Learn-to-Respond: Sequence-Predictive Recovery from Sensor Attacks in Cyber-Physical Systems

Mengyu Liu
University of Notre Dame
mliu9@nd.edu

Lin Zhang
University of Pennsylvania
cpsec@seas.upenn.edu

Vir V.Phoha
Syracuse University
vvphoha@syr.edu

Fanxin Kong
University of Notre Dame
fkong@nd.edu

***Abstract*—While many research efforts on Cyber-Physical System (CPS) security are devoted to attack detection, how to respond to the detected attacks receives little attention. Attack response is essential since serious consequences can be caused if CPS continues to act on the compromised data by the attacks. In this work, we aim at the response to sensor attacks and adapt machine learning techniques to recover CPSs from such attacks. There are, however, several major challenges. i) Cumulative error. Recovery needs to estimate the current state of a physical system (e.g., the speed of a vehicle) in order to know if the system has been driven to a certain state. However, the estimation error accumulates over time in presence of compromised sensors. ii) Timely response. A fast response is needed since slow recovery not only comes with large estimation errors but also may be too late to avoid irreparable consequences. To address these challenges, we propose a novel learning-based solution, named sequence-predictive recovery (or SeqRec). To reduce the estimation error, SeqRec designs the first sequence-to-sequence (Seq2Seq) model to uncover the temporal and spatial dependencies among sensors and control demands, and then uses the model to estimate system states using the trustworthy data logged in history. To achieve an adequate and fast recovery, SeqRec designs the second Seq2Seq model that considers both the current time step using the remaining intact sensors and the future time steps based on a given target state, and embeds the model into a novel recovery control algorithm to drive a physical system back to that state. Experimental results demonstrate that SeqRec can effectively and efficiently recover CPSs from sensor attacks.**

***Index Terms*—cyber-physical systems, recovery, sensor attacks**

I. Introduction

Cyber-Physical Systems (CPS) incorporate computational and communication components with physical processes via sensing and actuation. Integrating cutting-edge technology increases CPS autonomy and facilitates applications like autonomous cars and drones [1], [2]. In the meanwhile, the development brings new security vulnerabilities that may be exploited by attackers with malicious intentions [3]–[8].

In this paper, we investigate one of the most significant security threats in CPS, which is called a sensor attack. Besides compromising control software and networks, attackers can also non-invasively modify sensor values via spoofing or transduction attacks [6], [9]–[12]. In this type of assault, an attacker attempts to interrupt the functionality of a physical system by manipulating with sensor data. Taking action based on compromised sensor data can drive the physical system into unsafe states and bring dangerous effects [10], [13]–[16]. For instance, an attacker may spoof GPS signals to misguide a vehicle [17]–[20], or use sound waves to affect accelerometers [21], [22].

These new threats have motivated many research efforts on sensor attack detection [9], [13], [23]–[30]. However, a critical question, 'how to respond to an attack after the detection of it', receives less attention. If the system continues to operate based on malicious sensor data, it will continue to drift. The response must stop the deterioration and even reverse the attack's detrimental effects [10], [15], [25], [31]. Despite this importance, much less attention is paid to attack response, compared to the rich literature of attack detection. Recent surveys such as [25], [26], [32]–[34] also confirm this significant research gap of attack response. Thus, this paper focuses on this sensor attack response problem.

Recently, researchers start to make more efforts on studying how to respond to sensor attacks after detection in CPS. Several attempts to this end use a common method, that is, to discard the corrupted sensors and then derive state estimates for them based on a pre-known mathematical model of the physical system (e.g., linear time-invariant model) [14], [35], [36] or a learned model [24], [37]. However, these works do not present a specific recovery control and instead continue to use the original controller to supervise the physical system, which thus may not yield a timely response. To solve this issue, a real-time attack recovery framework is proposed in [10], [15], where a specific recovery controller is developed to drive the physical system back to target states in a timely manner. However, these works are confined to model-based designs, i.e., rely on the knowledge of the mathematical system model, and thus is hard to apply to black-box systems. Also, it is challenging to apply the model-based methods to complex system in real-time due to the large computing overhead. Further, the recovery control here is different from conventional robust control. The latter is usually good at handling bounded errors but is incapable to handle sensor attacks which can incur unbounded modifications [38], [39].

In this paper, we aim to overcome these above limitations by jointly considering physical state estimation and recovery control. However, several key challenges remain unaddressed for the pursuit of learning-based recovery as response to sensor attacks in CPS.

• First, the recovery needs an accurate estimation of the current state of the physical system, e.g., the speed of a car

2023 IEEE Real-Time Systems Symposium (RTSS) | 979-8-3503-2857-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/RTSS59052.2023.00017

2576-3172/23/$31.00 ©2023 IEEE
DOI 10.1109/RTSS59052.2023.00017

and the altitude of a drone, in order to know if it has been driven to a certain state. The estimation error accumulates over time in presence of compromised sensors.

• Second, in terms of timing, a timely response is required. A slow recovery not only has large estimation errors, but also may be too late to avoid irreparable consequences. Both cases may result in a failed recovery in the end.

• Last, attack detection usually comes with a certain detection delay, i.e., the time interval between the onset of an attack and the detection of it. The recovery needs to accommodate this delay in the design [10], [15], [29], [40].

To address these challenges, we propose a novel learning-based framework, named sequence-predictive recovery (or SeqRec). SeqRec employs the encoder-decoder architecture to design two learning models: one for system state estimation and one for recovery control. Specifically, to reduce the estimation error, SeqRec has the first sequence-to-sequence (Seq2Seq) model to uncover the temporal and spatial dependencies among sensors and control demands. The model is used to estimate the physical system states using the trustworthy data logged in history. To achieve an adequate and fast recovery, SeqRec has the second Seq2Seq model that considers both the current time step using the remaining intact sensors and the future time steps based on a given target state. The model is then embedded into a novel control algorithm to generate a sequence of control demands to drive the physical system back to the target state. SeqRec is a model-free method without knowing the system dynamics details. And it can be applied to complex systems in real-time since the learning models are good at capturing the complex dynamics of the system and the inference speed are short due to the boosting development of graphic cards. Finally, we evaluate the proposed SeqRec using multiple simulators including linear and non-linear numerical benchmarks as well as a high-fidelity simulator, SITL with ArduPilot [41]. Moreover, we implement SeqRec on a 4-wheel vehicle testbed to show its effectiveness. Specifically, the contribution of this work is as follows:

• We propose a novel sequence-predictive recovery framework to respond to sensor attacks in CPS. We propose a Seq2Seq model for the system state estimation to reduce the estimation error in presence of sensor attacks. We propose another Seq2Seq model to achieve fast recovery. And we embed the model into a novel sequence-predictive recovery control algorithm to drive the system back to target state.

• We conduct extensive simulations and experiments on various CPSs with varying levels of complexity and nonlinearity to demonstrate the effectiveness and efficiency of SeqRec at recovering CPS from sensor attacks. The proposed method is also implemented on a 4-wheel testbed.

The rest of this paper is organized as follows. Section II presents preliminaries. Section III gives an overview of our recovery framework. Section IV and Section V describe the two Seq2Seq models and the control algorithm. Section VI describes the experiment settings. Section VII evaluates our method. Section VIII discusses the scope of application. Section IX concludes the paper.

II. Preliminaries

In this section, we first present the system and threat models, and then briefly describe the encoder-decoder architecture of Seq2Seq learning and Recurrent Neural Networks.

A. System Model

We consider a CPS that consists of a physical system controlled by a computer program or controller. The controller operates at a predetermined time interval, called a control step. At each control step t, the controller reads sensor readings and estimates the current state of the physical system or the physical state. The physical state is denoted by a vector of real variables $\boldsymbol{x}(t) = \{x_1(t), \ldots, x_n(t)\}^T$, where n is the dimension of the vector. Then, the controller computes control demands, denoted as $\boldsymbol{u}(t) = \{u_1(t), \ldots, u_m(t)\}^T$, using a control algorithm $\boldsymbol{u}(t) = g(\boldsymbol{x}(t), \boldsymbol{x}_r(t))$, where m is the dimension of the control demand vector, $\boldsymbol{x}_r$ is the pre-defined reference or target state, g is the control law of the controller. Then the control demands are applied to the actuators which drive the physical system to follow the target state. For ease of presentation, we assume that the physical system is fully observable to the sensors, i.e. all state estimates can be obtained from sensor measurements. Further, we use $\boldsymbol{x}$, $\tilde{\boldsymbol{x}}$, and $\hat{\boldsymbol{x}}$ to denote the real physical state, the observed sensor measurements, and the estimated physical state by our Seq2Seq model, respectively.

B. Threat Model

The threat model considered in this paper is as follows. An attacker can launch sensor attacks by compromising the integrity (e.g., spoofing and transduction attacks) and availability (e.g., DoS attacks) of sensor data or measurements.

Acting on the corrupted values, the controller may drift the physical system from the target state to unsafe states. Formally, sensor attacks make $\boldsymbol{x}(t) \neq \tilde{\boldsymbol{x}}(t)$, and the difference is denoted as $\boldsymbol{e}(t) = \boldsymbol{x}(t) - \tilde{\boldsymbol{x}}(t)$. The sensors can be partially or fully affected, that is, the number of non-zero dimensions of $\boldsymbol{e}_t$ or l_0 norm of $\boldsymbol{e}_t$ is $0 < ||\boldsymbol{e}_t||_0 < n$, or $||\boldsymbol{e}_t||_0 = n$. Further, we assume that there is a sensor attack detector already in place and it can identify which sensors are under attack. The detector may have some detection delay, but the detector is responsible to diagnose when did the attack start.

Given that attack detection is not our focus, exiting detection methods such as [9], [13], [40] satisfy this assumption and thus can be applied. Note that the capability of recovery heavily depends on the attack detector, e.g., if it fails to detect an attack, the recovery will not be activated and thus offer no help to defend against the attack. Further, this work pursues a black-box solution and needs little knowledge of the mathematical model or dynamics of the physical system. It is possible that the system diverges again after the recovery. This work focusing on how to recover the physical system to the target state. For example, if a car's IMU is attacked, we can recover it and drive it to the shoulder and stop there. There is a very important question to be noticed: how to control the system after the recovery? This question is out of scope

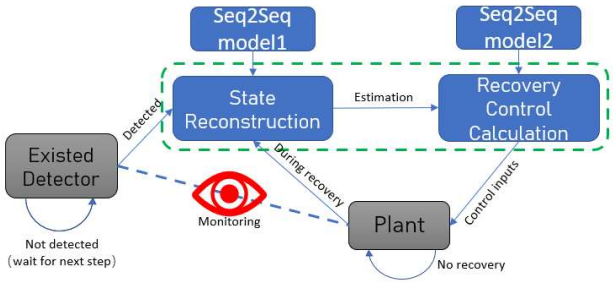


Fig. 1: Illustration of the SeqRec Framework.

of this paper, our recovery ends when the current estimate falls into the target set. The estimation error is small for low recovery time which will be shown in VII. Even if it was large, our recovery still gives cushion time for the operator (e.g. the driver) to react. Furthermore, we assume the detector keeps running. When no attack is detected anymore, the controller resumes to use observed values.

C. Introduction on Seq2Seq Learning

Encoder-Decoder Architecture. The encoder-decoder architecture is typically designed for sequence-to-sequence learning. A neural network under this architecture consists of two main parts: the encoder and the decoder. The encoder represents a sequence of input features as a hidden state vector, which then is forwarded to the decoder to generate an output sequence. In general, the encoder is composed of a stack of recurrent units, each of which accepts an element of the input sequence, represents the element as a hidden state, and forwards it to the next unit. The hidden vector is the final hidden state produced by the encoder, which aims to encapsulate the information for all input features so as to help the decoder make better predictions. The decoder also contains a stack of recurrent units, each of which accepts a hidden state from the previous unit and produces an output element and its own hidden state.

Recurrent Neural Networks. The encoder/decoder can be a Recurrent Neural Network (RNN). Simple RNNs suffer from the problem of vanishing gradients, which makes it difficult to learn long sequences. To solve this problem, Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) were proposed, which are the most widely used RNNs nowadays.

GRU cells have fewer parameters than and different gating design from that of LSTM. A GRU cell maintains only a hidden state and no cell state. Please refer to [42], [43] for more details on Seq2Seq learning.

III. Sequence-predictive recovery framework

This section presents the sequence-predictive recovery framework or SeqRec, and the idea of using recurrent units and the encoder-decoder architecture in SeqRec.

Framework Overview. Fig. 1 illustrates key components of SeqRec. SeqRec has two key components: state estimation and recovery control. Attacked sensors may incorrectly reflect the physical state. Thus, during recovery, the corrupted sensors are not used and the first component estimates the physical state in presence of them. Both the historical sensor data and control commands are required to do the state estimation. Due to the detection delay, the physical system may already drift away from the target state when the attack is detected. Therefore, the recovery requires the reconstructed states as inputs since the compromised sensor readings are not trustful anymore. Thus, the second component computes a control sequence to drive the physical system back to the target state. The recovery continues until the system is back to the target stare.

We assume there is an existed detector in the system monitoring the sensors of the physical plant. When the detector detects attacked sensors, SeqRec is activated to supervise the system; otherwise, the original control does the supervision and the procedures in the dashed-green box are not processed at this time. Note that though SeqRec follows the simplex architecture [10], [15], [44]–[46], the design of the recovery controller is different. Specially, we propose to use Seq2Seq learning models for the design.

Framework Workflow. The workflow of SeqRec is as follows. As shown in Fig. 2, suppose that a sensor attack starts at time t_0 and is detected at time t_a, i.e., the detection delay is $t_a - t_0$. Before time t_0, the system runs normally and the measurements are trustworthy, i.e., uses the original controller, and tracks the target state, as shown by the black curve. After time t_0, the system starts drifting and the measurements of attacked sensors are not trustworthy, as shown by the red curve. From time t_a to t_d. SeqRec takes over to control and drives the physical system back to the target set. The blue curve shows the recovery trajectory between t_a and t_d.

Idea of Using Seq2Seq Learning Models. The motivation of this idea, i.e., using recurrent units and the encoder-decoder architecture, is as follows. The first is to reduce the estimation/prediction error in state estimation and recovery control. If the prediction is made step-by-step, its error will accumulate very fast. By contrast, a sequence of recurrent units in the encoder/decoder can better capture the temporal and spatial dependencies among sensor measurements and control demands, and thus reduces the error accumulation. Although some papers such as [47]–[51] also use recurrent units and sequence learning for state estimation, they do not consider control demands and thus have degraded performance. Sec VII will show the gain and cost of including control demands to the sequence learning. Second, for the recovery control, a sequence of recurrent units in the decoder allows looking ahead to the future time points. That is, for each time of prediction, a sequence of control demands will be computed and the demands are related to each other. Compared to a step-wise controller, i.e., computing a single control demand at each control step, this look-ahead control will help on timely recovery, and also allows to include the intact sensors in each time of prediction. To the best of our knowledge, this method is the first sequence-predictive model-free method to recover CPSs. Note that the performance gain by this idea is also validated in the evaluation. The novel designs of realizing this

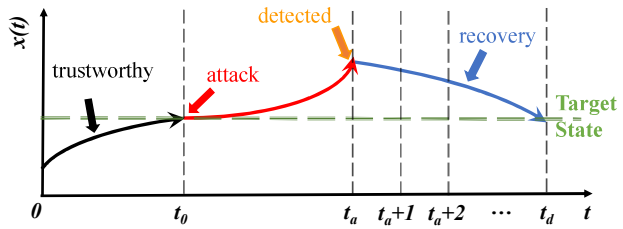


Fig. 2: Workflow of the SeqRec Framework.

idea are detailed in Section IV and Section V.

IV. Recovery Control Design

In this section, we detail the design of the recovery control component in SeqRec. This component needs the estimated system states obtained from the state estimation component, which will be detailed in the next section. The following first presents our novel sequence-predictive control algorithm then we explain how to use Seq2Seq model for recovery control.

A. Sequence-Predictive Control Algorithm

Our control algorithm mimics the principle of model-predictive control (MPC) (details on MPC in [52]). At each control step, given the state estimate, MPC looks ahead and generates control demands for multiple steps in future, but only the control demand at the current step will be applied to the actuator. MPC relies on the knowledge of the mathematical model in order to formulate an optimization problem, solving which gives the control demand sequence. By contrast, our algorithm is a black-box solution and uses the Seq2Seq model designed above. That is, at each control step, our algorithm uses the Seq2Seq model to generate a sequence of control demands and then applies only the control demand at the current step. Doing this allows us to use sensor measurements at each step. However, given attacked sensors, their measurements cannot be used. The difficulty here is how to utilize the intact sensors in presence of those corrupted ones.

To address this difficulty, we propose to replace the compromised sensor measurements by the corresponding state estimates from the state estimation component. The idea is described as follows. Without loss of generality, suppose that q sensors are compromised, i.e., $\{\tilde{x}_1(t), \cdots, \tilde{x}_q(t)\}$ is not trustworthy, and $\{\tilde{x}_{q+1}(t), \cdots, \tilde{x}_n(t)\}$ is. We first use the state estimation component (detailed in the next section) to estimate the current system state $\hat{\boldsymbol{x}}(t)$. Then we assemble the following vector $\overline{\boldsymbol{x}}$ as the state estimate for time t.

$$\overline{\boldsymbol{x}}(t) = \{\hat{x}_1(t), \cdots, \hat{x}_q(t), \tilde{x}_{q+1}(t), \cdots, \tilde{x}_n(t)\}^T. \quad (1)$$

That is to abandon the q compromised dimensions in $\tilde{\boldsymbol{x}}(t)$ and replace them by that in $\hat{\boldsymbol{x}}(t)$. Finally, we do the above assembling for all steps in the encoder sequence and will obtain

$$\overline{\boldsymbol{X}}_p(t) = \{\overline{\boldsymbol{x}}(t - n_e + 1), \cdots, \overline{\boldsymbol{x}}(t-1), \overline{\boldsymbol{x}}(t)\}. \quad (2)$$

Algorithm 1: Sequence-Predictive Control

Input: $n_e, n_d, m_e, m_d, t_0, t_a, q, R(), S()$
/* n_e, n_d: length of the encoder and decoder of the recovery control model in Fig. 3; */
/* m_e, m_d: length of the encoder and decoder of the state estimation model in Fig. 4; */
/* t_0, t_a, q: attack start time, detection time, and the number of compromised sensors, respectively, given by the detector; */
/* $n_e = m_d = t_a - t_0$; */
/* $R(), S()$: the recovery control and state estimation models in Fig. 3 and Fig. 4; */
Output: $\boldsymbol{u}(t_a + k)$
/* $\boldsymbol{u}(t_a + k)$: the control demand at time $t_a + k$; */
1 $\widehat{\boldsymbol{X}}_p(t_a + k) = S\left(\left(\overline{\boldsymbol{X}}_p(t_a + k - m_d), \boldsymbol{U}_p(t_a + k - m_d)\right); \boldsymbol{U}_p(t_a + k)\right)$;
2 Assemble $\overline{\boldsymbol{X}}_p(t_a + k)$ using Eq. (1) and Eq. (2);
3 $\boldsymbol{U}_r(t_a + k) = R\left(\left(\overline{\boldsymbol{X}}_p(t_a + k), \boldsymbol{U}_p(t_a + k)\right); \boldsymbol{X}_r(t_a + k)\right)$;
4 **return** $\boldsymbol{u}(t_a + k)$;

$\overline{\boldsymbol{X}}_p(t)$ in the above equation will be used as $\boldsymbol{X}_p(t)$ input to the encoder in Fig. 3. Hence, now using the Seq2Seq model above can generate control demands in presence of corrupted sensors.

Based on the idea above, we present our sequence-predictive control algorithm as shown in Algorithm 1. Line 1 estimates the physical state using the state estimation model $S()$ in the next section. The encoder sequence length, i.e., that of $\overline{\boldsymbol{X}}_p(t_a + k - m_d)$ and $\boldsymbol{U}_p(t_a + k - m_d)$, is m_e. $\overline{\boldsymbol{X}}_p(t_a + k - m_d)$ is also obtained by Eq. (1) and Eq. (2). The decoder sequence length estimates, i.e., the length of estimated states $\widehat{\boldsymbol{X}}_p(t_a + k)$, is m_d. Line 2 assembles $\overline{\boldsymbol{X}}_p(t_a + k)$ whose length is n_e. Note that our design lets $n_e = m_d = t_a - t_0$, i.e., the detection delay. Line 3 uses the recovery control model $R()$ to generate a control demand sequence $\boldsymbol{U}_r(t_a + k)$, starting at $t_a + k$ and with a length of n_d. Line 4 returns the first control demand in $\boldsymbol{U}_r(t_a + k)$, which is $\boldsymbol{u}(t_a + k)$ and then applied to the actuator.

B. Seq2Seq Model for Control

Problem Description. The focused problem here is to compute control demands that supervise the physical system to follow the target state. We formulate it as a multivariate time series forecasting problem. That is, given a sequence of state estimates and control demands in history, the objective is at the current control step, to generate a sequence of control demands for future steps.

Design. Fig. 3 depicts the design of the Seq2Seq model for generating control demands. The model employs the encoder-

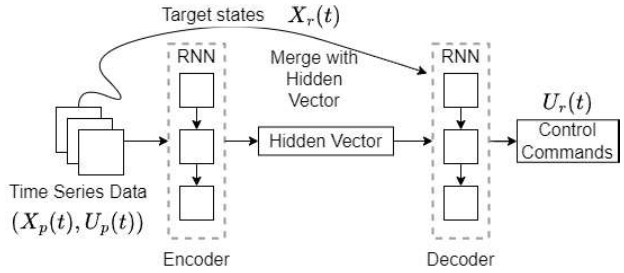


Fig. 3: Sequence-predictive Recovery Seq2Seq Model.

decoder architecture as mentioned above. For the encoder, it consists of a sequence of recurrent units (such as GRU and LSTM). At the current time step t, its input, denoted as $(\boldsymbol{X}_p(t), \boldsymbol{U}_p(t))$, is the historical or past state estimates and control demands between time $t - n_e$ and time t. That is,

$$(\boldsymbol{X}_p(t), \boldsymbol{U}_p(t)) = \{ (\tilde{\boldsymbol{x}}(t-n_e+1), \boldsymbol{u}(t-n_e+1)), \cdots, (\tilde{\boldsymbol{x}}(t-1), \boldsymbol{u}(t-1)), (\tilde{\boldsymbol{x}}(t), \boldsymbol{u}(t)) \},$$

where n_e is the length of the encoder sequence. The encoder captures the dependencies between state estimates and control demands as well as among control steps. Each recurrent unit $\boldsymbol{h}_e(k)$, $k \in (1, n_e]$ updates its state vectors based on the input and the previous hidden states passed into it. The hidden state of the last unit, $\boldsymbol{h}_e(n_e)$, can be considered to represent the summary of the input sequence. This hidden vector is then forwarded to the decoder. For the decoder, it also consists of a sequence of recurrent units. At the current time step t, its input, denoted as $\boldsymbol{X}_r(t) = \{\boldsymbol{x}(t+1), \boldsymbol{x}(t+2), \cdots, \boldsymbol{x}(t+n_d)\}$, is a sequence of target states for the subsequent steps in future, where n_d is the length of the decoder sequence. This sequence gives a reference for the model to achieve, in other words, we want the Seq2Seq to output a serie of control signals can make the system reach the targets. The hidden vector together with this input will guide the decoder to output a sequence of control demands, denoted as $\boldsymbol{U}_r(t) = \{\boldsymbol{u}(t), \boldsymbol{u}(t+1), \cdots, \boldsymbol{u}(t+n_d-1)\}$, that can recover the system to target states. For clarity of notation, subscript "p" is for the past control steps and "r" is for the future control steps, as to time t.

For ease of presentation, we abstract the recovery control model in Fig.3 as a function $R()$, i.e., $\boldsymbol{U}_r(t) = R((\boldsymbol{X}_p(t), \boldsymbol{U}_p(t)); \boldsymbol{X}_r(t))$.

V. State Estimation Design

In this section, we detail the design of the state estimation in SeqRec. The following describes the focused problem and then presents the Seq2Seq model for state estimation.

Problem Description. The focused problem is to estimate the physical states, and is also formulated as a multivariate time series forecasting problem. That is, given a sequence of state estimates and control demands in history, the objective is to produce a sequence of state estimates from a past control step to the current step.

Design. Fig. 4 shows the design of the Seq2Seq model. This model also employs the encoder-decoder architecture as discussed above. Although this model looks similar to that in Fig. 3, there are several major differences as follows. First, the encoder consists of a sequence of recurrent units, and at the current step t, its input, denoted as $(\boldsymbol{X}_p(t-m_d), \boldsymbol{U}_p(t-m_d))$, is a sequence of past state estimates and control demands between step $t - m_d - m_e$ and step $t - m_d$. That is,

$$(\boldsymbol{X}_p(t-m_d), \boldsymbol{U}_p(t-m_d)) = \{ (\tilde{\boldsymbol{x}}(t-m_d-m_e+1), \boldsymbol{u}(t-m_d-m_e+1)), \cdots, (\tilde{\boldsymbol{x}}(t-m_d-1), \boldsymbol{u}(t-m_d-1)), (\tilde{\boldsymbol{x}}(t-m_d), \boldsymbol{u}(t-m_d)) \},$$

where m_e and m_d are the lengths of the encoder and decoder, receptively. Second, the model here estimates system states till the current time t and does not predict for future control steps. The decoder also includes a sequence of recurrent units, and at the current step t, its input, denoted as $\boldsymbol{U}_p(t)$ is a sequence of control demands from time $t - m_d + 1$ to time t. That is, $\boldsymbol{U}_p(t) = \{\boldsymbol{u}(t-m_d+1), \cdots, \boldsymbol{u}(t-1), \boldsymbol{u}(t)\}$. The output of the decoder is $\widehat{\boldsymbol{X}}_p = \{\hat{\boldsymbol{x}}((t-m_d+1), \cdots, \hat{\boldsymbol{x}}((t-1), \hat{\boldsymbol{x}}((t)\}$. Note that we focus on sensor attacks and all control demands are trustworthy. If we let m_d be equal to the detection delay, then the input sensor measurements to the encoder are also trustworthy. Thus, all state estimates during the detection delay can be generated by this model, which are then used to replace the compromised sensor measurements. For ease or presentation, we abstract the state estimation model in Fig. 4 as a function $S()$, i.e., $\widehat{\boldsymbol{X}}_p(t) = S((\boldsymbol{X}_p(t-m_d), \boldsymbol{U}_p(t-m_d)); \boldsymbol{U}_p(t))$, which is already referred to in Algorithm 1.

Several points worth to note are follows. The first is the problem of how to log trustworthy sensor measurements, state estimates, and control demands. The logging protocols presented in [10], [15], [29] can be applied here. The logging protocols use checkpoints to make the historical system data trustworthy. Second, applying Seq2Seq models for state estimation can mitigate the accumulative error compared to single-step prediction models. This fact is also validated in the evaluation. Third, the Seq2Seq model here can learn the correlation among sensor measurements/state estimates and control demands. In other words, the model can learn the system dynamics of the system. And this is also the reason why our method does not require attack data during the training. Therefore, our method is different from the methods in [40], [50] though similar neural networks have been applied. It is possible to use only sensor measurements/state estimates, without control demands, for estimation or prediction such as [40], [50]. However, this method comes with lower estimation accuracy. We have compared the performance of SeqReq with that of above methods which do not use the control demands in section VII-B.

VI. Experimental Settings and Implementation

The experiments are tested on three numerical CPS simulators [53]–[55], a high-fidelity drone simulator [41] and a 4-wheel testbed. We collected data under normal operation from each simulator and the testbed and inject sensor attack

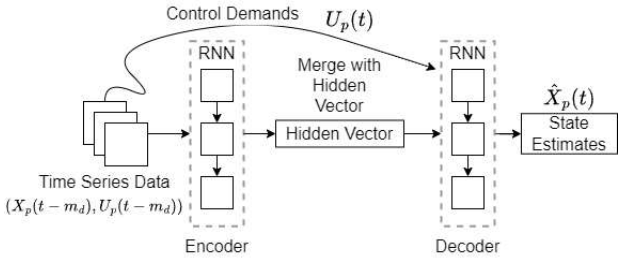


Fig. 4: State Estimation Seq2Seq Model.

in the testing phase, the details of each system and its dataset are given below. Additionally, the hardware and software configuration and the neural network hyper-parameters are described in details.

Hardware and Software Configurations The SeqRec models for numerical simulators and the high-fidelity simulator are running on a powerful PC with an Intel(R) Core(TM) i7-10700KF CPU @ 3.80GHz, a Nvidia GeForce GTX 3080 GPU and 64 GB RAM. Our implementation uses Python with Keras API under Tensorflow deep learning framework [56].

A. Numerical CPS Simulators

We use two linear simulators including DC motor position (DC) and Quadcoptor (Quad), and one non-linear simulators, Inverted Pendulum on Cart (IPC).

- DC is to maintain the rotation angle of the motor shaft. The system state has two dimensions: $x_1 =$ rotation angle and $x_2 =$ rotary angular velocity, and the control demand is $u =$ voltage given to the motor.
- Quad is to supervise its altitude. The system state has 12 dimensions: $(x_1, x_2, x_3), (x_4, x_5, x_6) =$ linear and angular position, and $(x_7, x_8, x_9), (x_{10}, x_{11}, x_{12}) =$ linear and angular velocity, and the control demand is $u =$ thrust. The original controls(Orig) of the two simulators above are both PID controller.
- IPC is to balance an inverted pendulum on a cart. The system state has four dimensions: $x_1 =$ angle between the pendulum and the horizontal direction, $x_2 =$ location of the cart, $x_3 =$ velocity of the cart, and $x_4 =$ velocity of the pendulum, and the control demand is $u =$ the horizontal force. The original controller of IPC is a linear–quadratic regulator (LQR). Details of these simulators can be found in [53]–[55].

Dataset. The datasets are collected from the running results of the three simulators above. For each simulator, 20,000 data points are collected and the dataset is split into 60/20/20 proportions for training/validation/testing. Further, we do the data preprocessing as follows. The scale of each state dimension and control demand may vary, and thus a normalization is used before training. We rescale all data to the range $[-1, 1]$ using the following normalization, $\mathbf{x}_{norm} = (\mathbf{x} - \mathbf{x}_{mean})/(\mathbf{x}_{max} - \mathbf{x}_{min})$.

B. High-fidelity Simulator

To further validate the effectiveness and efficiency of our method, we implement our SeqRec on the high fidelity simulator, SITL with ArduPilot [41]. ArduPilot can provide control support for simulators such as SITL, Gazebo, Xplane, etc. ArduPilot has been applied in various research fields such as adaptive control, security, CPS, etc. SITL can simulate various scenarios for real-world CPS systems such as quadcopters and rovers. The environment factors, such as noise, wind and vibration, can be also considered in the settings to maintain high fidelity.

1) Setting: In this experiment, we simulate a copter as marked in Fig. 10. ArduPilot applies multiple controllers to keep the system stable. To facilitate setting attacks and recovery, we change some default parameters of the simulator. First, the frequency is set to 100Hz, i.e., the control step is 10 milliseconds. Second, ArduPilot uses a PID controller to supervise the copter's altitude, i.e., to reduce the gap between its altitude and the target altitude. We increase the p of the PID to make the copter more responsive to attacks and recovery.

Dataset. The copter is equipped with IMU and GPS sensors and four motors as actuators. We collect the sensor readings and control demands as follows. For the dateset, we collect 80,000 continuous samples from the copter model in the SITL simulator. For each sample in the dataset, the control demand has five dimensions including one for each motor (there are four motors) and one for the main control. The system state has 15 dimensions including three for the copter's position, three for the velocity on x,y,z-axis respectively, yaw, pitch, roll, the angles of yaw, pitch, and roll, and the angular velocities of yaw, pitch, and roll. There is no attack happened when we collect the training data, i.e., all the training data are normal data. These data are in different controller settings that have different values for the p parameter. Its range is from 1 to 3 and a larger p value means a more aggressive PID controller.

C. 4-wheel testbed

In this subsection, we will cover the implementation details and configurations of the 4-wheel vehicle testbed.

1) Settings: As illustrated in Fig. 5 , the vehicle testbed is composed of four major components: a STM32F4 board, a Raspberry Pi Model 4B, a motor, and a servo. The speed sensor and voltage sensor were embedded on the STM32 board. The control period of this testbed is 50ms which is sufficient to tolerate the computation overhead. Different from the numerical simulators and the high-fidelity simulator, the Seq2Seq models were deployed on a lab computer with 16GB RAM, Intel i5-9600 3.1GHz cpu and a NVIDIA GTX1660 GPU. There are 2 PID controllers on the testbed to control the speed and turning of the vehicle.

Fig. 5 shown the basic architecture of the testbed. A virtual local area network(VLAN) was created by ZeroTier and we use ROS2 framework for the communication between the boards and the lab computer through WiFi. There is a camera and a indoor positioning block on the testbed, the camera captures pictures in real-time and run a vision algorithm to

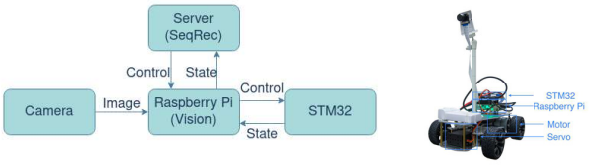


Fig. 5: Testbed Design

process them on the Raspberry Pi to maintain the car running through a straight line. We did not use the servo readings and the images for our model. The lab computer served as the server which run the SeqRec at this time, it reads the states message sent from the Raspberry Pi and send control messages to it. The Raspberry Pi is the core of the communication and holding the simple vision algorithm. The STM32 board is in charge of the testbed's real-time control and read senor values. For the turning experiment, we replace the camera with a indoor locating module to keep the testbed at the center of the road. We place locating four modules at the corners of the track as fixed anchor points, and the position of the testbed can be efficiently computed.

Fig. 6 shown an example recovery process on the testbed. We selected 6 frames from the demo video in the additional materials to show the recovery workflow on the testbed. The points in the plot represent the speed of the testbed at that frame. The first frame is not under attack, the second and the third frame is undering attack and the attack was not detected yet. The fourth frame is during recovery, the fifth and the last frame is after recovery. The data points are the approximation of the testbed speed since it is unavailable to match the frames and the collected with no error.

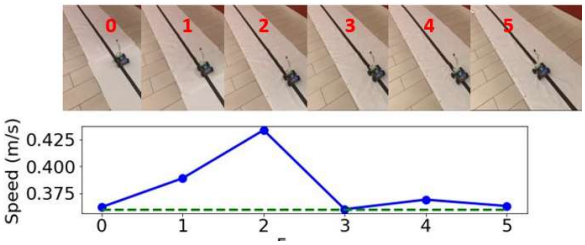


Fig. 6: Testbed Recovery Example

Dataset. The testbed equipped with IMU, gryoscope, 2 motors and a servo for turning. The system state have 10 states, the throttle(the average of the left motor and left motor), the speed on the x,y,z-axis, the acceleration on the x,y,z-axis, the pictch, yall, roll. For the second case study, we also collected the 2D-coordinates provided by the indoor localization modules. A dataset of 20000 continuous samples was collected from the testbed.

Scenarios. We consider two scenarios for the testbed: First, the testbed is controlled to run at a constant speed 0.36m/s. And we inject a bias attack which is -0.06m/s to the speed sensor measurement. The attack starts after the 200 control steps when the testbed already reaches the speed 0.36m/s. Second, the testbed is controlled to run at the center of the track, we inject bias attack to the readings from the localization modules. The goal of this attack is to make the testbed deviate from the center of the track. The recovery starts a number of control steps after the attack starts, where the number is the length of the sequence in our models.

D. Learning Configuration

All experiments are conducted using recurrent network models with a single hidden layer, a recurrent layer in the encoder and a recurrent layer in the decoder. The Exponential Linear Unit (ELU) is used as the activation function for σ_c in LSTM cells. For training, we use the Mean Absolute Error (MAE) as our loss function. Note that we also try the Mean Square Error (MSE), but it turns out that MAE is more robust in our experiment. Further, all training data are normal data, not attacked data, since our models are general to learn how to estimate states and control the physical system instead of confining to certain attacks. Furthermore, each model is trained for 100 epochs with a learning rate of 0.001 using Adam optimizer with $beta1 = 0.9$, $beta2 = 0.999$, which are the initial decay rates used when estimating the first and second moments of the gradient and $epsilon = 1e - 08$. We apply batch normalization with momentum of 0.6 after the last cell state and hidden state of the encoder. For fair comparison, dropout rate of the linear transformation of the inputs layer and dropout rate of linear transformation layer of the recurrent state in recurrent layers are set to 0.2, hidden size h, which is the number of LSTM units in each cell, is trivally set to 100 for all experiments.

VII. Performance Evaluation

In this section a brief description of the performance metrics is given, followed by the recovery performance and real-time property evaluation of the proposed method SeqRec.We compare multiple approaches including existing ones and the variants of our proposed method.

The performance metrics we consider include i) recovery performance: how well and how fast the system recovered to the target state, ii) state estimation and recovery accuracy, and iii) overhead of our Seq2Seq models. First, to the best of our knowledge, there is no previous work like our sequence-predictive control, therefore, we make comparison between the methods as follows.

- **Orig**. This method uses the original controller with state estimates generated by our state estimation model in Fig. 4 as input to the controller, this technique also know as virtual sensor which provides readings or estimates of quantities without relying on direct measurements from physical sensors.
- **SeqRec**. Our sequence-predictive recovery proposed in this paper. The recovery control is produced by a Seq2Seq model based on the reconstructed state estimated by another Seq2Seq model.

- **NoRec.** No recovery is applied after the detection, i.e., the attack continues to affect the system.

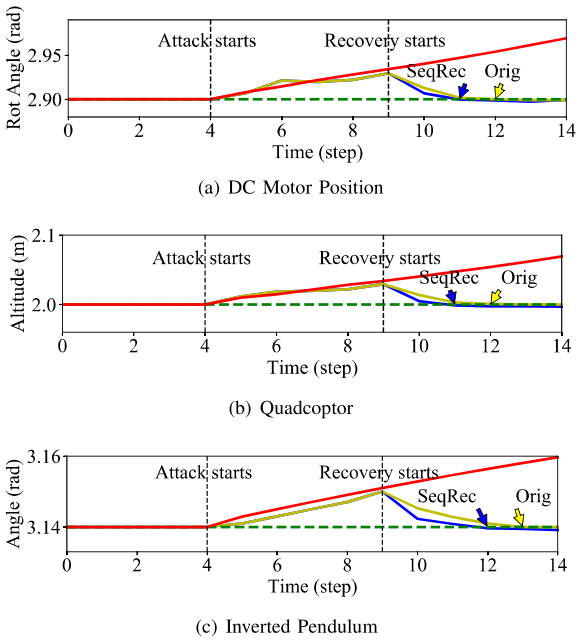
Second, we compare different types of recurrent units including (simple) RNN, LSTM and GRU. Third, we compare our state estimation method to existing ones such as [40], [50] which do not use control demands. Fourth, we also conduct sensitivity analysis, for example, comparing different lengths of sequences in our Seq2Seql models. Single-step LSTM is also considered whose sequence length can be seen as one.

A. Recovery Performance

Original Controller vs SeqRec. The original controller(Orig) is the existed controller in the CPS such as PID. We don't compare our recovery performance with the methods in [10], [15] since they assume the system dynamics is known which is different from our model-free assumption. A continuous bias attack (i.e., adding perturbation to sensor measurements) is injected to the rotation angle sensor in DC, altitude sensor in Quad, and the angle sensor in IPC. The attack runs from time 4 to 14. Note that the attack pattern is not important since we focus on recovery, not detection, and the compromised sensors are not used. In other words, SeqRec does not change its procedures due to the attack patterns, it only use trustworthy historical data. At time 9, the attack is detected and then the recovery starts. Fig. 7 shows the results, where blue, golden and red lines represent the state trajectories of SeqRec, Orig, and NoRec, respectively. We only plot till time 14 since as noted above, how to control the system after recovery is out of scope of this paper.

Several key observations are as follows. First, though both our SeqRec and Orig can drive the system back to the target set, it is worth to note that our method results in a faster recovery, as shown by the arrows in the figure. The reason is that SeqRec synthesizes a controller that captures the individual system dynamics instead of a general controller, and also the target state sequence is directed used as the input for the decoder. Second, by NoReq, i.e., no recovery after detection, the system continues to drift further, which validates the importance of attack response. Third, our state estimation model can well estimate the system states, as shown by the lines between time 4 and 9. More results on our Seq2Seq model accuracy follow the similar trend.

For the high-fidelity drone simulator, we consider a scenario that the copter is controlled to fly at a constant altitude of 100 meters or 100m. We inject a bias attack to the GPS z-axis, i.e., the altitude, which is -20m perturbation to the sensor. For example, if the sensor measurement is 100m, the sensor value given to the controller is 80m. The attack starts from when the copter reaches the altitude of 100m. The recovery starts a number of control steps after the attack starts, where the number is the length of the sequence in our models. In Fig. 10, we annotate the copter on the map, and also mark the current altitude above ground level (AGL) by the red boxes. The three consoles show the running status of the copter for the three cases. The target state/altitude is 100m, and the sequence lengths of both the encoder and decoder are set as 40. First, for



(a) DC Motor Position

(b) Quadcoptor

(c) Inverted Pendulum

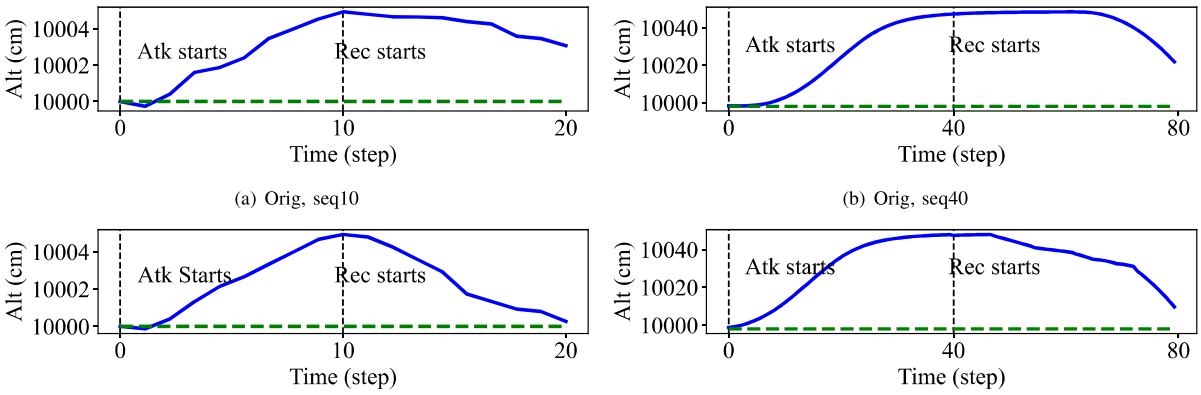
Fig. 7: Recovery Comparison. Blue line: SeqRec; Golden: original controller; Dotted green: target state; Red: NoRec

no recovery, the AGL will increase to over 110m after some time, which validates the importance of attack response, and it keeps increasing for there is no response to the sensor attack. Second, we can see that both Orig and SeqRec can recover the copter to the target altitude. For Orig, the exact current AGL reading is 99.67m and the last updated value is 100.17m. For SeqRec, the exact current AGL reading is 99.83m and the last updated value 100.03m. Note that the values are not 100m sharp because of the noise we put in the simulation. SeqRec can better supervise the altitude, i.e., closer to 100m, than Orig, even with the noise.

Fig. 8 shows the recovery trajectories for Orig and SeqRec with two different sequence lengths (both encoder and decoder) of 10 and 40. We can see that our SeqRec can faster recover the system than Orig. For example, comparing Fig. 8(a) and Fig. 8(c), SeqRec already drives the copter to the target set, 100m, at 20th control step, while Orig does not.

For the 4-wheel testbed, we consider two scenarios: First, the testbed is controller to run at a constant speed 0.36m/s. And we inject a bias attack which is -0.06m/s to the speed sensor. The attack starts after the 200 control steps when the testbed already reaches the speed 0.36m/s. The recovery starts a number of control steps after the attack starts, where the number is the length of the sequence in our models. Second, we inject a bias attack with -0.5m the locating module readings which contains the deviation from the center of the road, at this time, speed sensor is not attacked. The servo is controlled by a stanley controller for the turning of the testbed.

Fig. 11 shows the recovery comparison between SeqRec and Orig for the first scenario. From the $200th$ timestep, a



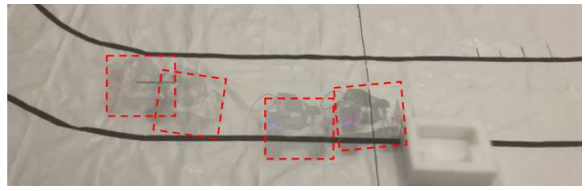
(a) Orig, seq10

(b) Orig, seq40

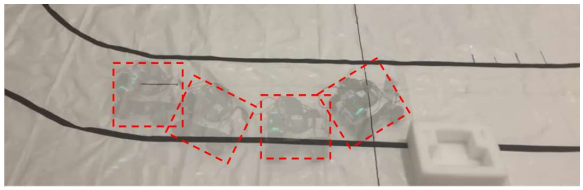
(c) SeqRec, seq10

(d) SeqRec, seq40

Fig. 8: Recovery comparison between SeqRec and Orig on High-fidelity Simulators. Green dotted line: target state; Blue: recovery trajectory. Atk: attack; Rec: recovery. seq#: the sequence length is #.



(a) Original Controller



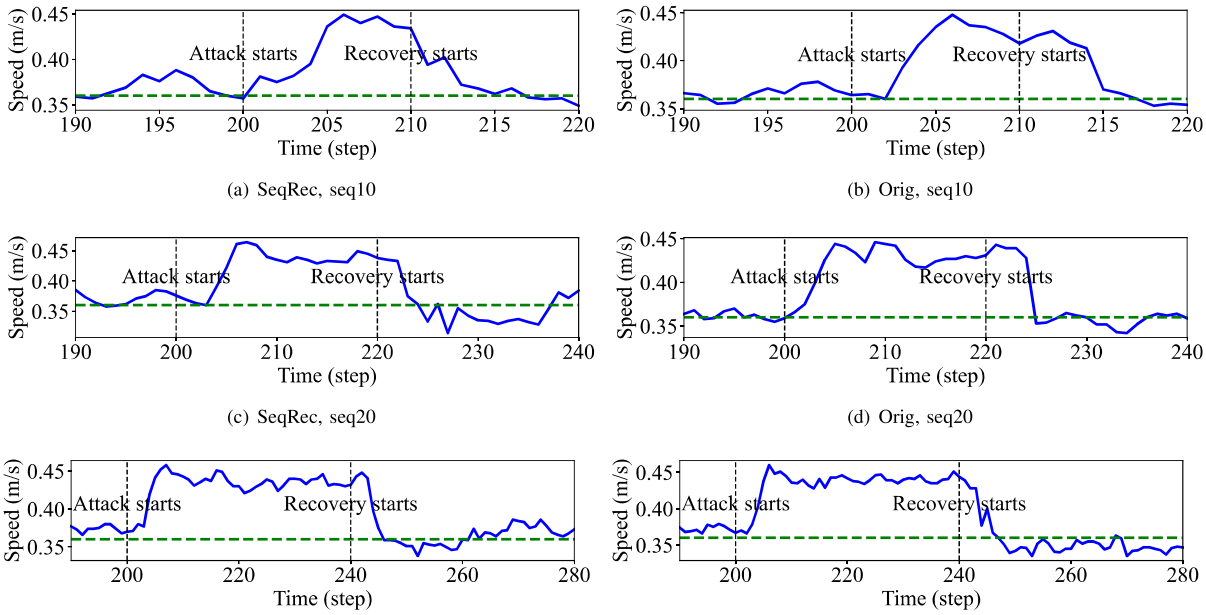
(b) SeqRec

Fig. 9: Recovery Comparison for the 4-wheel testbed turning, using original controller will have a collision with an obstacle due to slow recovery

Fig. 10: Sampled Results on High-fidelity Simulator, the top is the copter on the map, the bottom are the monitoring terminals.

continuous bias attack is injected to the speed sensor, velocity of the x-axis. At the $200 + i$ where i is the length of the sequence, we assumed the attack was detected and the recovery starts. We only plot till the recovery is done, how to control the system after recovery is out of the scope of this paper. Several main observations are as follows. First, both Orig and SeqRec can recover the system back to the target, and SeqRec results in faster recovery which is similar to the results shown in Fig. 7. Second, there are some minor drifts and the curve is not as smooth as the results on simulators. This is caused by multiple reasons such as environmental noise, sensor noises and some errors caused by the scenarios. We tried to make the 4-wheel testbed run through a straight line marked as black on a track(made from nylon curtains). And sometimes there are small folds on it so the resistance and slope would change which affected the speed of the testbed. Also, the line-keeping algorithm could be a minor reason. Since we deployed a simple vision algorithm to control the servo making sure the testbed is running through the line, if the testbed is a little bit deviated from the center of the line, the algorithm will produce a control command to the servo to adjust it, the frequent servo control adjustment is the main reason of the drifts. The speed of the testbed would be affected a little bit during the adjustment, too.

Fig. 9 shows the recovery comparison between SeqRec and Orig for the second scenario. We record the recovery experiments and stack four frames to one picture to show the



(a) SeqRec, seq10 (b) Orig, seq10

(c) SeqRec, seq20 (d) Orig, seq20

(e) SeqRec, seq40 (f) Orig, seq40

Fig. 11: Recovery Comparison of 4-wheel testbed, sequence length of 10, 20, 40, attack starts at the 200th time step.

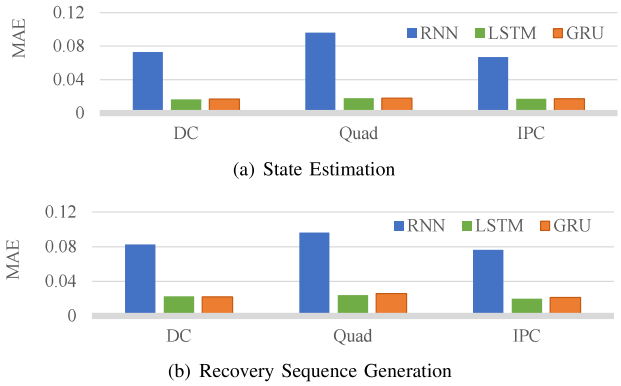
recovery trajectories, the body of the testbed is surrounded by red dashed box in the figure. We set a white box to represent an obstacle. Both SeqRec and Orig have intent to recover the testbed to the center of the road. But the recovery from the original controller is not fast enough which causes a collision as shown in Fig. 9(a). The right front of the testbed hits the obstacle at the end.

B. Accuracy Performance

In this subsection, we discuss the accuracy performance of our model by in quantitative ways in terms of MAE.

Average Error of Our Seq2Seq Models. Table I shows the average state estimation and recovery error (i.e., MAE), which is also calculated from 4000 sequence examples. First, we can see that the average errors are reasonably small, i.e., accurate estimation and recovery sequence generation. This indicates that the learnt Seq2Seq models can accurately capture the system dynamics. Second, the average error becomes larger as the length of the generated sequence increases. This increment is still quite small, which infers that our Seq2Seq models can perform well for a relatively long sequence. Thus, our models are sufficiently good for CPS which usually does not require too long sequences. Third, the average error grows as the system becomes more complex. For example, Quad has a 12 dimension state which is larger than that of DC and IPC, and thus the former's error is higher. However, the increment is not exploding since learning-based methods can well handle high-dimensional data.

Comparison of RNN, LSTM, and GRU. Fig. 12 shows the average state estimation and recovery error for three different



(a) State Estimation

(b) Recovery Sequence Generation

Fig. 12: State estimation and recovery sequence generation errors with a sequence length of 10 using three different recurrent neural networks.

recurrent units over all three simulators. The error is MAE with a sequence length of 10 and averaged over 4000 sequence examples. First, we can see that simple RNN has the worst performance among all for both state estimation and recovery. Note that worse state estimation usually results in worse recovery sequence generation. Simple RNN does not have gate mechanisms and suffers from the problem of vanishing gradients, and thus cannot well capture the information over multiple control steps. Second, LSTM performs slightly better than GRU, and thus, in the following, we show the experimental results mainly on LSTM.

TABLE I: Average State Estimation and Recovery Errors. seq#: the sequence length is #.

	Average Estimation Error			Average Recovery Error		
	seq5	seq10	seq20	seq5	seq10	seq20
DC	0.0112	0.0163	0.0225	0.0183	0.0228	0.0242
Quad	0.0146	0.0193	0.0394	0.0197	0.0243	0.0431
IPC	0.0133	0.0171	0.0271	0.0141	0.0202	0.0253

Single-step LSTM vs Seq2Seq Models. To validate the motivation of using sequence models, we compare single-step LSTM and our Seq2Seq models. Table II shows the average MAE over 4000 examples for state estimation. First, Seq2Seq has lower error than single-step LSTM. The reason is that Seq2Seq models are good at sequence forecasting, while single-step LSTM suffers from the accumulated error due to recursive prediction. Second, as the sequence length increases, the performance gap between Seq2Seq and single-step LSTM becomes larger. The gap increment is not linear since the accumulated error does not follow a linear increasing trend.

TABLE II: Single-step LSTM vs Seq2Seq on MAE for state estimation. S-LSTM: single-step LSTM; Seq len: sequence length.

		Estimates only		Estimates and control demands	
	Seq len	S-LSTM	Seq2Seq	S-LSTM	Seq2Seq
DC	5	0.0269	0.0184	0.0237	0.0141
Quad	5	0.0543	0.0203	0.0433	0.0145
IPC	5	0.0694	0.0202	0.0653	0.0133
DC	20	0.2012	0.0319	0.1731	0.0203
Quad	20	0.2736	0.0495	0.2133	0.0347
IPC	20	0.2334	0.0427	0.1428	0.0245

C. Computational Overhead

To validate that our models can be deployed for on-line use, we measure the inference time of the Seq2Seq models, which are the main source of the computational overhead for SeqRec. Table III shows the inference time for single-step LSTM, our Seq2Seq models, and recovery sequence generation. First, we can see that all inference time is in several milliseconds and is much shorter than the length of a control step which is 20 milliseconds. Second, Seq2Seq has longer inference time than single-step LSTM, but the difference is not large. The reason is that the former has more parameters, and the former produces a sequence while the latter just does a single data point. It is possible to reduced the number of parameters of Seq2Seq models by reducing the number of hidden units in the LSTM cells. However, if we look at the total inference time for producing a sequence, Seq2Seq will be much shorter than single-step LSTM. The reason is that Seq2Seq produces a sequence by one time inference, while the latter needs to predict multiple times (i.e., the number of data points in the sequence) in order to predict a sequence. Third, the inference time grows, but not much, as the sequence length increases. For example, for Quad, the sequence length increases from 5 to 20, but the inference time only increase around 30%.

TABLE III: Inference time of models in milliseconds. seq#: the sequence length is #.

		Estimates only		Estimates and control demands		Recovery
	Seq len	LSTM	Seq2Seq	LSTM	Seq2Seq	Seq2Seq
DC	5	0.7284	1.1291	0.7791	1.1364	1.1921
Quad	5	1.2161	1.5186	1.2203	1.5688	1.7117
IPC	5	0.8936	1.2667	0.9177	1.3091	1.3004
DC	20	0.7284	1.4013	0.7791	1.4021	1.4381
Quad	20	1.2161	1.9974	1.2203	1.9722	2.0373
IPC	20	0.8936	1.5128	0.9177	1.5073	1.4986

TABLE IV: Inference time in milliseconds and overhead in percentage for SeqRec on ArduPilot.

Sequence length	State estimation	Recovery	Total	%
5	2.2977	2.3348	4.6325	46.33%
10	2.3735	2.3911	4.7646	47.65%
40	2.7386	2.7846	5.5232	55.23%
50	2.8341	2.8712	5.7053	57.05%

With or Without Control Demands. We also compare the MAE and inference time of state estimation for the cases using control demands and not in learning models. This comparison is to show the cost and gain of including control commands feature to learning. Table II and Table III show the results. We can see that using control demands results in much better estimation than not, and the two cases have similar inference time. For example, for IPC, using control demands performs 1.75X better in terms of estimation error while there is only $\sim$0.006 millisecond difference.

Table IV shows the overhead of our method by the inference time in milliseconds as well as the percentage on the high-fidelity simulaotr. The percentage is the total inference time of state estimation and recovery sequence generation, over the control step size. We can see that the inference can be done within a control step. Though the inference time increases as the sequence length grows, the percentage is around 50%. Since CPS systems usually do not require a very long sequence, the sequence length of 50 is more than sufficient and its overhead percentage is just 57.05%. Thus, we conclude that SeqRec can be used on-the-fly.

Table V shows the overhead by the inference time in milliseconds as well as the percentage. The percentage is the total inference time over the control step which is 50ms on the testbed. Be noticed the inference time increases with the increment of the recovery sequence length. But the inferences can be finished in a single step and the percentage is around 52.89%. The inference time is highly related to the hardware,

TABLE V: Inference time in milliseconds and overhead in percentage of SeqRec on the 4-wheel testbed.

Sequence length	State estimation	Recovery	Total	%
5	7.6312	7.4428	15.0740	30.15%
10	8.5197	8.6412	17.1609	34.32%
20	10.2163	10.3729	20.5892	40.57%
40	13.0685	13.3753	26.4438	52.89%

especially the configuration of memory, CPU and GPU. In other words, the inference time could be reduced if the models are running on a more powerful machine such as the one we run the simulators. Thus, we conclude that SeqRec can be applied to real CPSs.

VIII. Discussion

In this section, several critical topics of our method will be discussed to specify the scope of application including the security argument, the scalability of our method, applying our method to real-world CPS and some practical considerations.

Attacks on sensor measurements can be recovered using SeqRec in real-time if the attack start time has been diagnosed and the attacked sensors are known. The types of sensor attacks do not affect the recovery scheme since the attacked measurements will not be considered as inputs to SeqRec. For example, SeqRec does not care the sensor attack is bias attack, replay attack or other attacks since the compromised sensor measurements will be discarded.

In this work, we evaluates SeqReq on 3 numerical CPSs, a high fidelity simulator and a real 4-wheel testbed on different settings. We also examined the length of the sequence for recovery to indicate the proposed method is scalable. We have studied the feasibility of SeqRec in terms of inference time under the real-time scenario and different settings which also points out the scalability and applicability of SeqRec. The time overhead mainly has two parts: state estimation and recovery control sequence computation. The first part is needed whether using the original controller or our recovery controller. The second part is only needed by our method. We have tested the method on a 4-wheel testbed to show its effectiveness. However, deploying the method on a testbed is different from applying it to a real-world CPS with more harsh settings and extreme environments. For example, there is a possibility that the control messages sent from the PC are missing. Also, the communication between the devices might be delayed due to the environment such as electromagnetic noise and network congestion. Furthermore, the recovery may fail if the existing detector in the system missed an attack or the diagnosis of the attack is inaccurate. Additionally, our method relies on checkpointing protocols such as [14], [15] to ensure trustworthy historical data.

The recovery performance is highly depends on the accurate state reconstruction and trustworthy historical data. We assume the detector can identify which sensors are under attack and it can diagnose when did the attack starts. If the diagnosis of attack start time is inaccurate, the state reconstruction accuracy will be directly affected since some untrustful data are trusted. If the attacked sensor is mistrusted, the whole state reconstruction will be deviated. Therefore, it is very important to develop detection algorithms for the recovery. Our method does not treat false-positives differently, but triggers recovery as for true-positives and this will cause unnecessary recovery. This may decrease the usability of the system and make it more conservative but safety is not violated. Reducing false-positives is one of the most important focuses of detection. Recovery needs to work with detection and thus is subject to its accuracy. Recently, there are some works focus on the stealthy attack detection and diagnosis [30]. Unfortunately, hidden attacks [6], which cannot be detected by the detectors, violate the assumption and SeqRec can not handle it since no trustworthy historical data are provided.

It is required to have enough computing resources to deploy the proposed method to real-time systems, either on mobile devices or an edge server. Ideally, the computation can be finished onboard and no communication between devices are required. However, if the onboard resources are not met the computing needs, it is flexible to deploy SeqReq on an edge server. Currently, we set a lab computer which is responsible for the computing tasks and the control commands are sent to the testbed since the Raspberry Pi is not capable for the online inference due to the limited resource.

Attacks studied in this paper manipulate the sensor measurements by adding constant values to the sensor measurements. If there is no recovery scheme deployed on the system, the system might be driven to unsafe states after a period of time. In this work, we only evaluate SeqRec when one of the sensors is under attack. If more sensors are attacked, the state reconstruction and attack detection will be more challenging since there are less trustworthy historical data that can be used.

We address two timing issues in this work. The first is to reduce recovery time, i.e. to pursue fast recovery. The longer a system runs under attack, the more risk it will have. Reducing recovery time will reduce the risk and result in successful recovery. The second is to lower computational overhead, i.e. timely computed results for online use. Comparing the proposed method with the recovery using original controller, the proposed method can recover the system in a shorter time in general. A shorter recovery is desired since the system may become unsafe before the recovery is finished. On the other hand, there is no free cake, SeqRec requires more computing resources to maintain in real-time since SeqRec needs to run two Seq2Seq models and the virtual sensor techinque only needs the state estimation network only.

IX. Conclusion and future work

In this paper, we propose a novel sequence-predictive recovery framework as response to sensor attacks. Our method employs Seq2Seq learning models and an MPC-like control algorithm to recover the system to the target state. We evaluate our method using several non-linear simulators and also implement it in a high-fidelity simulator and a 4-wheel testbed. The experimental results demonstrate that our method can rapidly recover CPS to target states effectively and efficiently. In the future, we plan to make the models light-weight and hold the computing on the mobile devices with limited resources such as Raspberry Pi and Nvidia Jetson Nano. It is also interesting to deploy SeqRec on more complex non-linear real-world systems such as drones, boats and underwater vehicles. Another interesting research direction is adapting SeqRec to recover CPSs from actuator attacks or other more complex attack scenarios in practical real-world problems.

X. Acknowledgement

This work was supported in part by NSF CNS- 2333980. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the National Science Foundation (NSF).

References

[1] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Design Automation Conference (DAC)*. IEEE, 2010, pp. 731–736.

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

[3] A. A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *The 28th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2008, pp. 495–500.

[4] M. Wolf and D. Serpanos, "Safety and security in cyber-physical systems and internet-of-things systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 9–20, 2017.

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

[6] M. Liu, L. Zhang, P. Lu, K. Sridhar, F. Kong, O. Sokolsky, and I. Lee, "Fail-safe: Securing cyber-physical systems against hidden sensor attacks," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 240–252.

[7] L. Zhang, Z. Wang, and F. Kong, "Work-in-progress: Optimal checkpointing strategy for real-time systems with both logical and timing correctness," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 515–518.

[8] F. Kong, O. Sokolsky, J. Weimer, and I. Lee, "State consistencies for cyber-physical system recovery," in *Workshop on Cyber-Physical Systems Security and Resilience (CPS-SR)*, 2019.

[9] R. Quinonez, J. Giraldo, L. Salazar, E. Bauman, A. Cardenas, and Z. Lin, "SAVIOR: Securing autonomous vehicles with robust physical invariants," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[10] L. Zhang, X. Chen, F. Kong, and A. A. Cardenas, "Real-time recovery for cyber-physical systems using linear approximations," in *41st IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020.

[11] Y. Zhang and K. Rasmussen, "Detection of electromagnetic interference attacks on sensor systems," in *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[12] Y. Chen, T. Zhang, F. Kong, L. Zhang, and Q. Deng, "Attack-resilient fusion of sensor data with uncertain delays," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 4, pp. 1–25, 2022.

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

[14] F. Kong, M. Xu, J. Weimer, O. Sokolsky, and I. Lee, "Cyber-physical system checkpointing and recovery," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 22–31.

[15] L. Zhang, P. Lu, F. Kong, X. Chen, O. Sokolsky, and I. Lee, "Real-time attack-recovery for cyber-physical systems using linear-quadratic regulator," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[16] A. Watson, J. Park, S. Pugh, O. Sokolsky, J. Weimer, and I. Lee, "Medical cyber-physical systems: Iomt applications and challenges," in *2022 56th Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2022, pp. 998–1004.

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

[18] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, no. 4, pp. 617–636, 2014.

[19] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful gps spoofing attacks," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 75–86.

[20] J. Noh, Y. Kwon, Y. Son, H. Shin, D. Kim, J. Choi, and Y. Kim, "Tractor beam: Safe-hijacking of consumer drones with adaptive gps spoofing," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1–26, 2019.

[21] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. 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)*. IEEE, 2017, pp. 3–18.

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

[23] H. Choi, W.-C. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Deng, "Detecting attacks against robotic vehicles: A control invariant approach," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 801–816.

[24] K. Vatanparvar and M. A. Al Faruque, "Self-secured control with anomaly detection and recovery in automotive cyber-physical systems," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 788–793.

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

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

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

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

[29] L. Zhang, Z. Wang, M. Liu, and F. Kong, "Adaptive window-based sensor attack detection for cyber-physical systems," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 919–924.

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

[31] A. A. Cardenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks against process control systems: risk assessment, detection, and response," in *Proceedings of the 6th ACM symposium on information, computer and communications security*, 2011, pp. 355–366.

[32] R. Mitchell and I.-R. Chen, "A survey of intrusion detection techniques for cyber-physical systems," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 1–29, 2014.

[33] J. Giraldo, E. Sarkar, A. A. Cardenas, M. Maniatakos, and M. Kantarcioglu, "Security and privacy in cyber-physical systems: A survey of surveys," *IEEE Design & Test*, vol. 34, no. 4, pp. 7–17, 2017.

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

[35] H. Choi, S. Kate, Y. Aafer, X. Zhang, and D. Xu, "Software-based realtime recovery from sensor attacks on robotic vehicles," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020, pp. 349–364.

[36] R. Ma, S. Basumallik, S. Eftekharnejad, and F. Kong, "A data-driven model predictive control for alleviating thermal overloads in the presence of possible false data," *IEEE Transactions on Industry Applications*, vol. 57, no. 2, pp. 1872–1881, 2021.

[37] F. Akowuah, R. Prasad, C. O. Espinoza, and F. Kong, "Recovery-by-learning: Restoring autonomous cyber-physical systems from sensor attacks," in *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2021, pp. 61–66.

[38] K. Zhou and J. C. Doyle, *Essentials of robust control*. Prentice hall Upper Saddle River, NJ, 1998, vol. 104.
[39] M. Green and D. J. Limebeer, *Linear robust control*. Courier Corporation, 2012.
[40] F. Akowuah and F. Kong, "Real-time adaptive sensor attack detection in autonomous cyber-physical systems," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 237–250.
[41] ArduPilot, "https://ardupilot.org/," Online; accessed May-2022.
[42] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.
[43] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *arXiv preprint arXiv:2106.11342*, 2021.
[44] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. Kumar, "The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures," in *28th IEEE International Real-Time Systems Symposium (RTSS)*. IEEE, 2007, pp. 400–412.
[45] X. Wang, N. Hovakimyan, and L. Sha, "L1simplex: fault-tolerant control of cyber-physical systems," in *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2013, pp. 41–50.
[46] S. Mohan, S. Bak, E. Betti, H. Yun, L. Sha, and M. Caccamo, "S3a: Secure system simplex architecture for enhanced security and robustness of cyber-physical systems," in *Proceedings of the 2nd ACM international conference on High confidence networked systems*, 2013, pp. 65–74.
[47] P. Filonov, A. Lavrentyev, and A. Vorontsov, "Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model," *arXiv preprint arXiv:1612.06676*, 2016.
[48] P. Filonov, F. Kitashov, and A. Lavrentyev, "Rnn-based early cyber-attack detection for the tennessee eastman process," *arXiv preprint arXiv:1709.02232*, 2017.
[49] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.
[50] N. Muralidhar, S. Muthiah, K. Nakayama, R. Sharma, and N. Ramakrishnan, "Multivariate long-term state forecasting in cyber-physical systems: A sequence to sequence approach," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 543–552.
[51] N. Muralidhar, S. Muthiah, and N. Ramakrishnan, "Dyat nets: Dynamic attention networks for state forecasting in cyber-physical systems." in *IJCAI*, 2019, pp. 3180–3186.
[52] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.
[53] S. L. Brunton and J. N. Kutz, *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.
[54] F. Sabatino, "Quadrotor control: modeling, nonlinear control design, and simulation," Master's thesis, KTH Royal Institute of Technology, 2015.
[55] K. Tan and Y. Li, "Performance-based control system design automation via evolutionary computing," *Engineering Applications of Artificial Intelligence*, vol. 14, no. 4, pp. 473–486, 2001.
[56] Keras, "https://keras.io/," 2022.