in \mathbb{R}$ are the coefficients of the AR model such that q^{ar} is the size of input window over the input matrix. v_{t-k} is the past series values (lagged values).

At time step t, the DL model makes a prediction $\hat{y}_t$ by integrating the outputs of the neural network part and the AR component:

$$\hat{y}_t = h_t^D + h_t^L \tag{3.6}$$

Objective function. We use absolute loss (L1-loss) as the objective function which is formulated as:

$$\min_{\Theta} \sum_{t \in \Omega_{Train}} \sum_{i=0}^{n-1} |y_{t,i} - \hat{y}_{t,i}| \tag{3.7}$$

where Θ denotes the parameter set of our model, Ω_{Train} is the set of time stamps used for training.

Although squared error function is an option often used, experiment results in [54] indicate the absolute loss function is more robust.

3.1.4 Design of Drift Adaptor

In this section, we present the design of the drift adaptor. This component ensures attack detection occurs before a detection deadline.

The requisite detection deadline for an autonomous CPS varies with its physical environment. In other words, the deadline by which the attack has to be detected depends on the physical environment. The deadline can change as the physical environment varies.

For instance, the deadline for detecting a wheel speed attack of a vehicle that is 50m away from an object it can crash into will be different from the situation where the crashing object is 200m away. Hence, there is a need for real-time attack detection that adapts its mechanism based on the physical environment or how the system is close to unsafe states, such that the detection delay will be less than the required detection deadline.

Another motivation is the trade-off between detection delay and false alarms in our experiment. The attack detector discussed above (in Section 3.1.2) is augmented with a drift parameter λ that can be adjusted to produce varying detection delays and false positives. Fig. 3.5 and Fig. 3.6 show how the drift parameter affects the detection delay and number of false positives. We note that as the drift parameter increases, the time to detection or detection delay increases while the number of false positives decreases. Hence, adjusting the drift parameter enables our attack detection mechanism to adapt its behavior for an appropriate trade-off while meeting the real-time constraint.

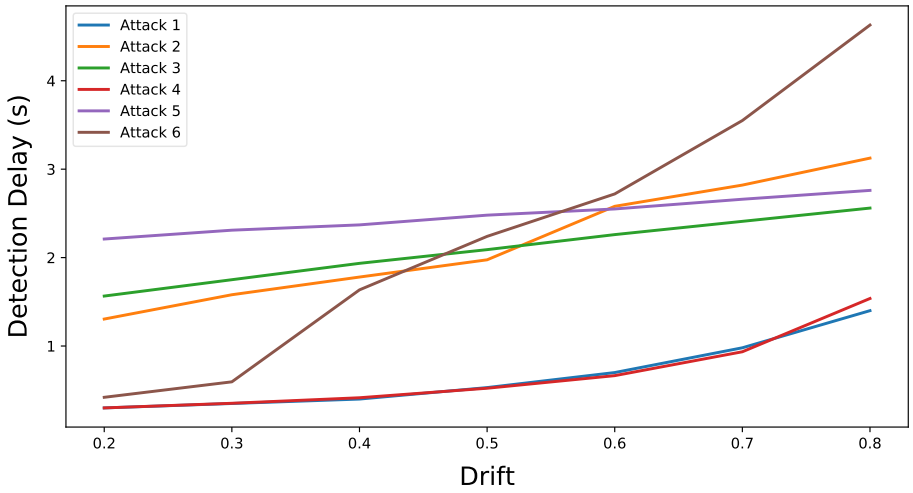


Fig. 3.5.: The relationship between drift parameter and detection delay for various attack scenarios.

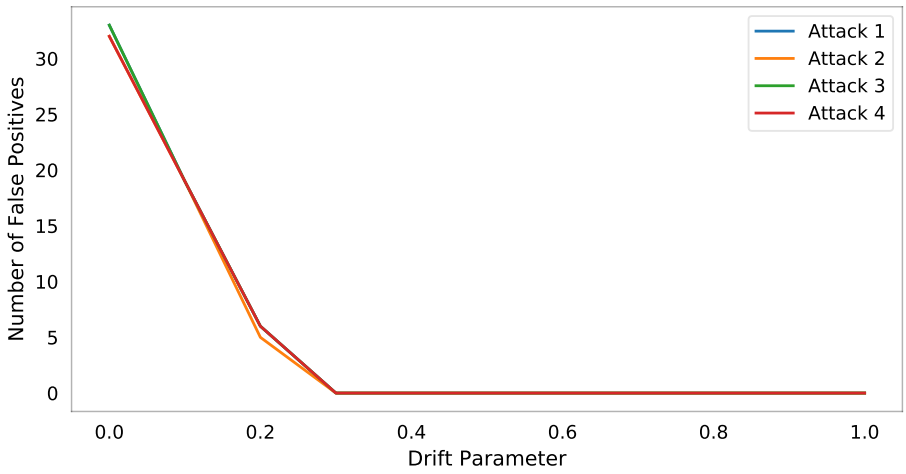


Fig. 3.6.: Relationship between drift parameter and number of false positives.

The Drift Adaptor component is made up of two sub-components: Deadline Estimator and Drift Analyzer. The deadline estimator determines the detection deadline whereas the drift analyzer determines the appropriate drift parameter.

Deadline Estimator

The detection deadline considered in this work is the time in the future when the system may touch the unsafe set. We consider a time that is estimated in a conservative way, i.e., at a worst case. The authors of [102] propose a reachability-based deadline estimation method, but it requires knowing the system dynamics. By contrast, we propose a pure data-driven method towards this end.

The core idea of the proposed method is to first calculate the maximum change rate of the sensor value and then use it to estimate the shortest time when the system may touch the unsafe set. The proposed method has two phases: offline and online.

(i) At the offline phase, we consider the collected time series of each individual sensor i, denoted as $\{y_1(i), y_2(i), ...y_T(i)\}$. The change rate of the sensor value of two adjacent periods is defined as

$$\Delta_t(i) = \frac{y_t(i) - y_{t-1}(i)}{\delta}. \tag{3.8}$$

Then using the collected time series, we use the following equations to calculate the maximum (Δ^+) and minimum (Δ^-) change rate.

$$\begin{aligned} \Delta^+(i) &= [max\{\Delta_t(i), 2 \leq t \leq T\}]^+, \\ \Delta^-(i) &= [min\{\Delta_t(i), 2 \leq t \leq T\}]^-, \end{aligned} \tag{3.9}$$

where $[a]^+ = max\{a, 0\}$ and $[a]^- = min\{a, 0\}$.

(ii) At the online phase, based on the fastest change rate given in Eq. (3.9), we can perform the following reachability analysis to estimate the detection deadline. At current time t, we calculate the reachable value for each sensor by

$$\begin{aligned} y_d^+(i) &= y_t(i) \times (1 + \Delta^+(i) \times \delta \times (d - t)), d > t, \\ y_d^-(i) &= y_t(i) \times (1 + \Delta^-(i) \times \delta \times (d - t)), d > t. \end{aligned} \tag{3.10}$$

The earliest time $D(i)$ when the value of sensor i may touch the unsafe set is

$$D(i) = min\{d | y_d^+(i) \in U(i) \vee y_d^-(i) \in U(i)\}, \tag{3.11}$$

where $U(i)$ is the unsafe set associated with sensor i. Finally, the detection deadline D is calculated by

$$D = min\{D(i)|1 \leq i \leq n\}. \tag{3.12}$$

Note that our framework does not rely on any specific deadline estimation method, and is always applicable as long as a detection deadline is outputted.

Drift Analyzer

With a detection deadline D as input, the Drift Analyzer determines the best drift parameter that allows the attack to be detected before the deadline. For this component to function properly, we need to first establish the relationship between the detection delay and the drift parameter. This is achieved by performing offline profiling. Fig. 3.5 and Fig. 3.6 depict that there is a relationship among the drift parameter, detection delay and false positives. Armed with this information and the CUSUM tuning tools provided in [69], we are able to build a drift-parameter-detection delay pair that ensures we do not exceed the acceptable false positive rate. In other words, We build a lookup table based on the offline profiling results. To perform its online adaptation functionality, the Drift Analyzer simply queries the lookup table to output the drift parameter that adjusts the detection delay to meet the given detection deadline.

3.1.5 Evaluation

Implementation and Experimental Setup

We implemented our deep learning model in Python, utilizing PyTorch Deep Learning framework. We train the model on Ubuntu 18.04 64-bit with sixteen Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz CPUs, two Nvidia GeForce GTX 1080 GPUs and 64 GB RAM. We follow a 60/20/20 proportions for splitting the original dataset into training/validation/test sets. The experimental model is made up of 100 hidden CNN layers and 100 hidden RNN layers. The model was trained for 100 epochs. Metrics used for the test accuracy were Root Relative Square Error (RSE) and Relative Absolute Error (RAE). The accuracy for our experimental model was 0.0032 (RSE) and 0.0018 (RAE).

Dataset Description

We used the publicly-available real-world automotive CAN bus dataset from the AEGIS Big Data Project [44] [1] for our experiment. The sensor data, sampled at 20Hz, was collected during trips in the same passenger vehicle. More than 40 sensor measurements were collected including but not limited to those listed in Table 3.1. Specifically, the data contains about 2.5 hours of driving data (about 160,000 data points).

[1] https://zenodo.org/record/3267184#.X5YtpIhKg2x

Table 3.1: Some sensors in the dataset used in experiment.

CAN bus Sensors	**GPS Sensors**	**IMU Sensors**
ASR	Acceleration	Accelerometer_X
AccPedal	Current_sec	Accelerometer_Y
AirIntakeTemperature	Direction	Accelerometer_Z
AmbientTemperature	Distance	Body_acceleration_X
BoostPressure	Velocity	Body_acceleration_Y
BrkVoltage		Body_acceleration_Z
EngineSpeed_CAN		G_force
EngineTemperature		Magnetometer_X
Kickdown		Magnetometer_Y
MFS_Tip_Down		Magnetometer_Z
MFS_Tip_Up		Velocity_X
SteerAngle		Velocity_Y
Trq_FrictionLoss		Velocity_Z
Trq_Indicated		
VehicleSpeed		
WheelSpeed_FL		
WheelSpeed_FR		
WheelSpeed_RL		
WheelSpeed_RR		
Yawrate		

ASR = Acceleration Slip Regulation, ACC = Acceleration,
BRK = Break, MFS = Misfiring System, TRQ = Torque,
FL = Front Left, FR = Front Right, RL = Rear Left,
RR = Rear Right, G = Gravity

Attacks

The dataset does not include any anomalous events or scenarios, hence we manually modify portions of the dataset to simulate physical attacks that achieve similar goals of a real attacker. Based on the attacks discussed in section 3.1.1 and the attacks in [58], we evaluate our work under (1) modification, (2) delay and (3) replay attacks.

We simulate four attack scenarios under modification attacks and one each for the delay and replay attacks. **Attack 1** adds a fixed value to the sensor readings for a period of time. This simulates the attacker spoofing the sensor measurement as done in Fake Data Injection (FDI) attacks. **Attack 2** sets the sensor reading to a fixed value indicating a spoofing attack that spoofs sensor readings to a specific value. **Attack 3** incrementally changes sensor measurement. Here the attacker has a target value to spoof the sensor, yet he does not set the value right away. Rather, he gradually adds small values (e.g 0.01 kph) to current sensor readings until the target spoofing value is reached. In real life, an attacker might use this strategy with the intent of evading detection mechanisms. **Attack 4** sends both normal sensor values and malicious/fake sensor values alternately and repeatedly. Like Attack 3, a real attack might use this tactic to mislead attack detection mechanisms.

Attack 5 mimics delay attack where the attacker causes a 10s delay in sensor data transmission. **Attack 6** is a replay attack where the sensor data from a previous time are replayed.

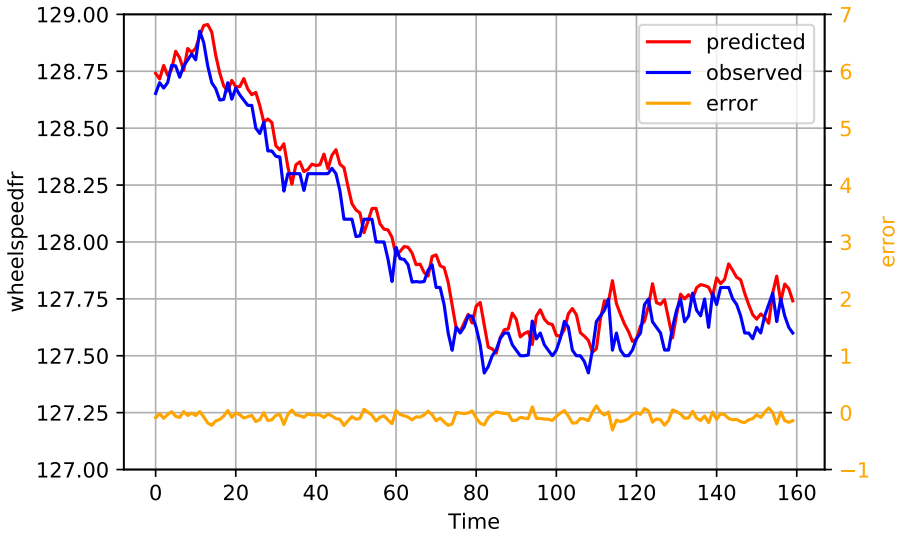


Fig. 3.7.: Front-Left wheel speed sensor measurement showing the predicted and observed values.

Experiments and Results

We perform various experiments to evaluate the effectiveness and efficiency of our proposed framework.

Experiment I: The first experiment tests if the behavior predictor component was able to capture the behavior of the automotive CPS accurately. Fig. 3.7 shows the normal behavior of the front-left wheel speed sensor. It can be seen in the figure that the behavior predictor's prediction closely matches the observed sensor measurement operating under normal conditions. A similar plot for the engine speed, oil temperature and boost pressure sensors shown in Fig. 3.8 further indicate the behavior predictor is able to capture the system's nominal behavior. The error shown in Fig.3.7 has a close to zero-average, an indication that the predictor method is not biased. Further residual analysis shows the

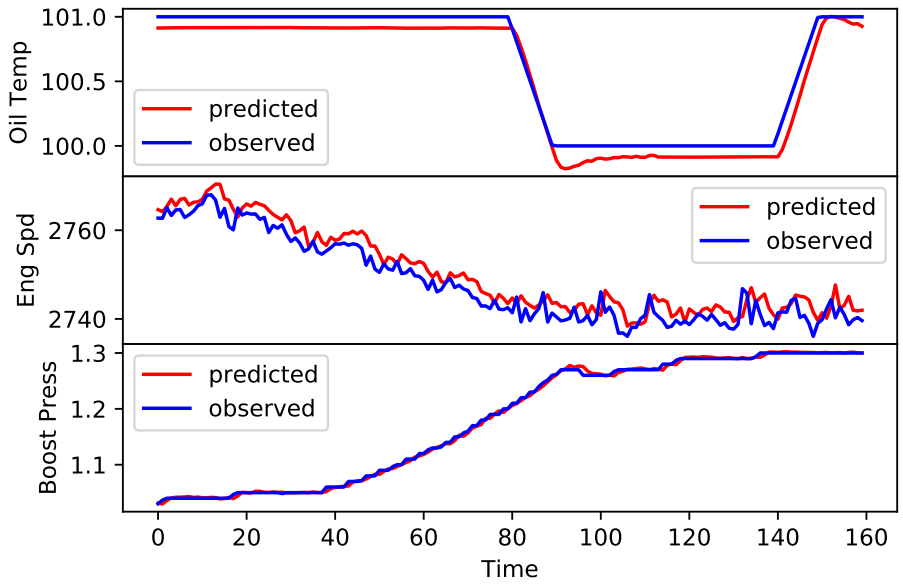
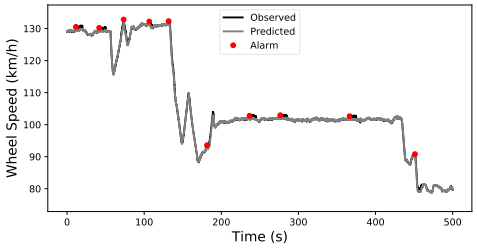


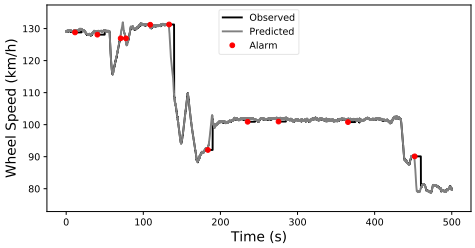
Fig. 3.8.: Observed and predicted values for the oil temperature, engine speed and boost pressure sensors.

residuals follow a normal Gaussian distribution and do not have any trend, cyclic or seasonal structure in the error plot.

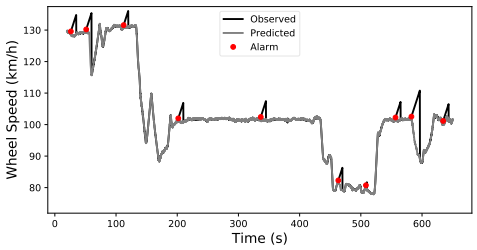
Experiment II: We evaluate the attack detector component in this experiment. The component is tasked to detect the attack scenarios described in 3.1.5. We randomly placed 10 simulated attacks in each case. Fig. 3.9 shows the proposed framework is effective in detecting various attack scenarios. In the figure, the red dots indicate the points where the CUSUM-based attack detector raises an alert for the attacks. It can also be observed that the detector raises alarm only when the abrupt change is significant and has persisted for a while thus reducing flagging transients faults as attacks. For instance, a close-up look at one of the attack points (see Fig. 3.10), shows the detector observed an abrupt change in speed at time 39.95s (indicated by the green arrow) but did not raise alarm immediately until 42.1s.



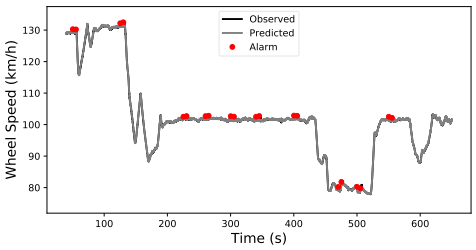
(a) Attack 1.



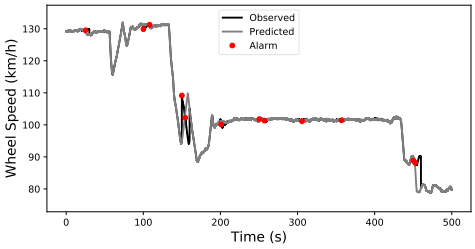
(b) Attack 2



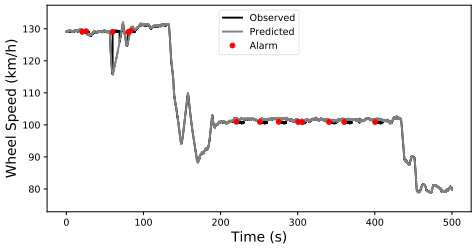
(c) Attack 3



(d) Attack 4



(e) Attack 5



(f) Attack 6

Fig. 3.9.: Results of the attack detector's detection of various attack scenarios discussed in section 3.1.5

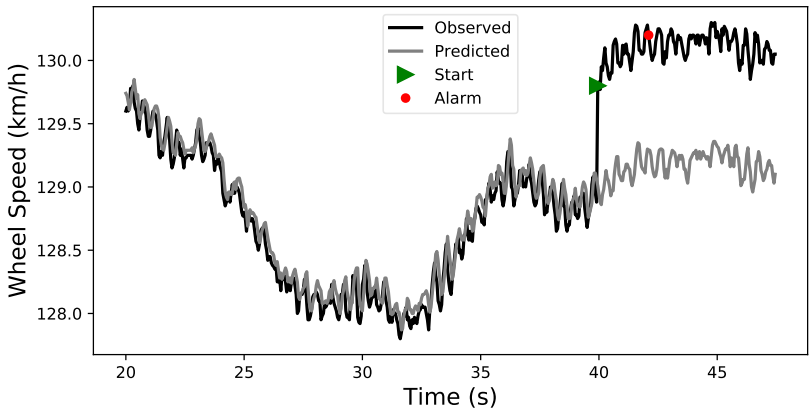


Fig. 3.10.: A close-up look at one case of attack 1 detection.

Experiment III: In this experiment, we measure and analyze the false-positive (FP) and false-negative (FN) rates under various drift and threshold monitoring parameters. FP is measured by inputting normal data (no attack) into the framework and counting the number of false alarms rate. We measure FN by inputting data containing simulated attacks and counting the number of attacks that the framework missed raising an alert for. Note here that *miss* means that an alert was not raised within the duration of the attack. Therefore, alarms that occurred shortly after the attack ended were not counted. The results in Fig. 3.11 and Fig. 3.12 show the FP and FN rates respectively for this experiment. In the legend of the figures, A1, A2...A6 represent the various attack scenarios discussed above, the numbers (3, 4, 5) represent the threshold monitoring parameter. For example, A1-3 represents Attack 1 being monitored with a threshold of 3. In FP analysis, we observe that the CUSUM drift parameter with value 0 produces very high FP rates. However, the FP rate plummets with drift values greater than 0. The FN rate results, on the other hand, show the drift parameter ranging from 0.2 to 0.8 produce zero rates for all attack case scenarios. This implies that for most attacks, the behavior of the detector framework can be adapted to meet attack deadlines whilst still maintaining very low FN rates. However, beyond that range (0.2 - 0.8), we observe the FN increases.

Experiment IV: This experiment measures the detection delay φ i.e. the time it takes to detect the attack after its launch. This metric allows us to evaluate the time needed for our attack detection framework to disclose or alert an attack. If an attack starts at time k_s, and the attack detection mechanism detects it at time k_d, φ is defined as: $\varphi = k_d - k_s$. The lower the value of φ, the better the attack detection mechanism and as such, reduces the impact of the attack.

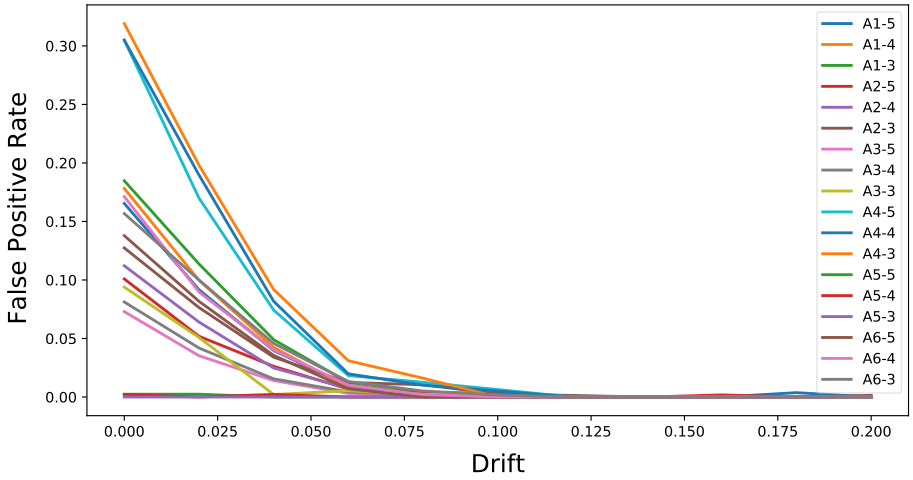


Fig. 3.11.: False positive rate for the various attack scenarios under different monitoring parameters (drift and threshold).

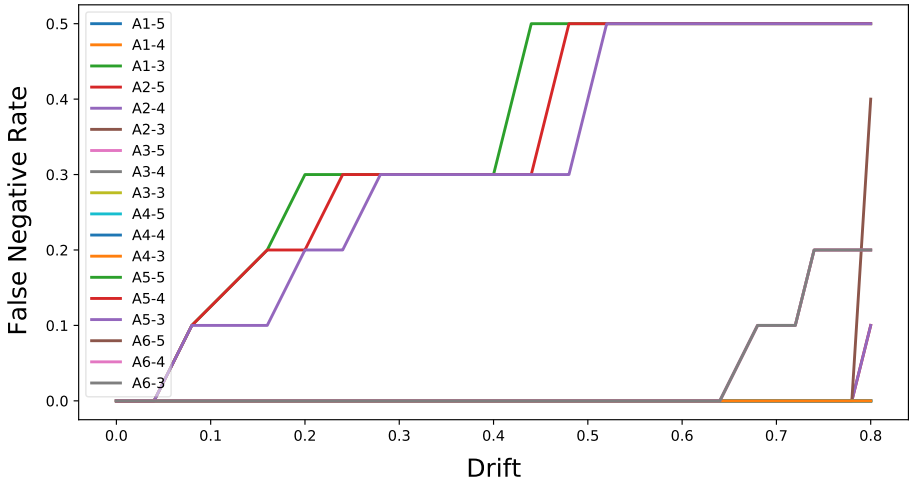


Fig. 3.12.: False negative rate for the various attack scenarios under different monitoring parameters (drift and threshold).

Table 3.2: Detection delay in seconds for the attack scenarios. A1 refers to Attack 1, A2 refers to Attack 2 and so on.

Drift	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**
0.2	0.3	1.31	1.57	0.30	2.21	0.42
0.3	0.35	1.58	1.75	0.35	2.31	0.60
0.4	0.4	1.78	1.94	0.42	2.37	1.64
0.5	0.53	1.98	2.09	0.52	2.48	2.24
0.6	0.7	2.58	2.26	0.67	2.55	2.72
0.7	0.98	2.82	2.41	0.94	2.66	3.55
0.8	1.40	3.13	2.56	1.54	2.76	4.63

Table 3.2 shows the results of the detection delay φ for the various simulated attacks (see section 3.1.5). Note that the simulated attacks lasted for 10s. The result in the table suggests the attacks were detected while the system was being attacked rather than after the attack ended. While this is desirable, we show in a subsequent experiment that it is more desirable and important for real-time systems to detect an attack *before a detection deadline*. Further, the result shows the relationship between the CUSUM drift parameter and the detection delay. The detection delay increases as the drift parameter increases.

Experiment V: This experiment analyses the effect of adaptive detection. For real-time systems, it is not only desirable but required that the attack detection mechanisms are able to detect attacks before the detection deadline D, $0 \leq \varphi \leq D$. Fig. 3.13 shows how the real-time adaptive detection enables Attack 3 to be detected under different deadlines. In the figure, we observe that, in order to meet Deadline 1 (1sec), the drift parameter has to be adjusted to a value not greater than 0.25. Though Attack 3 can be detected with a drift parameter of say 0.8, it cannot satisfy Deadline 1 because its corresponding detection delay is 1.8 seconds.

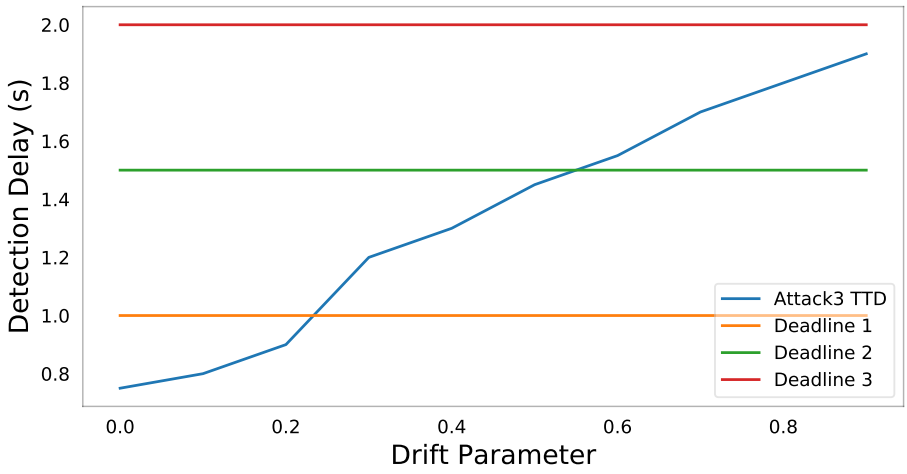
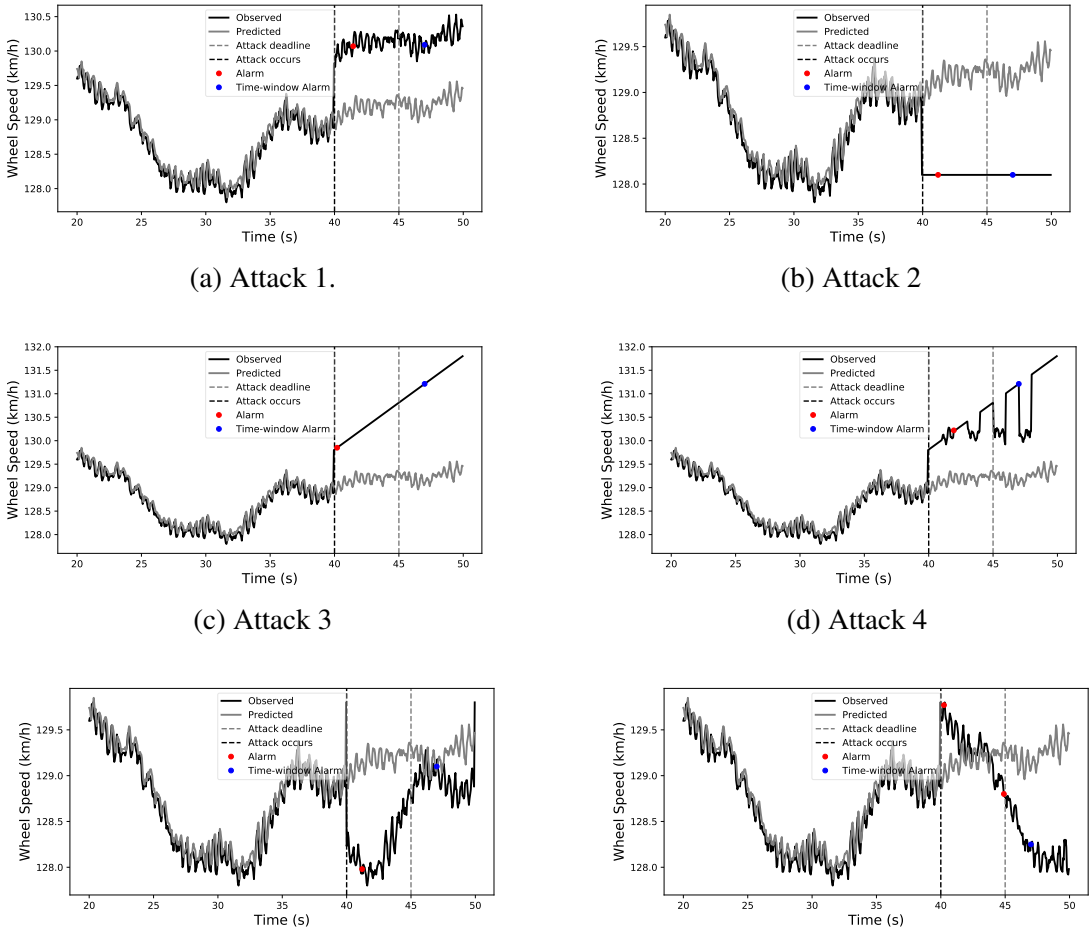


Fig. 3.13.: Adaptive detection for Attack 3.

Experiment VI: This experiment compares a fixed time-window approach with our real-time adaptive attack detection approach. The goal is to show how our framework adapts it behavior based on the drift parameter in order to meet an attack deadline.

The time-window implementation used in this experiment is similar to [14]'s attack detector monitoring algorithm. It sums up the square errors between the observed and the prediction. When the time window expires, it determines if the accumulated mean square exceeds a threshold. The accumulated sum is reset when the time window expires. Note that the time-window approach can only raise an alarm after its time-window expiration.

We compare the two approaches based on the attacks described in Section 3.1.5 and the case scenario depicted in Fig. 3.14. In this case scenario, an attack occurs at $40s$ with a detection deadline estimated at $45s$. The red dots in the figure represent the alarm raised by our framework whereas the blue dot refers to the alarm raised by the time-window attack detector approach. We observe that the time-window approach rightly determines that an attack occurred in all cases, however, the alert is raised after the detection deadline.



(a) Attack 1.

(b) Attack 2

(c) Attack 3

(d) Attack 4

(e) Attack 5

(f) Attack 6

Fig. 3.14.: Comparing our framework with a fixed time-window approach. In all attack scenarios (see 3.1.5), the attack occurs at 10s and has a deadline set at 45s.

In a real-world situation, this would mean the attack detection alert is raised after the damage has occurred. Our framework, on the other, raises an alarm before the deadline.

Experiment VII: We compare our work with a recent anomaly detector that closely relates to our work [37]. Like our work, the researchers exploit the natural redundancy that exists among heterogeneous sensors and they also employ deep learning techniques (deep autoencoder) to detect attacks. Whereas they focus only on the rightness of attack detection, we focus on detecting attacks before a detection deadline by adapting the attack

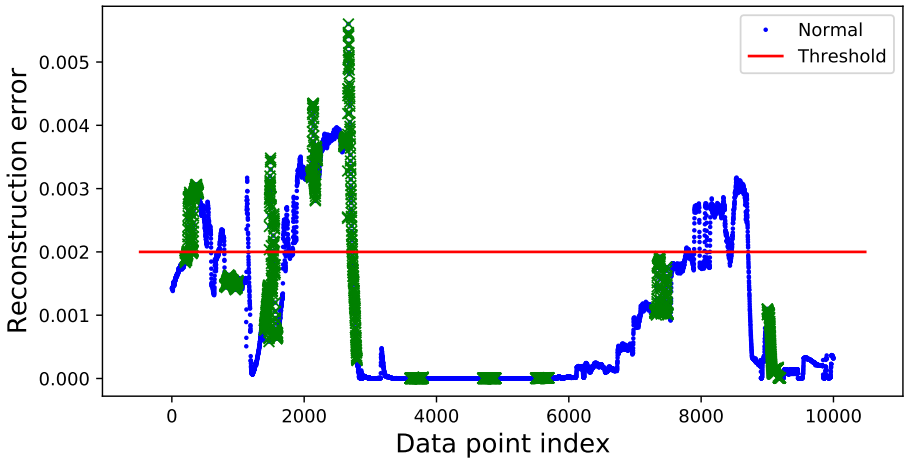


Fig. 3.15.: Attack detection in deep automated [37] attack detector.

detection mechanism. More importantly, in order to decide to raise an attack alert, their approach monitors only one control period. Due to the adaptive nature of our approach to meet a deadline, the number of control periods that it monitors varies. Their detection mechanism raises an alert when the reconstruction error of the decoder is above a certain threshold. We trained a model based on [37] and tuned the hyper-parameters with our dataset for a fair comparison. The deep autoencoder attack detector is tasked to detect the same attack scenarios we subjected our approach to in Experiment II. Fig. 3.15 shows the results. The green marks are the data points that represent the attack. Comparing this results with our results shown in Fig. 3.9, the detector in [37] produces high false alarms.

3.1.6 Discussion

Stealthy Attacks

While our proposed system effectively detects physical attacks against automotive CPS we do not rule out completely the possibility of it being vulnerable to stealthy

attacks. In this attack, the attacker spoofs sensor values that do not exceed the determined threshold, hence the attack detection system raises no alarm. Gradually, the attacker is able to deviate the CPS to his desired target. References [78] and [91] note that this weakness is also found in physics-based attack detection (PBAD) systems. In the real world, a stealthy attack is hard to launch as it requires very detailed knowledge about the system dynamics and ensuring that all the laws of physics are obeyed [14]. On one hand, our proposed detection framework provides some defense as the Behavior Predictor component learns the system dynamics from multiple heterogeneous sensors. To evade our framework, the attacker may have to launch spoofing attacks against all the heterogeneous sensors simultaneously such that it maintains the natural correlation among the sensors that the proposed framework also learned. Achieving such a sophisticated attack in the real world is hard since each sensor attack requires specific tools and equipment to successfully launch. On the other hand, we agree with [91] that a combination of detection schemes can also be implemented to mitigate stealthy attacks. We suggest combining a deep learning approach like our work with PBAD approaches can be a viable solution against stealthy attacks.

State Estimation and Attack Response

This work has focused on physical sensor attack detection without attack response. Once our framework detects an attack, the behavior predictor can also be used to predict values that can be used for state estimation. The state estimation can be forwarded to the controller for recovery control. In the next chapter, we discuss our proposed framework

for attack response that recovers the CPS from an attack so that continual functioning is attained.

3.2 Variable Window-based attack detection

We have shown in preceding chapters that the deep intertwinement of software and hardware has increased the attack surface of systems that once had closed architectures. The research community has responded with solutions that allow physical attacks to be detected. The main idea behind these detectors is the use of various techniques such as Kalman-filter, machine or deep learning techniques, vector autoregression (VAR) models, Auto-Regressive Moving Average with eXogenous inputs (ARMAX), AutoRegressive (AR) models, Linear Dynamical Statespace (LDS) models, etc, to predict the evolution of the system state $\hat{y}_k$ [91]. The forecast is compared with the observed sensor reading y_k. If the residual r_k, i.e. the difference between the forecast and sensor measurement, exceed what is expected, that may be an indication of an attack or fault. Many well-known techniques have been used to examine or perform statistical tests on the residuals to subsequently detect the attack or alarm. Such techniques include Cumulative Sum (CUSUM) [34, 71], Sequential Probability Ratio Testing (SPRT) [93, 96], Generalized Likelihood Ratio (GLR) testing [10], Compound Scalar Testing (CST) [27, 79] and windowed techniques. Each of these techniques have their advantages and disadvantages which are often dictated by the scenario. We focus on attack detectors that employ the windowed approach to detect attacks in this work.

The windowed procedure generally accumulates the sum of the residuals over a sliding window. Many existing solutions have deployed this procedure for fault detection and attack detection [14, 33]. Compared with one-shot approaches such as the static chi-squared detectors where only one control period is considered, windowed approach has the benefit of historical observations and therefore tend to have fewer false alarms. However, we assert that these time-windowed detectors are inadequate for real-time systems for two reasons. First, authors of windowed solutions strive to select the "best" time window length that produces low false alarms and short *detection delay*. Detection delay is the time it takes to detect an attack after its launch. As the results in [14] showed, on one hand, a short time window leads to high false positive alarms and short detection delay. On the other hand, a longer time window leads to low false positives and longer detection delay. Clearly, there is a trade-off between the false alarm rate and the detection delay. Therefore selecting a window length that attains shortest detection delay and low false alarm can be difficult task if not unachievable.

Secondly, selecting a fixed window length for the attack detector gives it a static behavior which is contradictory to the behavior of dynamic CPS which evolve into various states as it interacts with its environment. Choosing a fixed time-window length means that an alert can only be raised when the time-window expires, causing the detector to have a fixed detection delay. With a fixed detection delay, these detectors are unable to meet detection deadline i.e. the time by which attack must be detected before the system enters unsafe operating state. It is desirable and requisite that the CPS does not enter unsafe operating states when an attack occurs, unfortunately, fixed-window detectors are unable to ensure this in dynamic systems. It can be said that existing works have focused

only on raising attack alerts whilst overlooking detection deadlines. Obviously, raising an attack alert after damaging consequences have occurred is as bad as the non-existence of an attack detector.

In this dissertation, (1) we show that a variable-time window detector is more usable in real-time systems and (2) we propose a variable-time windowed framework for detecting sensor attacks before the system enters unsafe state. The framework consist of three major components: attack detector, state predictor and deadline analyzer. At the core of our framework is the attack detector that uses a stateful detection strategy that performs statistical tests on residuals using variable window size depending on the detection deadline to be met. The state predictor utilizes a data model that captures the nominal behavior of the automotive CPS to predict sensor measurement. The deadline analyzer component proposes a method that calculates the detection deadline after which the system might enter unsafe operating state.

The contributions of this work is as follows: (1) we argue and show that a variable-time window detector is more usable for real-time systems. (2) we propose, design and implement the variable-time attack detector prototype. (3) we perform evaluations of the proposed framework using data from a real testbed, real vehicle and Ardupilot's SITL Rover simulator

3.2.1 Preliminaries

Detection Strategy

As noted above, attack detectors perform statistical tests on the residuals r_k. They raise alerts whenever the expected and observed significantly differ i.e. the residual is large. Two main strategies are used for attack detection namely *stateless* and *stateful* tests [91]. In a stateless test, only one time shot is considered and the detector raises an alert for every deviation at time k (i.e. $|y_k - \hat{y}_k| = r_k \geq \tau$, where τ is a predetermined threshold). This strategy tend to produce many false alarms as the cause of the deviation may be a transient fault at that point in time. Typical example is found in chi-squared ($\tilde{\chi}^2$) detectors.

Stateful strategy, on the other hand, considers multiple time-steps to determine if an alert should be raised. It maintains a statistic S_k that keeps track of the historical changes of r_k. An alert is raised whenever a persistent deviation is observed over multiple time-steps (i.e. $S_k \geq \tau$). Keeping track of the historical changes of r_k can be done in multiple ways such as (1) using change detectors (2) taking an exponential weighted moving average (EWMA) and (3) taking an average over a time-window. We focus on detectors that use the stateful strategy and uses a fixed-time window.

Threat Model

The threat model assumes that the adversary is able to compromise the sensor by leveraging any physical attack techniques that injects interfering signals (magnetic field, light, etc) in the physical environment of the sensor. Such an attack compromises the

integrity of sensor measurements, hence, a false system state is transmitted to the controller. The false sensor data have a misleading ripple effect on the control input that the controller computes and the control output performed by the actuator. In the end, the physical attack drifts the system away from its reference state.

We assume that the attacker, however, does not have access to the control program running on-board and the proposed framework components. As noted above, we do not focus on cyber attacks i.e. attacks that are launched via software or firmware, as they can be effectively defended by existing software security techniques (e.g., CFI). Rather, we focus only on sensor attacks. Attacks that target non-vehicle control logic such as the automotive CPS' computer vision system are out of scope.

Framework Overview

The components of the proposed framework function together to raise an alert for a sensor attack before the CPS enters into unsafe operating state. The state predictor component predicts the expected sensor measurements as the system operates. The window adaptor component computes the detection deadline and the window length (detection delay) that enables the framework to meet detection deadlines. The attack detector component takes input from the window adaptor and the state predictor to perform the following tasks respectively: (1) computes the residual r_k which is the difference between the expected values and the observed. If r_k is large, it obtains the window size as input from the deadline analyzer.

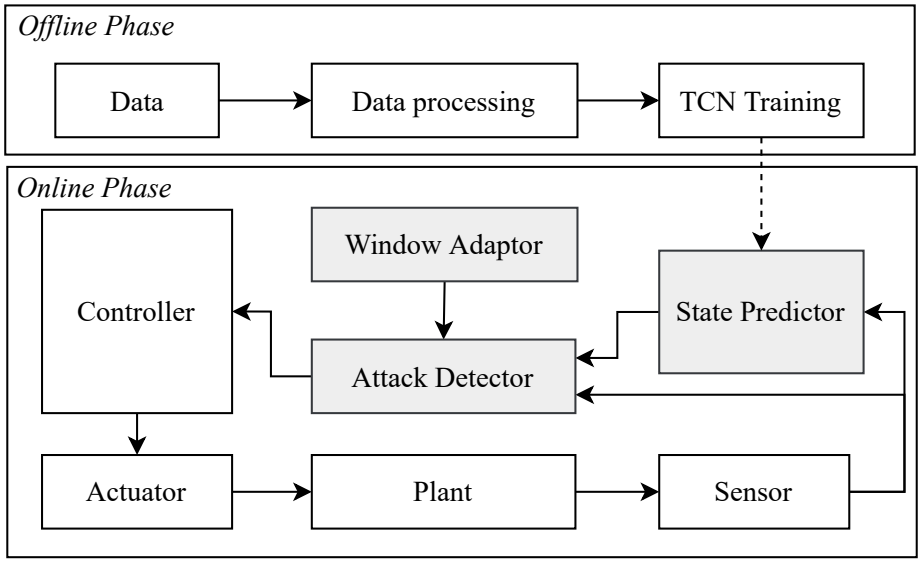


Fig. 3.16.: System design of variable window real-time sensor attack detection framework.

(2) the window size input is used in the detection strategy. Note that choosing variable window size allows our detector framework to adapt its behavior to meet detection deadlines.

3.2.2 Design of Attack Detector

We present the design of the attack detector component in this section. The component is responsible for performing the attack detection strategy of our framework utilizing inputs from the Window Adaptor (§3.2.4) and the State Predictor (§3.2.3) components.

We formulate the attack detection problem as follows. Given the predicted (expected) sensor value $\hat{y}_t \in \mathbb{R}^n$, the sensor reading $y_t \in \mathbb{R}^n$, a predetermined threshold τ, and the

window length l, we want to determine the appropriate time to raise an alarm t_{alarm} before the system touch unsafe state:

$$t_{alarm} = \sum_{t=k-l+1}^{k} r_t > \tau \tag{3.13}$$

where $r_t = |y_t - \hat{y}_t|$. Note here that the moving sum of the residuals is taken over a window $[k - l + 1, k]$. $\hat{y}_t$ is the output of the state predictor which has the responsibility of predicting expected sensor values. The window adaptor provides the window length l to be used in the detection strategy.

3.2.3 Design of State Predictor

We present the design of the state predictor component in this subsection. This component is responsible for predicting or estimating the system states based on historical data from a fixed-size sliding window, i.e. fixed-size reception fields.

Temporal Convolutional Network(TCN) is an architecture that is designed to capture the action segmentation in time-series at first [56, 57]. Recently, some work showed TCN outperformed RNN-based structure on various time-series tasks [38, 59, 64, 99]. Additionally, TCN can be easily trained in parallel since there is no gate components in the network. Aside from that, there are 3 motivations of using TCN instead of RNN-based structure such as LSTM and GRU for our state predictor.

- First, the prediction accuracy is higher than LSTM. In other words, TCN can generate better prediction that are more close to the the system state.

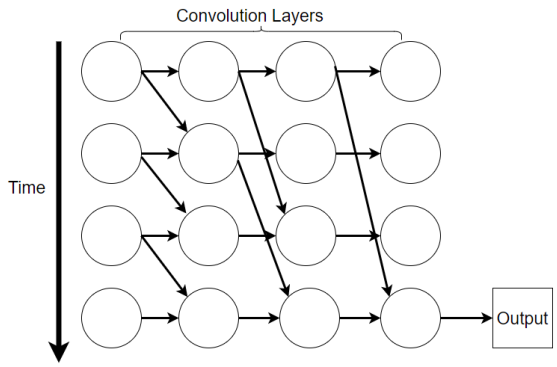


Fig. 3.17.: Temporal Convolutional Networks structure

- Secondly, some work showed that the inference time of TCN is less than LSTM [55]. This attribute can improve the framework's decision making speed as the expected sensor speed is inferred more quickly.

- Lastly, since the TCN could be fairly deep, TCN is capable of memorizing longer history than LSTM. A longer memory helps to generate better prediction when there are long-term dependency between data among the time horizon.

The structure of TCN as shown in Fig. 3.17, the prediction on each step get benefits from various length time dependency cross each layer. Therefore, TCN has fairly long memory with the increment of the depth. TCN does not have complex components in the network as compared with LSTM which has gates, the inference speed of TCN should be faster than LSTM at the same level of accuracy.

As shown in Fig. 3.17, each layer has a different dilation rate s, i.e. distance between convolution steps, which helps the networks decide the steps that the convolution will be applied to. Furthermore, a residual connection is responsible for combining the

convolution signal and inputs of the layer. Then, the dilated $\hat{S}_{\mathrm{t}}^{(l)}$ of l^{th} layer at step t and results after implementing residual connections $S_t^{(l)}$ could be formalized as [56]:

$$\hat{S}_{\mathrm{t}}^{(l)} = f\left(W^{(1)} S_{t-s}^{(l-1)} + W^{(2)} S_t^{(l-1)} + b\right) \tag{3.14}$$

$$S_t^{(l)} = S_t^{(l-1)} + V\hat{S}_t^{(l)} + e \tag{3.15}$$

where $W(i)$ denotes the i^{th} convolution filter in the layer, b denotes the bias vector, v and e denotes the weights and bias vector of the residual.

3.2.4 Design of Window Adaptor

This component functions to determine the time window length to be used in the attack detector. It is made up two sub-systems: the deadline estimator and window-length analyzer. Each of the components is discussed below.

Deadline Estimator

The deadline estimation method used here is the same as the one used in the CUSUM-based framework discussed above and therefore, it shall not be repeated here. The interested reader is referred to §3.1.4 above for the details of the deadline estimation method. In the end, this component outputs the detection deadline by which an alert must be raised before the system enters unsafe state. Again, note that our framework does not

rely on any specific deadline estimation method, and is always applicable as long as a detection deadline is outputted.

Window-length Analyzer

This component determines the appropriate window length to be used in the detection strategy. Remember that the choice of time-window length dictates a trade-off between false alarm rate and detection delay. This component enables the framework to bias detection delay and false alarm rate. One of the goals of the proposed attack detector framework is to meet the attack detection deadline.

This component functions in two phases. The offline phase profiles the CPS to build a lookup table that establishes the relationship between the time-window length, and the detection delay. This phase is performed only once for the CPS. During the online phase, to perform its online adaptive functionality, the window analyzer queries the lookup table to output the time window length that adjusts the detection delay to meet the given detection deadline.

3.2.5 Evaluation

This section evaluates the proposed solution's effectiveness. First, we assess the state predictor's ability to learn the nominal system behavior. Second, we compare the fixed window approach with our variable approach. Lastly, we measure the false-positive and false negative rates.

We implemented our deep learning model in Python, utilizing PyTorch Deep Learning framework. We train the model on Ubuntu 18.04 64-bit with sixteen Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz CPUs, two Nvidia GeForce GTX 1080 GPUs and 64 GB RAM. We follow a 60/20/20 proportions for splitting the original dataset into training/validation/test sets. We used the publicly-available real-world automotive CAN bus dataset from the AEGIS Big Data Project [44] [2] for our experiment. The sensor data, sampled at 20Hz, was collected during trips in the same passenger vehicle.

3.2.6 Experiments and results

Experiment I: This experiment measures the state estimation capability of the state predictor. Its ability to make predictions means the model captured the nominal behavior

[2]https://zenodo.org/record/3267184#.X5YtpIhKg2x

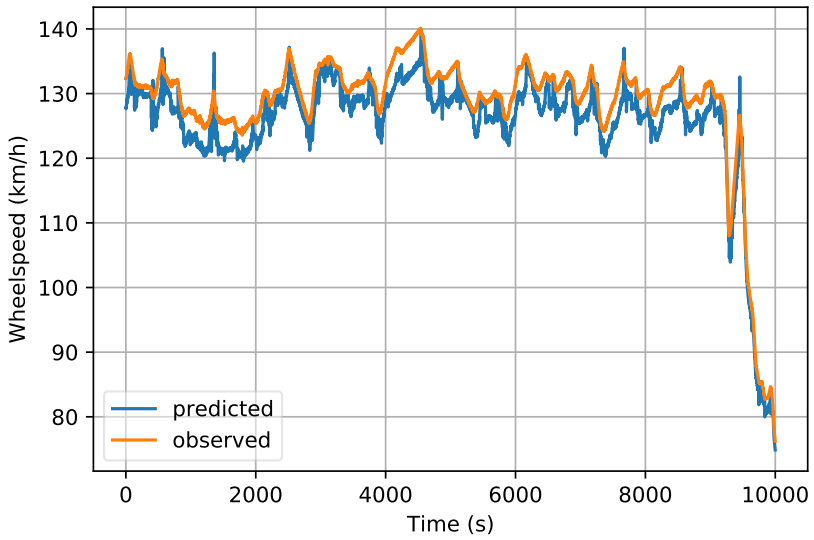


Fig. 3.18.: The state predictor forecasting the wheel speed sensor.

well. Fig. 3.18 shows the results of the state predictors prediction of the wheel speed measurement. It can be observed from the figure that the two lines representing the actual measurements and the forecast are matching closely. It is therefore an indication that the nominal system behavior were captured. As an approximation of the system behavior, the model incurred near-zero errors on the average. This can be improved upon by using more training data that contains more nominal system data.

Experiment II: In this experiment, we measure the false-positive (FP) and false-negative (FN) rates of the detector. First, we embed ten simulated attack ranges in the test data set. The first attack range compromises the observed sensor measurement by adding 0.2 km/h. The magnitude of subsequent attack ranges increases by 0.2 km/h i.e. the second attack range adds 0.4 km/h to the data, the third 0.6 km/h and so on.

Second, the detector is tasked to detect the ten attacks under different monitoring parameters (window length and threshold). Fig. 3.19 shows the results of FP. It can be observed that as the window length gets larger, the FP decreases. This observation is due to normalization of the accumulated errors before the threshold comparison. Hence, a larger window results in smaller normalized errors. On the other hand, the FN is observed in Fig. 3.20 to increase as the window get larger. The same reason for the FP results also attributes to the FN observation. These two observations show the detector can vary its behavior by varying the window length and still achieve acceptable low FP and FN.

Experiment III: In this experiment, we show the proposed framework enables the detector to achieve varying detection delays in order to meet detection deadlines. We experiment under a scenario shown in Fig. 3.21. An attack occurs at 10s which has to be detected by 16s (detection deadline). In the figure, we observe that the various window

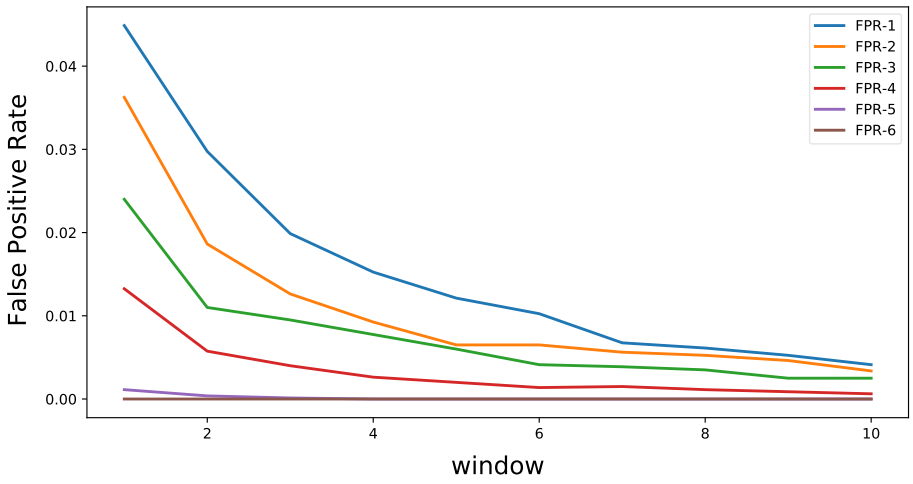


Fig. 3.19.: The false-positive rate measurements under various monitoring parameters. FPR-1 means a threshold of 1. FPR-2 means a threshold of 2 and so on.

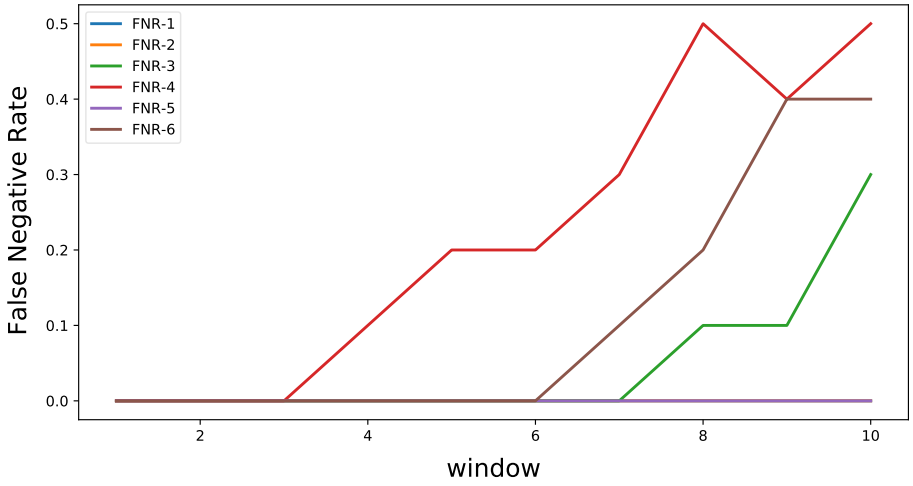
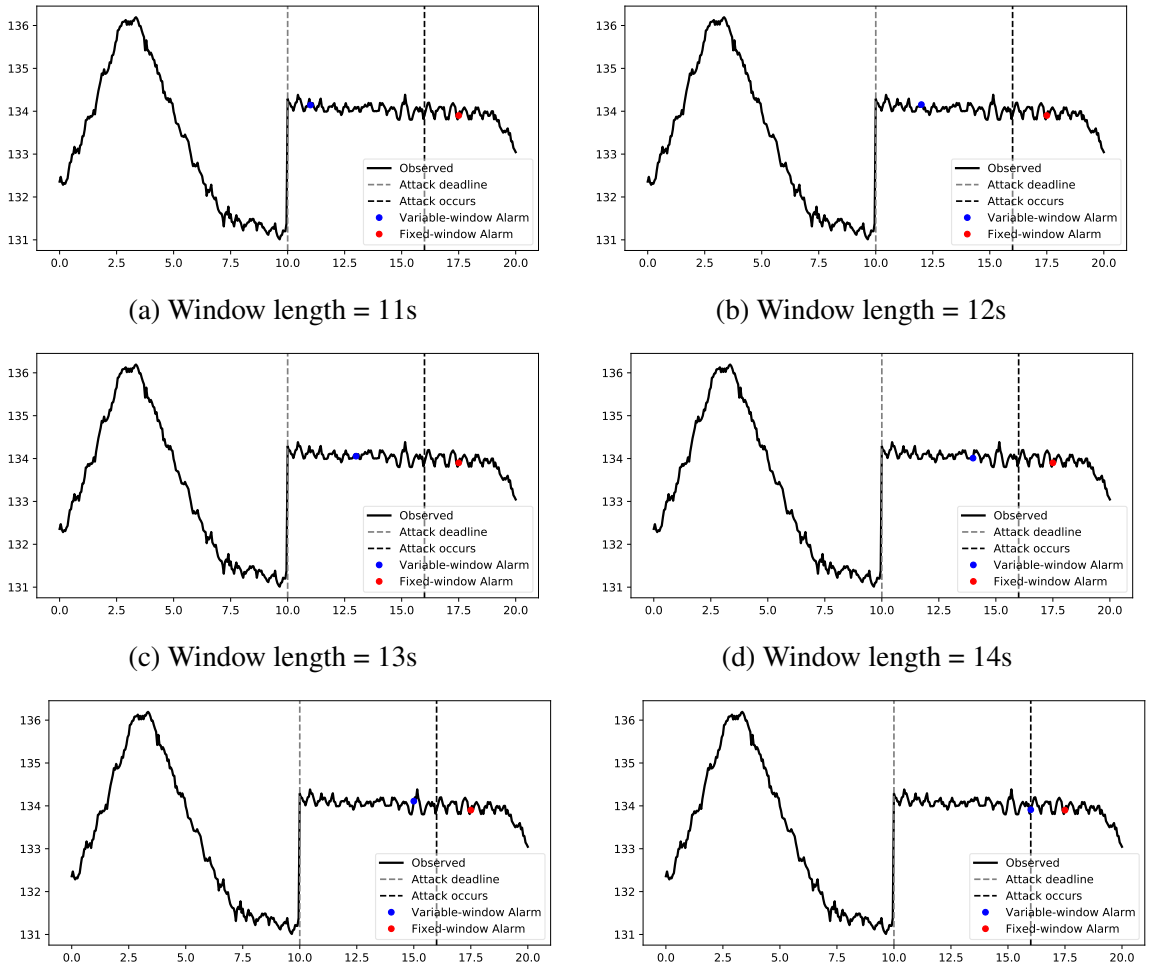


Fig. 3.20.: The false-negative rate measurements under various monitoring parameters. FNR-1 means a threshold of 1. FNR-2 means a threshold of 2 and so on.



(a) Window length = 11s

(b) Window length = 12s

(c) Window length = 13s

(d) Window length = 14s

(e) Window length = 15s

(f) Window length = 16s

Fig. 3.21.: Comparing our framework with a fixed time-window approach. In all scenarios the fixed-time-window detector raises the alarm at 17.5s always. Our approach enables varying the window length such that alarms can be raised before the detection deadline.

length achieve varying detection delays. For instance, Fig 3.21a results in a detection delay of 11s whereas Fig 3.21e results in a detection delay of 15s. Note that a fixed-time-window detector always have the same detection delay (17.5s in our experiment) which prevents it from meeting many detection deadline.

3.2.7 Conclusion

In this work, we have shown that sensor detectors should be able to bias its metrics when the system sits in various states. Specifically, we argue that window-based detectors can vary the window length to achieve faster detection delay to meet a detection deadline or to achieve a certain rate of false alarm. Further, we proposed and evaluated an adaptive detection framework that uses three components to achieve these.

4. RECOVERY-BY-LEARNING

Autonomous cyber-physical systems (CPS) are susceptible to non-invasive physical attacks such as sensor spoofing attacks that are beyond the classical cybersecurity domain. These attacks have motivated numerous research efforts on attack detection, but little attention on what to do after detecting an attack. The importance of attack recovery is emphasized by the need to mitigate the attack's impact on a system and restore it to continue functioning. There are only a few works addressing attack recovery, but they all rely on prior knowledge of system dynamics. To overcome this limitation, this dissertation proposes Recovery-by-Learning, a data-driven attack recovery framework that restores CPS from sensor attacks. The framework leverages natural redundancy among heterogeneous sensors and historical data for attack recovery. Specially, the framework consists of two major components: state predictor and data checkpointer. First, the predictor is triggered to estimate systems states after the detection of an attack. We propose a deep learning-based prediction model that exploits the temporal correlation among heterogeneous sensors. Second, the checkpointer executes when no attack is detected. We propose a double sliding window based checkpointing protocol to remove compromised data and keep trustful data as input to the state predictor. Third, we implement and evaluate the effectiveness of our framework using a realistic data set and a ground vehicle simulator. The results show that our method restores a system to continue functioning in presence of sensor attacks.

4.1 Preliminaries

4.1.1 Scope and Contributions.

This paper focuses on sensor attack-recovery. We assume an attack detector is already in place, and our goal is to take the alerts generated by the detector to recover the system from the attacks. The contributions of this work are as follows. (i) We propose Recovery-by-Learning, a model-free attack recovery framework. (ii) We propose a deep learning based state prediction method and a double sliding window based checkpointing protocol. (iii) We perform extensive data-driven simulations to validate the proposed methods.

4.1.2 Sensor Correlation

Autonomous CPSs are equipped with a number of sensors that enable them to perform their function. The sensors monitor various physical properties such as engine revolutions, vehicle and wheel speed, oil temperature, boost pressure, accelerator pedals, location, etc. It is observed that a subset of sensors on the CPS responds to a physical phenomenon in a correlated or related manner. Such a group of sensors are referred to as heterogeneous sensors or are said to exhibit inherent sensor redundancy. For instance, applying the brakes of a vehicle causes decrements in the engine's RPM, wheel speed, vehicle speed and GPS speed sensors measurements. Similarly, pressing the accelerator pedal leads to increases in the readings of these heterogeneous sensors. Fig. 3.3 shows the pairwise correlation among sensors in an automobile using the dataset of [44]. Details of the data

set will be given in Section 4.3.2. It is observed in the figure that the wheel speed sensors have a strong correlation with the engine RPM and boost pressure sensors. Hence, when the wheel speed sensor reading increases as a result of applying the accelerator pedals, the readings of the engine RPM and boost pressure will also be observed to increase under normal conditions [25, 31, 58, 90, 94].

We believe that exploiting and capturing this inherent sensor redundancy allows us to approximate the nominal system behavior which in turn, enables the accurate prediction of system behavior such as sensor readings. We leverage this notion in our proposed attack recovery system.

4.1.3 Threat Model

We consider the attack scenario where an attacker launches physical attacks against the CPS sensors. Examples of such attacks include optical sensor spoofing [17], gyroscope sensor spoofing [87], accelerometer spoofing attacks [88] among others. These attacks transmit compromised sensor data which does not reflect the actual system state. When the controller receives and processes such data, erroneous control inputs are calculated and issued resulting in safety problems, abnormal system operation and possibly stalling the CPS.

As noted above, there are many proposed attack detection solutions, therefore, we assume the existence of a sensor attack detector that is able to raise an alert whenever any of these attacks occur. Our goal is to automatically respond to the attack alert and steer the system towards a reference state thereby ensuring safety and CPS operation continuity.

We also assume the controller, actuator, the proposed deep learning model and the stored historical sensor data are not compromised.

4.1.4 System Overview

Fig. 4.1 shows an overview of the proposed system. It consists of an offline phase and an online phase. The offline phase involves the data collection, pre-processing of the data, and training the model on the data. The online phase has two major components: the State Predictor and Checkpointer. The state predictor estimates system states when the observed sensor data are no longer trustworthy. The state predictor is built on a deep learning model that captures the nominal system behavior. The Checkpointer ensures valid historical state estimates are stored so that the state predictor can output accurate state estimations.

The proposed system works as follows: When a time-window-based attack detector raises an alert, the state predictor is activated. The checkpointer provides valid and trustful sensor values as input to the state predictor to predict state estimates. The predicted values are forwarded to the controller to perform recovery control commands. In the event the attack detector does not raise an alert, the system continues to function and only the *checkpointer* performs tasks to warrant that only valid historical sensor data is stored as checkpoints.

4.2 Recovery System Design

We describe the details of the proposed attack recovery system in this section.

Rationale. Estimating the accurate behavior of a cyber-physical system, specifically sensor measurement is non-trivial, yet it is a crucial step in attack recovery. The actual behavior of the CPS is guarded by physical laws hence under normal conditions, the physical system properties, called *physical invariant* should always hold. While physical invariant can be captured using a physical system model, it requires in-depth knowledge of the system dynamics which may not be easy to attain.

We propose an attack recovery system that does not require substantial knowledge of system dynamics. We treat the physical system as a black box yet we are able to approximate the nominal system behavior. We achieve this through a deep learning technique that explores the natural redundancy among its heterogeneous sensors. Our approach is based on the insight that under normal conditions, where physical laws are obeyed, the sensor readings also indirectly obey physical laws. Therefore, by learning the relationships among sensor data, the physical invariants are approximated. Having modeled the system from sensor data, we are able to estimate sensor readings with a near-zero error.

4.2.1 Problem formulation

We envision a solution for the attack recovery problem to involve two main steps. The first step seeks to replace the corrupted sensor data that no longer reflect the true state of the system with reconstructed system state estimates. The second step attempts to control the CPS with the reconstructed values which we call *recovery control*. Given that we consider multiple heterogeneous sensor time-series data or measurements in making a

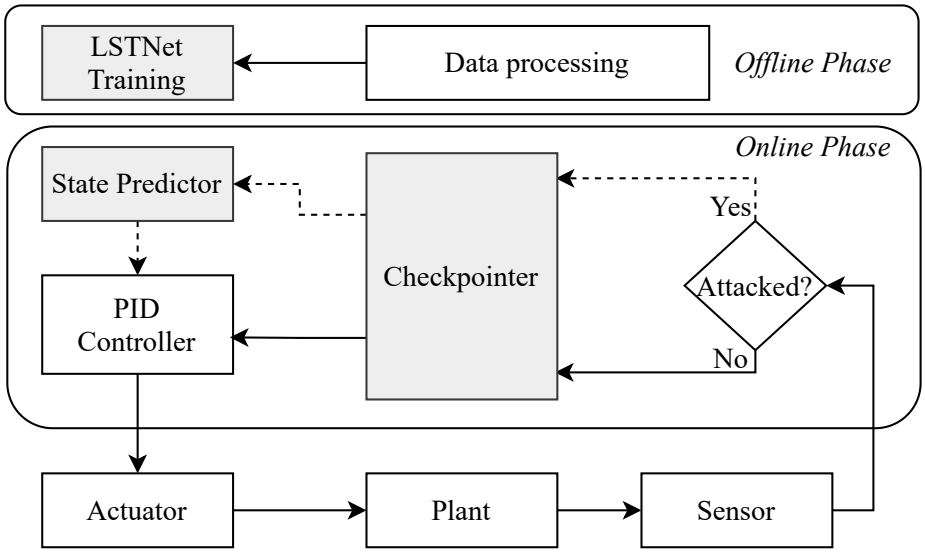


Fig. 4.1.: Design overview of our Recovery-by-Learning framework.

prediction, we formulate the first step as a deep learning multivariate time series forecasting problem [54].

Hence, given valid historical time series output of n heterogeneous sensors $Y = \{y_1, y_2, y_3, ...y_K\}$ where $y_k \in \mathbb{R}^n$, we aim to predict y_{k+h} where h is a time ahead of the current time k. We ensure $Y = \{y_1, y_2, y_3, ...y_K\}$ is always available by proposing a checkpointing protocol to store such valid historical sensor data. For ease of presentation, we assume that the system has full observability. Thus, we formulate the input matrix as $X_K = \{y_1, y_2, y_3, ..., y_K\} \in \mathbb{R}^{n \times K}$. Once a valid prediction is made, the second step undertakes a recovery control that uses the predicted values to drive the system.

4.2.2 System Components

Data Processor: Data pre-processing is an important step in a machine or deep learning task. Specifically, we ensure we extract only features that have a strong

correlation with the sensor of interest. While this can be achieved from domain knowledge, we leverage Pearson Correlation Coefficients (PCC) statistic to observe this correlation in the dataset. This step also has the potential to reveal correlations that may not be obvious to humans. PCC outputs values between -1.0 and $+1.0$. Feature pairs that have a strong positive correlation have PCC values close to $+1.0$. Conversely, a strong negative correlation has PCC values close to -1.0. A zero PCC value indicates there is no correlation between the features. For example, in Fig. 3.3, the vehicle speed sensor has PCC of approximately 1.0 with the engine RPM and the wheel speed sensors. The vehicle speed sensor, however, has a PCC value close to zero with the ambient temperature sensor.

LSTNet Training: In order to automatically exploit the correlation that exists in the sensor data, we train a deep learning model based on LSTNet [54]. LSTNet was originally developed to model long and short term temporal forecasting for multivariate time series. The deep learning architecture captures nonlinear aspects of the system by using a convolutional neural network (CNN) and recurrent neural network (RNN) for exploiting short and long term correlations respectively. To improve scalability and robustness, the model also includes autoregressive units that enable the DL model to capture linear aspects of the system as well. Fig. 4.2 shows the deep learning architecture that trains our model. Training the model requires a number of hyperparameters to be specified, notable among them is the *window* p. The window p specifies how many historical data points should be used in making a prediction.

State Predictor: This component is built on the trained deep learning model training discussed above. At this point, the data model has captured the nominal behavior of the system by exploring the correlation among heterogeneous sensors. It serves as a nominal

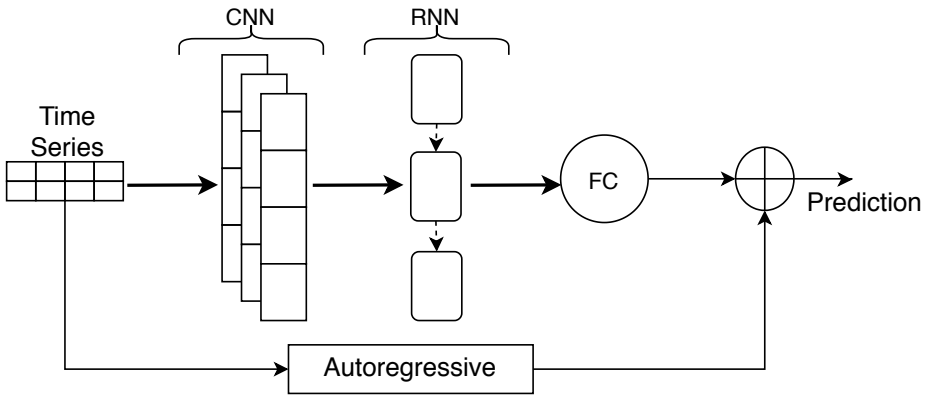


Fig. 4.2.: Overview of deep learning model architecture (LSTNet [54]). FC refers to Fully Connected.

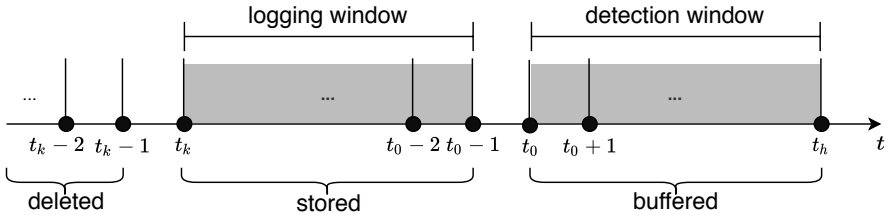


Fig. 4.3.: A double sliding window based checkpointing protocol.

approximation of the system behavior and hence it is able to make predictions of the system behavior given the right input. In order to make the deep learning model useful in a non-Python environment, we utilize Torchscript to build an intermediate representation (IR) so that it can be used in high-performance environments such as C++ to make predictions. The firmware of the ground vehicle simulator used in our experiment is written in C++.

Checkpointer: Attack detection mechanisms take some time to detect an attack after the attack's launch, called *detection delay*, before raising an alert. As a result, we cannot trust the sensor readings during the detection delay since they may have been compromised. A successful recovery cannot be achieved if we rely on corrupted data,

hence to address this issue, checkpointing protocols [50, 51, 62, 72] have been proposed to provide trustworthy historical data that can be used for recovery. Though viable, especially for model-based recovery methods [102], these protocols have limitations of storing only one data point making it unsuitable for learning-based methods such as ours which require an interval of data points for reconstructing sensor data.

The checkpointer component addresses this limitation by proposing a checkpointing protocol shown in Fig. 4.3 that is not only applicable to learning-based methods but model-based methods as well. The proposed protocol adopts a double sliding window instead of the single sliding window approach used in existing works. This approach enables the protocol to capture an interval of historical data and the detection delay. The two windows of the protocol slide forward and records the sensor values $x(t)$ as time ticks.

The protocol has three steps namely buffer, store, and delete. (1) *Buffer*: The data in this step is possibly compromised and has a duration equivalent to the detector's detection delay. State estimates or sensor data within the detection window, $x(t_0), ..., x(t_h)$ are first buffered. (2) *Store*: Given that the detection window equals the detection delay, the data points in this step have moved outside the detection window and are therefore considered trustworthy. Note that an interval of time series datapoints (logging window data) are stored instead of a single data point. Hence, for the interval $[t_k, t_0 - 1]$, datapoints $\{x(t_k), ..., x(t_0 - 1)\}$ are stored. Remember that this length is equal to the p window hyperparameter of the LSTNet component discussed above. (3) *Delete*: All historical data that are older than those in the logging window are no longer needed and should therefore be deleted. Data points $x(t_k - 1)$ and $x(t_k - 2)$ are discarded as shown in the figure.

When the detector raises an alarm, time series datapoints of the interval $[t_k, t_0]$ will be used to rebuild estimate $x(t_h)$.

4.3 Evaluation

We perform experiments to evaluate the effectiveness of our proposed attack recovery.

4.3.1 Implementation and Experimental Setup

We implemented our deep learning model in Python, utilizing PyTorch Deep Learning framework. The experimental model is made up of 120 convolutional layers, 120 GRU layers and an AR model. A batch size of 128 serves as input to the network. We train our proposed DL model on Ubuntu 18.04 64-bit with sixteen Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz CPUs, two Nvidia GeForce GTX 1080 GPUs and 64 GB RAM. We split the original dataset into 60% training, 20% validation and 20% test sets.

4.3.2 Dataset Description

We evaluate the efficiency of the proposed recovery system using the publicly-available automotive CAN bus dataset from the AEGIS Big Data Project [44] and data collected from Ardupilot SITL virtual ground vehicle.

The AEGIS data was collected during trips conducted by three drivers driving the same vehicle. It contains more than 40 sensor measurements including but not limited to the four wheel speed sensors, engine speed, vehicle speed, steering angle, ambient temperature, GPS, oil temperature and boost pressure.

Ardupilot SITL (software in the loop) is a simulator package that provides a native executable that allows one to run Plane, Copter or Rover (ground vehicle) without any hardware. The virtual ground vehicle that we use in our experiment runs a firmware (APMrover2 2.5) that is used in real unmanned ground vehicle boards. The vehicle is equipped with a number of sensors including GPS, IMU, RPM, optical flow sensors.

4.3.3 Experiments and Results

Experiment I: This experiment verifies if the State Predictor learned the nominal behavior of the CPS and evaluates its effectiveness in reconstructing sensor data. We build two models: the first one is based on the AEGIS dataset and the second is based on the virtual unmanned ground vehicle's sensor data. The second model is also used in a case study to demonstrate how the proposed framework recovers the unmanned ground vehicle (UGV) from a speed sensor attack.

Note, however, that in this experiment no attack has been launched. Fig. 4.4 and Fig. 4.5 show the model predictions for the engine speed and boost pressure sensors in the AEGIS test dataset. In the figures, the predicted speed (red line) closely matches the observed speed (blue line) indicating the model captured the nominal behavior of the vehicle. The figures also show the mean error or residual is near zero indicating the predictor is not biased. Residual analysis shows the error follows a normal Gaussian distribution and it is free from any cyclic, trend and seasonal structures.

We perform a similar experiment on the UGV. Using Mission Planner [8], we generate missions (trajectory) that the vehicle executes. We collected the sensor data from the

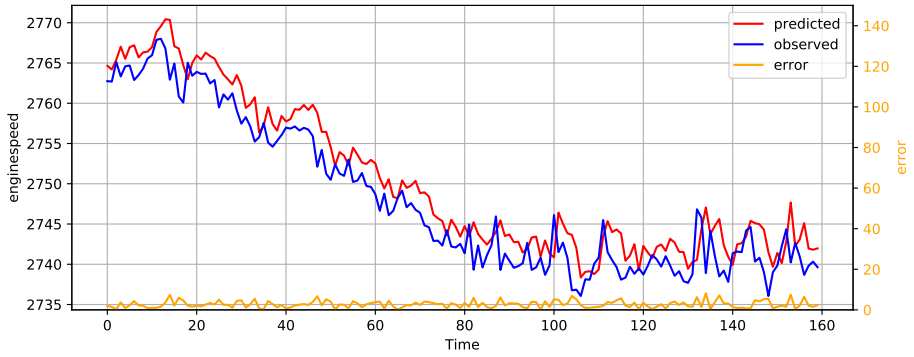


Fig. 4.4.: Observed and predicted engine speed sensor readings in the AEGIS dataset.

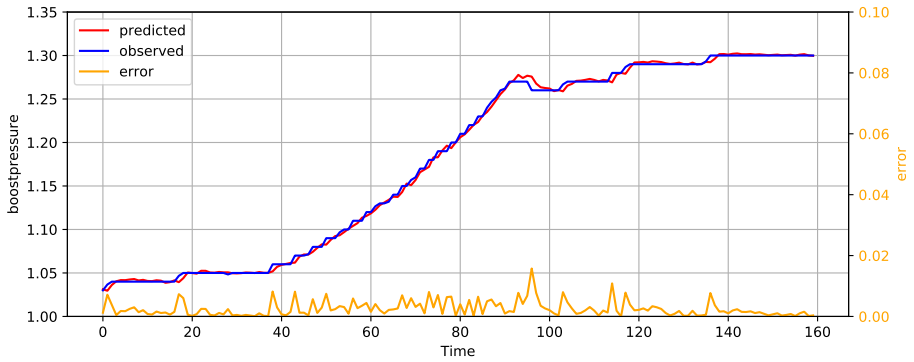


Fig. 4.5.: Observed and predicted boost pressure sensor readings in the AEGIS dataset.

dataflash log and used it to build a data model. The model's predictions for the UGV's speed sensor test dataset is shown in Fig. 4.6. The results depict a close match between the model predictions and the observed sensor values. Similar to the results in Fig. 4.4 and Fig. 4.5, there is a mean error in the model that is near-zero.

Experiment II - Case Study: We demonstrate attack recovery in this case study. The attacker launches attack on the speed sensor which leads the cruise/speed controller to issue wrong control inputs resulting in the UGV to travel above its cruise speed of 5 m/s. Fig. 4.7 shows the case study considered in this experiment. At 60 sec, we simulate an attack that transmits $\rho - 4$ m/s as the forward speed of the vehicle, where ρ is the actual

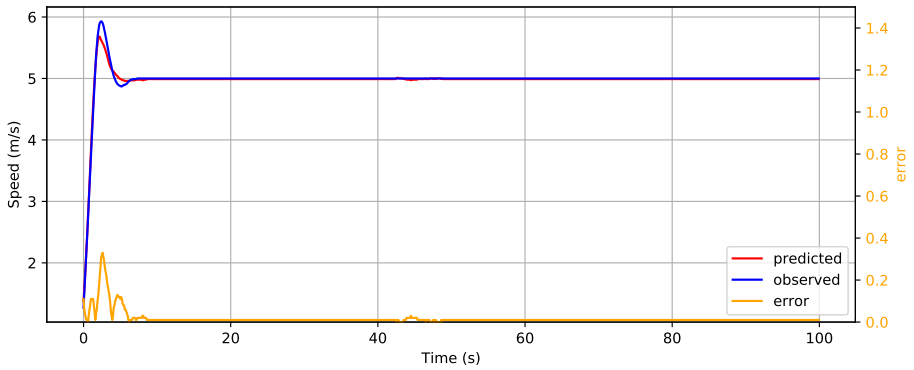


Fig. 4.6.: Observed and predicted boost speed measurements of the unmanned ground vehicle.

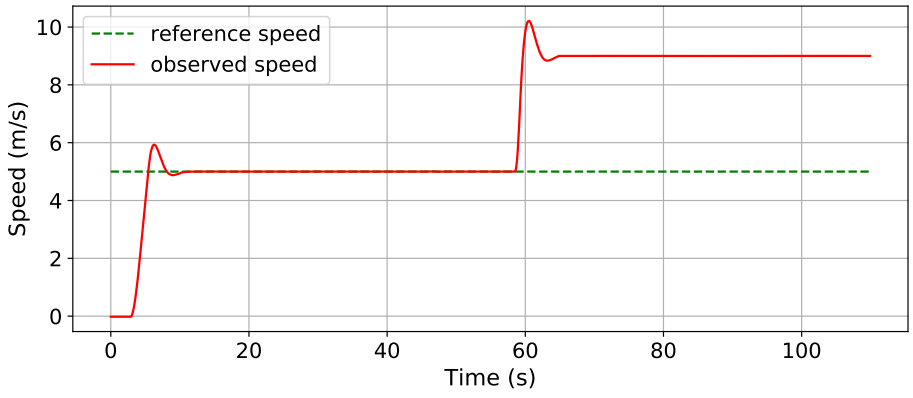


Fig. 4.7.: UGV cruises above its reference speed of 5 m/s.

speed of the vehicle. Hence, in order to maintain the reference speed, the PID controller issued a higher throttle output that resulted in the vehicle to cruise at about 9 m/s. In real-life, this scenario can be a safety concern.

Recovery control: When this attack is detected, we can no longer trust the sensor data and therefore the system states must be estimated or reconstructed. Our proposed framework responds with the goal of getting the UGV to cruise at its reference speed (5 m/s). It achieves that by activating the state predictor component which uses the data in the checkpointer as input. The state predictor reconstructs sensor values that are forwarded to the cruise controller. The controller calculates throttle outputs based on the reconstructed values that eventually cause the vehicle to travel at the reference speed as seen in Fig. 4.8.

Experiment III: The effectiveness of the proposed framework largely depends on the p value selected i.e. how many historical values are used for the deep learning model's prediction (see Section 4.2.2). Remember also that the p value is equal to the length of the logging window in the proposed checkpointing protocol. In this experiment, we provide an analysis of the p value so that the optimal value can be selected to predict more accurate sensor values. We compare various p values based on accuracy metrics: mean square error (mse), root relative squared error (rse) and relative absolute error (rae). Table 4.1 shows the results of the vehicle speed sensor values in the AEGIS dataset. Values between 56 and 84 produced the highest prediction values. A similar analysis done on the ground vehicle showed a slightly different results which leads us to conclude that the best p value is device/dataset-specific.

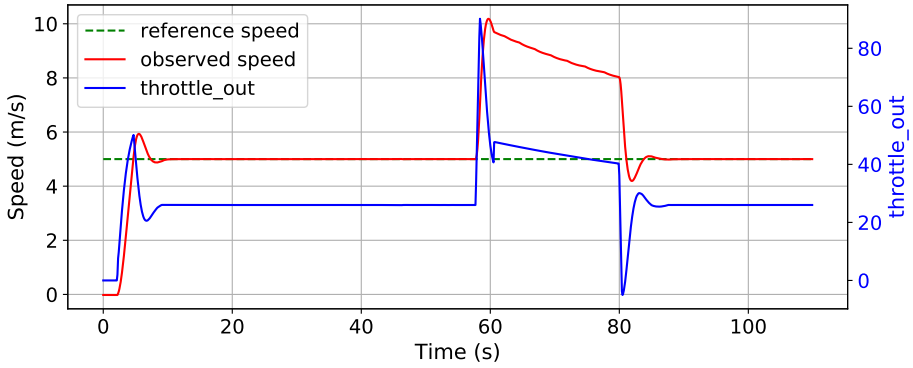


Fig. 4.8.: Speed attack recovery.

Table 4.1: Comparison of p values based one accuracy metrics.

p	**mse**	**rse**	**rae**
28	0.020843	0.0034	0.0020
42	0.019189	0.0034	0.0020
56	0.012859	0.0031	0.0018
84	0.011068	0.0032	0.0018
168	0.021194	0.0034	0.0020
188	0.008365	0.0034	0.0018

4.4 Conclusion

In this paper, we have presented a model-free attack recovery system that does not require in-depth knowledge of system dynamics and also allow autonomous CPS to use existing components (controllers and sensors) without further duplication. We achieve this by applying a novel deep learning framework to capture the nominal behavior of the cyber-physical system. We proposed a new generalized double sliding window checkpointing protocol that is usable both in model-based and learning-based recovery methods. We performed experiments to evaluate the effectiveness of the proposed framework using real-world dataset and realistic unmanned ground vehicle simulator. Our results show that our method restores a system to continue functioning in the presence of sensor attacks.

5. SUMMARY

Autonomous cyber-physical systems have transitioned from once-closed architectures to open architectures due to the integration with information technology (IT). Increasingly, these systems are getting more connected to the outside. The integration has enabled the development of many convenient features but at the same time, it has exposed the system to many new threats. The research community has proposed sensor attack detection solutions as a reactive attack resilience measure. However, these solutions have not adequately addressed timing constraints and usability.

This dissertation states the thesis that attack detection should bias different metrics when a system sits in different states. For example, if the system is close to unsafe states, reducing the detection delay is preferable to lowering the false alarm rate, and vice versa. To that end, this dissertation presents the design and evaluation of two new frameworks for the real-time detection of sensor attacks. Chapter 3 presents the two frameworks.

The first framework is a cumulative sum (CUSUM) based detection framework that enables real-time adaptive detection using three necessary components: attack detector, behavior predictor, and drift adaptor.

(i) Attack Detector. As the core of our framework, this component detects anomalies using a CUSUM algorithm that monitors the cumulative sum of residuals between the nominal (estimated by the behavior predictor) and observed sensor values. The algorithm raises an alarm when the cumulative sum of the residuals is greater than a predefined threshold.

Importantly, we augment this algorithm with a drift parameter that governs both the detection delay and false alarms. That is, the algorithm can adjust the two metrics by changing the drift parameter.

(ii) Behavior Predictor. This component estimates nominal sensor values that are fed to the core component. It uses a deep learning (DL) model that is offline extracted through uncovering and exploiting both the local and complex long-term dependencies in multivariate sequential sensor measurements. Thus this model depends on little knowledge of the physical system (e.g., dynamics). Further, this model leverages convolutional neural network (CNN) and recurrent neural network (RNN) to capture non-linear aspects in sensor data and uses autoregressive models to capture linear aspects. This combination results in high robustness and scalability in handling the sequential sensor data.

(iii) Drift Adaptor. The third component is a drift adaptor that estimates a detection deadline and then determines the drift parameter. The detector component uses this parameter for adjusting the detection delay to ensure timely detection as the detection deadline varies over time.

We implement our framework and validate it using realistic sensor data of automotive CPS from the AEGIS Big Data Project [44]. The results demonstrate that our framework can detect attacks in a real-time manner.

The second framework, also presented in Chapter 3, is a real-time adaptive attack detector that biases detection delay and false alarm metrics by varying the window length of the attack strategy. The three components of the proposed framework function together to raise an alert for a sensor attack before the CPS enters into unsafe operating state.

(i) State Predictor. Utilizing the higher prediction accuracy and inference time of Temporal Convolutional Network (TCN), this component models the nominal system behavior from offline system data. The trained model predicts the expected sensor measurements in its online phase.

(ii) Window Adaptor. It computes the detection deadline and output the window length (detection delay) that enables the framework to meet the detection deadlines.

(iii) Attack detector. This component determines the time at which an alert must be raised. It combines the output of the window adaptor and the state predictor in the detection algorithm. The algorithm computes the residual sequence r_k over a window length. r_k is the difference between the expected sensor reading and the observed sensor measurement. If r_k is larger than the predetermined threshold, an alert is raised. Note that choosing variable window size allows our detector framework to adapt its behavior to meet detection deadlines.

The results obtained in the real-time adaptive sensor attack detection frameworks shows we can dynamically alter the behavior of a detector by altering the CUSUM drift parameter and time window length within a certain range. Values outside this range can produce unacceptable system and detector functionalities.

Further, this dissertation notes that raising an alert alone after an attack occurs is not a holistic solution for attack resiliency, this dissertation proposes *Recovery-by-Learning*, a data-driven attack recovery framework that restores automotive cyber-physical systems from sensor attacks. The framework, presented in Chapter 4, requires little knowledge of the system's dynamics, but leverages natural redundancy among heterogeneous sensors and historical data for attack recovery. Specially, the framework consists of two major

components: state predictor and data checkpointer. The state predictor is activated to estimate system states when an attack is detected. The predicted states are forwarded to the controller to calculate and issue appropriate control commands to bring the system back to normalcy. The data checkpointer executes in the normal mode when no attack is detected. It employs a checkpointing protocol to remove corrupted data and keep valid historical data as input to the state predictor to make state estimation. The protocol uses double sliding windows: detection window and logging window. The former accommodates the substantial detection delay (i.e., the time interval between the start of an attack and the detection of it), during which the correctness of the sensor data is still in question and thus using them may result in unsuccessful recovery. The logging window governs sufficient trustful data for the state prediction.

The recovery framework is implemented and evaluated on AEGIS dataset and Ardupilot SITL rover. The results of the experiment and case study shows the framework enables the CPS to return to its reference point after an attack has been detected.

In conclusion, this dissertation has argued that attack detection should have a preference on different metrics when a system sits in different states. We have proposed solutions that enable CPS to achieve that and a solution to provide a recovery measure that mitigates the effects of an attack. The proposed solutions in this dissertation provide a holistic real-time attack resilient solution. Also, our real-time adaptive attack detection sets a new research direction that creates the awareness for the need of dynamic detectors, and motivates the proposal of new solutions to achieve that.

Future Work: This dissertation has carried out simulation-based evaluation of the proposed frameworks as our lab's testbed is still under construction. While the results

show a viable solution, future work is required to provide hardware evaluations and make necessary improvements to the framework. Further, to ensure the smooth interaction of system components, future work is required to handle the scheduling of the component. Lastly, code and model optimization tasks are required to ensure the frameworks are executable on resource constrained systems.

LIST OF REFERENCES

LIST OF REFERENCES

[1] SITL Simulator (Software in the Loop). https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. [Online; accessed Nov-2020].

[2] Alireza Abbaspour, Kang K Yen, Shirin Noei, and Arman Sargolzaei. Detection of fault data injection attack on UAV using adaptive neural network. *Procedia computer science*, 95:193–200, 2016.

[3] Evan Ackerman. Fatal tesla self-driving car crash reminds us that robots aren't perfect. *IEEE-Spectrum*, 1, 2016.

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

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

[6] Omar Al-Jarrah and Ahmad Arafat. Network intrusion detection system using neural network classification of attack behavior. *Journal of Advances in Information Technology Vol*, 6(1), 2015.

[7] Amazon. *Amazon Prime Air*, 2020 (accessed June 29, 2020). https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011s.

[8] Ardupilot. *Mission Planner Home*, 2020 (accessed July 20, 2020). https://ardupilot.org/planner/.

[9] Stanley Bak, Karthik Manamcheri, Sayan Mitra, and Marco Caccamo. Sandboxing controllers for cyber-physical systems. In *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pages 3–12. IEEE, 2011.

[10] Michèle Basseville. *Detecting changes in signals and systems*. PhD thesis, INRIA, 1987.

[11] Richard R Brooks and S Sitharama Iyengar. Robust distributed computing and sensing algorithm. *Computer*, 29(6):53–60, 1996.

[12] Z Chair and PK Varshney. Optimal data fusion in multiple sensor detection systems. *IEEE Transactions on Aerospace and Electronic Systems*, pages 98–101, 1986.

[13] Paul Chew and Keith Marzullo. Masking failures of multidimensional sensors. Technical report, CORNELL UNIV ITHACA NY DEPT OF COMPUTER SCIENCE, 1991.

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

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

[16] Abraham A Clements, Naif Saleh Almakhdhub, Khaled S Saab, Prashast Srivastava, Jinkyu Koo, Saurabh Bagchi, and Mathias Payer. Protecting bare-metal embedded systems with privilege overlays. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 289–303. IEEE, 2017.

[17] Drew Davidson, Hao Wu, Rob Jellinek, Vikas Singh, and Thomas Ristenpart. Controlling uavs with sensor input spoofing attacks. In *10th {USENIX} Workshop on Offensive Technologies (WOOT 16)*, 2016.

[18] Veronique Delouille, Ramesh Neelamani, and Richard Baraniuk. Robust distributed estimation in sensor networks using the embedded polygons algorithm. In *Proceedings of the 3rd international symposium on information processing in sensor networks*, pages 405–413, 2004.

[19] DroneFly. *Police Drone Infographic*, 2020 (accessed June 29, 2020). `https://www.dronefly.com/police-drone-infographic/`.

[20] DroneUp. *Best aerial photography drones for business in 2020: DJI, Freefly, Skydio, and more*, 2020 (accessed June 29, 2020). `https://www.zdnet.com/article/best-aerial-photography-drones/`.

[21] DroneUp. *Complete Drone Solutions*, 2020 (accessed June 29, 2020). `https://www.droneup.com/`.

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

[23] Fan Fei, Zhan Tu, D Xu, and Xinyan Deng. Learn-to-recover: Retrofitting UAVs with reinforcement learning-assisted flight control under cyberphysical attacks. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[24] Fan Fei, Zhan Tu, Ruikun Yu, Taegyu Kim, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. Cross-layer retrofitting of uavs against cyber-physical attacks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 550–557. IEEE, 2018.

[25] Arun Ganesan, Jayanthi Rao, and Kang Shin. Exploiting consistency among heterogeneous sensors for vehicle anomaly detection. Technical report, SAE Technical Paper, 2017.

[26] Wei Gao and Thomas H Morris. On cyber attacks and signature based intrusion detection for modbus based industrial control systems. *Journal of Digital Forensics, Security and Law*, 9(1):3, 2014.

[27] Janos J Gertler. Survey of model-based failure detection and isolation in complex plants. *IEEE Control systems magazine*, 8(6):3–11, 1988.

[28] Thomas D Gillespie. *Fundamentals of vehicle dynamics*, volume 400. Society of automotive engineers Warrendale, PA, 1992.

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

[30] Jonathan Goh, Sridhar Adepu, Marcus Tan, and Zi Shan Lee. Anomaly detection in cyber physical systems using recurrent neural networks. In *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, pages 140–145. IEEE, 2017.

[31] Fei Guo, Zichang Wang, Suguo Du, Huaxin Li, Haojin Zhu, Qingqi Pei, Zhenfu Cao, and Jianhong Zhao. Detecting vehicle anomaly in the edge via sensor consistency and frequency characteristic. *IEEE Transactions on Vehicular Technology*, 2019.

[32] Pinyao Guo, Hunmin Kim, Nurali Virani, Jun Xu, Minghui Zhu, and Peng Liu. Exploiting physical dynamics to detect actuator and sensor attacks in mobile robots. *arXiv preprint arXiv:1708.01834*, 2017.

[33] Ziyang Guo, Dawei Shi, Karl Henrik Johansson, and Ling Shi. Optimal linear cyber-attack on remote state estimation. *IEEE Transactions on Control of Network Systems*, 4(1):4–13, 2016.

[34] Fredrik Gustafsson and Fredrik Gustafsson. *Adaptive filtering and change detection*, volume 1. Citeseer, 2000.

[35] Peter Gutierrez. *Infrastructure Inspection - UAS Are All Over It*, 2020 (accessed June 29, 2020). `https://insideunmannedsystems.com/infrastructure-inspection-uas-are-all-over-it/`.

[36] Samsel Haley. *How Police Forces Are Using Drones to Keep Officers Out of The Line of Fire*, 2020 (accessed June 29, 2020). `https://securitytoday.com/articles/2019/07/25/how-police-forces-are-using-drones-to-keep-officers-out-of-the-li aspx?admgarea=mag&m=1`.

[37] Tianjia He, Lin Zhang, Fanxin Kong, and Asif Salekin. Exploring inherent sensor redundancy for automotive anomaly detection. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[38] Pradeep Hewage, Ardhendu Behera, Marcello Trovati, Ella Pereira, Morteza Ghahremani, Francesco Palmieri, and Yonghuai Liu. Temporal convolutional neural (tcn) network for an effective weather forecasting using time-series data from the local weather station. *Soft Computing*, 24(21):16453–16482, 2020.

[39] Tim Higgins and Matt Grossman. *Amazon to Acquire Self-Driving Startup Zoox*, 2020 (accessed June 29, 2020). `https://www.wsj.com/articles/amazon-to-acquire-self-driving-startup-zoox-11593183986`.

[40] Jun Inoue, Yoriyuki Yamagata, Yuqi Chen, Christopher M Poskitt, and Jun Sun. Anomaly detection for a water treatment system using unsupervised machine learning. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 1058–1065. IEEE, 2017.

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

[42] Abdul Rehman Javed, Muhammad Usman, Saif Ur Rehman, Mohib Ullah Khan, and Mohammad Sayad Haghighi. Anomaly detection in automated vehicles using multistage attention-based convolutional neural network. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[43] Khurum Nazir Junejo and Jonathan Goh. Behaviour-based attack detection and classification in cyber physical systems using machine learning. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 34–43, 2016.

[44] Christian Kaiser, Alexander Stocker, and Andreas Festl. *Automotive CAN bus data: An example dataset from the AEGIS Big Data Project*, July 2019.

[45] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[46] Natallia Katenka, Elizaveta Levina, and George Michailidis. Local vote decision fusion for target detection in wireless sensor networks. *IEEE Transactions on Signal Processing*, 56(1):329–338, 2007.

[47] Sanmeet Kaur and Maninder Singh. Automatic attack signature generation systems: A review. *IEEE Security & Privacy*, 11(6):54–61, 2013.

[48] Chung Hwan Kim, Taegyu Kim, Hongjun Choi, Zhongshu Gu, Byoungyoung Lee, Xiangyu Zhang, and Dongyan Xu. Securing real-time microcontroller systems through customized memory view switching. In *NDSS*, 2018.

[49] Junsoo Kim, Chanhwa Lee, Hyungbo Shim, Yongsoon Eun, and Jin H Seo. Detection of sensor attack and resilient state estimation for uniformly observable nonlinear systems having redundant sensors. *IEEE Transactions on Automatic Control*, 64(3):1162–1169, 2018.

[50] Fanxin Kong, Oleg Sokolsky, James Weimer, and Insup Lee. State consistencies for cyber-physical system recovery. In *the 2nd Workshop on Cyber-Physical Systems Security and Resilience (CPS-SR)*, pages 3–7, 2019.

[51] Fanxin Kong, Meng Xu, James Weimer, Oleg Sokolsky, and Insup Lee. Cyber-physical system checkpointing and recovery. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 22–31. IEEE, 2018.

[52] Moshe Kravchik and Asaf Shabtai. Detecting cyber attacks in industrial control systems using convolutional neural networks. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 72–83, 2018.

[53] Benjamin A Kwapong, Richard Anarfi, and Kenneth K Fletcher. Personalized service recommendation based on user dynamic preferences. In *International Conference on Services Computing*, pages 77–91. Springer, 2019.

[54] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 95–104, 2018.

[55] Pedro Lara-Benítez, Manuel Carranza-García, José M Luna-Romera, and José C Riquelme. Temporal convolutional networks applied to energy-related time series forecasting. *Applied Sciences*, 10(7):2322, 2020.

[56] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks for action segmentation and detection. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 156–165, 2017.

[57] Colin Lea, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks: A unified approach to action segmentation. In *European Conference on Computer Vision*, pages 47–54. Springer, 2016.

[58] Huaxin Li, Li Zhao, Marcio Juliato, Shabbir Ahmed, Manoj R Sastry, and Lily L Yang. Poster: Intrusion detection system for in-vehicle networks using sensor correlation and integration. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2531–2533. ACM, 2017.

[59] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

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

[61] Teppo Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 22, 2011.

[62] Rui Ma, Sagnik Basumallik, Sara Eftekharnejad, and Fanxin Kong. Recovery-based model predictive control for cascade mitigation under cyber-physical attacks. In *2020 IEEE Texas Power and Energy Conference (TPEC)*, pages 1–6. IEEE, 2020.

[63] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89. Presses universitaires de Louvain, 2015.

[64] Brais Martinez, Pingchuan Ma, Stavros Petridis, and Maja Pantic. Lipreading using temporal convolutional networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6319–6323. IEEE, 2020.

[65] Keith Marzullo. Tolerating failures of continuous-valued sensors. *ACM Transactions on Computer Systems (TOCS)*, 8(4):284–304, 1990.

[66] Mario Milanese and Carlo Novara. Set membership identification of nonlinear systems. *Automatica*, 40(6):957–975, 2004.

[67] Robert Mitchell and Ray Chen. Adaptive intrusion detection of malicious unmanned air vehicles using behavior rule specifications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(5):593–604, 2013.

[68] Robert Mitchell and Ray Chen. Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems. *IEEE Transactions on Dependable and Secure Computing*, 12(1):16–30, 2014.

[69] Carlos Murguia and Justin Ruths. Cusum and chi-squared attack detection of compromised sensors. In *2016 IEEE Conference on Control Applications (CCA)*, pages 474–480. IEEE, 2016.

[70] Patric Nader, Paul Honeine, and Pierre Beauseroy. Mahalanobis-based one-class classification. In *2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2014.

[71] Ewan S Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[72] Miroslav Pajic, Insup Lee, and George J Pappas. Attack-resilient state estimation for noisy dynamical systems. *IEEE Transactions on Control of Network Systems*, 4(1):82–92, 2016.

[73] Miroslav Pajic, James Weimer, Nicola Bezzo, Paulo Tabuada, Oleg Sokolsky, Insup Lee, and George J Pappas. Robustness of attack-resilient state estimators. In *ICCPS'14: ACM/IEEE 5th International Conference on Cyber-Physical Systems (with CPS Week 2014)*, pages 163–174. IEEE Computer Society, 2014.

[74] Kaveh Paridari, Niamh O'Mahony, Alie El-Din Mady, Rohan Chabukswar, Menouer Boubekeur, and Henrik Sandberg. A framework for attack-resilient industrial control systems: Attack detection and controller reconfiguration. *Proceedings of the IEEE*, 106(1):113–128, 2017.

[75] Junkil Park, Radoslav Ivanov, James Weimer, Miroslav Pajic, and Insup Lee. Sensor attack detection in the presence of transient faults. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, pages 1–10, 2015.

[76] Jonathan Petit, Bas Stottelaar, Michael Feiri, and Frank Kargl. Remote attacks on automated vehicles sensors: Experiments on camera and lidar. *Black Hat Europe*, 11:2015, 2015.

[77] Kelsey Piper. *It's 2020. Where are our self-driving cars?*, 2020 (accessed June 29, 2020). `https://www.wsj.com/articles/amazon-to-acquire-self-driving-startup-zoox-11593183986`.

[78] Raul Quinonez, Jairo Giraldo, Luis Salazar, and Erick Bauman. Savior: Securing autonomous vehicles with robust physical invariants. *Usenix*, 2020.

[79] JA Romagnoli and George Stephanopoulos. Rectification of process measurement data in the presence of gross errors. *Chemical Engineering Science*, 36(11):1849–1863, 1981.

[80] Aviva Hope Rutkin. Spoofers use fake GPS signals to knock a yacht off course. *MIT Technology Review*, 2013.

[81] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 298–307, 2004.

[82] Prinkle Sharma, Jonathan Petit, and Hong Liu. Pearson correlation analysis to detect misbehavior in vanet. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–5. IEEE, 2018.

[83] Qikun Shen, Bin Jiang, Peng Shi, and Cheng-Chew Lim. Novel neural networks-based fault tolerant control scheme with fault alarm. *IEEE transactions on cybernetics*, 44(11):2190–2201, 2014.

[84] Jongho Shin, Youngmi Baek, Yongsoon Eun, and Sang Hyuk Son. Intelligent sensor attack detection and identification for automotive cyber-physical systems. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2017.

[85] Jongho Shin, Youngmi Baek, Jaeseong Lee, and Seonghun Lee. Cyber-physical attack detection and recovery based on rnn in automotive brake systems. *Applied Sciences*, 9(1):82, 2019.

[86] Yasser Shoukry, Paul Martin, Paulo Tabuada, and Mani Srivastava. Non-invasive spoofing attacks for anti-lock braking systems. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 55–72. Springer, 2013.

[87] Yunmok Son, Hocheol Shin, Dongkwan Kim, Youngseok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, and Yongdae Kim. Rocking drones with intentional sound noise on gyroscopic sensors. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 881–896, 2015.

[88] Timothy Trippel, Ofir Weisse, Wenyuan Xu, Peter Honeyman, and Kevin Fu. Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks. In *2017 IEEE European symposium on security and privacy (EuroS&P)*, pages 3–18. IEEE, 2017.

[89] Rohit Tunga, Carlos Murguia, and Justin Ruths. Tuning windowed chi-squared detectors for sensor attacks. In *2018 Annual American Control Conference (ACC)*, pages 1752–1757. IEEE, 2018.

[90] Zachariah Tyree, Robert A Bridges, Frank L Combs, and Michael R Moore. Exploiting the shape of can data for in-vehicle intrusion detection. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–5. IEEE, 2018.

[91] David I Urbina, Jairo A Giraldo, Alvaro A Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg. Limiting the impact of stealthy attacks on industrial control systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1092–1105, 2016.

[92] Franco van Wyk, Yiyang Wang, Anahita Khojandi, and Neda Masoud. Real-time sensor anomaly detection and identification in automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1264–1276, 2019.

[93] Abraham Wald. Sequential tests of statistical hypotheses. *The annals of mathematical statistics*, 16(2):117–186, 1945.

[94] Zichang Wang, Fei Guo, Yan Meng, Huaxin Li, Haojin Zhu, and Zhenfu Cao. Detecting vehicle anomaly by sensor consistency: An edge computing based mechanism. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2018.

[95] James Weimer, Radoslav Ivanov, Sanjian Chen, Alexander Roederer, Oleg Sokolsky, and Insup Lee. Parameter-invariant monitor design for cyber–physical systems. *Proceedings of the IEEE*, 106(1):71–92, 2017.

[96] Alan S Willsky. A survey of design methods for failure detection in dynamic systems. *Automatica*, 12(6):601–611, 1976.

[97] Lin Xiao, Stephen Boyd, and Sanjay Lall. A scheme for robust distributed sensor fusion based on average consensus. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 63–70. IEEE, 2005.

[98] Kun Xie, Xueping Ning, Xin Wang, Dongliang Xie, Jiannong Cao, Gaogang Xie, and Jigang Wen. Recover corrupted data in sensor networks: A matrix completion solution. *IEEE Transactions on Mobile Computing*, 16(5):1434–1448, 2016.

[99] Jining Yan, Lin Mu, Lizhe Wang, Rajiv Ranjan, and Albert Y Zomaya. Temporal convolutional networks for the advance prediction of enso. *Scientific reports*, 10(1):1–15, 2020.

[100] Man-Ki Yoon, Bo Liu, Naira Hovakimyan, and Lui Sha. Virtualdrone: virtual sensing, actuation, and communication for attack-resilient unmanned aerial systems. In *Proceedings of the 8th International Conference on Cyber-Physical Systems*, pages 143–154, 2017.

[101] Man-Ki Yoon, Sibin Mohan, Jaesik Choi, Jung-Eun Kim, and Lui Sha. Securecore: A multicore-based intrusion detection architecture for real-time embedded systems. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 21–32. IEEE, 2013.

[102] Lin Zhang, Xin Chen, Fanxin Kong, and Alvaro A. Cardenas. Real-time recovery for cyber-physical systems using linear approximations. In *41st IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020.

[103] Yunmin Zhu and Baohua Li. Optimal interval estimation fusion based on sensor interval estimates with confidence degrees. *Automatica*, 42(1):101–108, 2006.

VITA

VITA

Francis E. Akowuah was born in Kumasi, Ghana. He received his Bachelor of Science degree in Computer Science at Kwame Nkrumah University of Science and Technology (Kumasi, Ghana). He received his Masters of Science degree from North Carolina A & T State University. He is expected to receive his PhD in Computer & Information Science and Engineering from Syracuse University (Syracuse, New York, USA) in July 2021.