

Resilience via Blackbox Self-Piloting Plants

Michel Barbeau¹, Joaquin Garcia-Alfaro^{2,5,1}, Christian Lübben^{3,6},
Marc-Oliver Pahl^{4,5,6,1} and Lars Wüstrich^{3,6}

¹School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada K1S 5B6

²Télécom SudParis, SAMOVAR, Institut Polytechnique de Paris, 91120, Palaiseau, France

³Institut für Informatik der Technischen Universität München Lehrstuhl I8 Boltzmannstr. 3 85748 Garching bei München - Germany

⁴IMT Atlantique, Rennes, France IRISA, UMR IRISA CNRS 6074

⁵IMT Industrial chair Cybersecurity for Critical Networked Infrastructures (CyberCNI.fr)

⁶TUM Smart Space Orchestration team / Chair for Network Architectures and Services (s2o.net.in.tum.de)

Abstract

Distributed control is a reality of today's industrial automation and systems. Parts of a system are on-site, and other elements are on the edge of the cloud. The overall system-functioning relies on the reliable operation of local *and* remote components. However, all system parts can be attacked. Typically, local entities of a cyber-physical system, such as robot arms or conveyor belts, get affected by cyber attacks. However, attacking the control and monitoring channels between a plant and its *remote* controller is attractive, too. There is a diversity of attacks, such as manipulating a plant's input signals, controller logic, and output signals. To detect and mitigate the impact of such various attacks and to make a plant more resilient, we introduce a self-learning controller proxy in the plant's communication channel to the controller. It acts as a local trust anchor to the commands received from a remote controller. It does black box self-learning of the controller algorithms and audits its operations. Once an attack is detected, the plant pivots into self-piloting mode. We investigate design alternatives for the controller proxy. We evaluate how complex the control algorithms can be to enable self-piloting resilience.

Keywords

Cyber-Physical System, Networked-Control Systems, security, incident response, resilience.

1. Introduction

A Cyber-Physical System (CPS), such as a critical infrastructure, consists of distributed physical elements, including local and off-site control algorithms that work together to accomplish a task. All together, the physical elements form the plant. The algorithms make up the controller. Controller and plant communicate through a network, constituting a Networked-Control System (NCS). It is a flexible environment but vulnerable to attacks: attacks on one component can affect the entire system. Hence, the importance of securing a NCS.

C&ESAR'22: Computer & Electronics Security Application Rendezvous, Nov. 15-16, 2022, Rennes, France

EMAIL: barbeau@scs.carleton.ca (M. Barbeau); jgalfaro@ieee.org (J. Garcia-Alfaro); christian.luebben@tum.de

(C. Lübben); marc-oliver.pahl@imt-atlantique.fr (M. Pahl); lars.wuestrich@tum.de (L. Wüstrich)

URL: <https://carleton.ca/scs/people/michel-barbeau/> (M. Barbeau); <http://j.mp/jgalfaro> (J. Garcia-Alfaro);


<https://www.net.in.tum.de/members/luebben/> (C. Lübben); <https://cyberCNI.fr/> (M. Pahl);

<https://www.net.in.tum.de/members/wuestrich/> (L. Wüstrich)

ORCID: 0000-0003-3531-4926 (M. Barbeau); 0000-0002-7453-4393 (J. Garcia-Alfaro); 0000-0001-5241-3809 (M. Pahl)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

The security of a NCS encompasses several aspects, including protection against attacks, recognition of attacks and incident response. Protection is realized by leveraging cryptography techniques and security protocols. Recognition of attacks leverages tools such as intrusion detection techniques. In this paper, we explore an idea for preparing responses to attacks. In particular, when the severity of attacks is such, it becomes safer for a plant component to disconnect from the network and operate autonomously, at least for a short while.

We propose a method to automatically configure a local model of a remote controller by observing its actions. In case a network disconnection is required, the resulting local controller proxy can take over and assure the operation for some time. It makes the local infrastructure more resilient to cyber-attacks. Starting from a mathematical model, the paper develops an approach to learning and consequently predicts a remote controller's behavior. The resulting algorithm is evaluated with an emulated industrial control process.

The related work is reviewed in Section 2. Our system model is described in Section 3. In Section 4, we discuss learning by imitating. Our approach is evaluated in Section 5. We conclude with Section 6.

2. Related Work

A CPS is a combination of hardware and software resources, collectively called a plant or system. The term NCS refers to a CPS monitored and controlled remotely through a communication network. Autonomous transport systems and energy distribution systems are NCS examples. A CPS is vulnerable to several availability, covert [1, 2] and integrity attacks [3, 4, 5]. Adversaries perpetrate attacks by manipulating signals to actuators and from sensors. Perpetrated attacks can lead to disastrous consequences. Hence, a CPS needs to be protected.

Protection involves using cryptographic methods such as digital signatures, encryption, and key establishment. Despite protection, impactful attacks will likely be perpetrated. Hence, detection methods and response plans are required. Several methods have been proposed to detect attacks on a CPS, such as challenge-response authentication [6, 7, 8] and auxiliary states [9]. Adequate incident response mitigates the impacts of attacks and achieves resilience. Resilience to attacks can be obtained by redundancy, diversity [10] and response planning, which means that when attacks are detected, triggering behavior that mitigates impact. Incident response is the aspect that we develop further in this paper.

Our work is related to model predictive control and learning. In predictive control, at time t , the controller sends an input row vector \vec{u} of length $k+1$, the time horizon. The first element of \vec{u} is the input to apply at time t . Predicting a response from the plant (using a model of it), the vector \vec{u} also contains the expected inputs for times $t+1$ to $t+k$. When the plant receives an input vector \vec{u} , it applies the first element in normal mode. The remaining k elements are stored. When a situation occurs, the plant stops accepting new inputs. It keeps running, applying the inputs predicted by the controller. The approach has been used by Quevedo et al. and Franzè et al. to make a system resilient to packet loss [11, 12] or packet replay [13]. Leveraging machine learning, we take this idea to another level.

3. Networked-Control System Model

A CPS is modeled by the following two discrete time equations:

$$x_{i+1} = f(x_i, u_i) \quad (1)$$

$$y_i = g(x_i) \quad (2)$$

x_i and u_i are the current state and actuator inputs, at time i , with $x_i \in \mathbb{R}^m$ and $u_i \in \mathbb{R}^n$. x_{i+1} is the successor state determined by the evaluation of the state-input function $f(x_i, u_i)$, at time $i + 1$, with $x_{i+1} \in \mathbb{R}^m$. y_i is the sensor outputs in the current state defined by the evaluation of output function $g(x_i)$, with $y_i \in \mathbb{R}^p$. The variables i , m , n and p denote positive integers.

As an example, let us consider a system that consists of a single cylindrical tank. The tank has one inflow and one outflow of liquid. The dynamics of liquid level in the tank are captured by the following differential equation [14]:

$$\frac{dh(t)}{dt} = \frac{F(t) - a\sqrt{h(t)}}{\alpha} \quad (3)$$

Eq. (3) models the relationship between instantaneous changes in liquid level and the difference between the inlet flow rate and outlet flow rate. As a function of time t (second), the level of the liquid in the tank is $h(t)$ (cm). Variable α represents a cross-sectional area of the tank (cm^2). The term $F(t)$ represents the inlet flow rate ($\text{cm}^3/\text{second}$). The parameter a denotes the outlet valve coefficient. The outlet flow rate ($\text{cm}^3/\text{second}$) is proportional to the product a times the square root of the pressure represented by the term $h(t)$. Note that because of the square root term, the system is nonlinear.

Mapping Eq. (3) into the model of Eq. (1), we get:

$$x_{i+1} = f(x_i, u_i) = x_i + \frac{u_i - a\sqrt{x_i}}{\alpha} \quad (4)$$

$$y_i = g(x_i) = x_i \quad (5)$$

where u_i represents the input flow rate at time i . In the sequel, this dynamics is used to build a NCS case study.

3.1. Architecture

We assume that remote monitoring control of the CPS is required. Hence, it is a NCS. The architecture of a NCS is pictured in Figure 1 (a). There is a System consisting of resources that can be controlled and monitored. There is a Controller that posts inputs and gets outputs from the System. The inputs are commands to actuators of the System. The outputs are readings from sensors attached to the System. Inputs and outputs travel over a network. This architecture is relevant to situations requiring remote control and monitoring. The attack model is pictured in Figure 1 (b). Somewhere in the network, an adversary stands between the Controller and System. The adversary can modify inputs and outputs. In the sequel, we assume that data modification attacks can be detected by both the Controller and System, using, for instance, a digital signature mechanism.

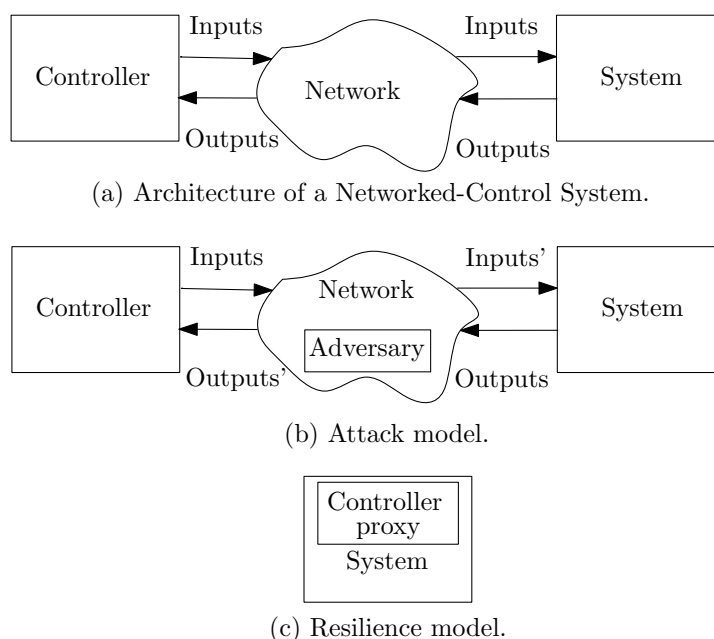


Figure 1: Networked-Control System: architecture, attack model and resilience model.

3.2. Resilience Model

We define resilience as a system's ability to maintain operation while an attack is being perpetrated. In Figure 1 (c), we propose a system architecture with resilience by design. The remote controller controls and monitors the system during regular operation. The system can detect attacks from cyberspace and disconnect itself from the network when attacks occur. When the perpetration of an attack is detected, the System disconnects itself from the network. The System embeds a controller proxy. While disconnected from the network, control of the CPS is handed over to the controller proxy. The controller proxy may act on the physical system with its actuators and observe it with its sensors. Hence, the controller proxy may have its own state-input function $f_L(x_i, u_i)$ and output function $g_L(x_i)$.

4. Learning to Control by Imitating

We develop a controller proxy learning approach for the architecture of Figure 1 (c). Let us first consider radio-controlled model airplanes. They have a pre-programmed solution. They memorize their launching location. When a model airplane goes out of the wireless range of its controller, the aircraft detects the signal loss. An auto return to launch site function kicks in. The plane returns over the takeoff point, circles around it, and lands safely. The return to launch site solution has also been adopted by modern quadcopters. This solution is interesting and specific to a context. It can hardly be generalized to other types of situations. We aim at general solutions.

We follow the more generic idea of imitation learning [15, 16, 17]. The learning entity is called an agent. In imitation learning, the agent observes a demonstration of a task and tries to mimic the behavior. There are two different categories of imitation learning. The first one is behavioral cloning which uses supervised learning to infer the relationship from observations to actions [17, 18]. In contrast, with inverse reinforcement learning, a reward function is estimated to explain the teacher’s behavior [19, 20]. In our case, we are interested in the former, i. e., finding an appropriate mapping between the observations and actions. Behavioral cloning and DAGGER are examples of imitation supervised learning [18, 19, 21]. They do not require a specification of the reward function.

We propose a flexible solution where a controller proxy observes, imitates, and acquires the behavior of a remote controller, i. e., the actions it performs in each state. In this context, imitation learning means that the controller proxy is an agent that infers the controller’s behavior. The remote controller is a teacher. The controller proxy is a learner. The teacher does not need to be aware of the learner. It performs its task normally. The learner is an observer. The teacher is part of its environment. When an attack is detected, the controller proxy agent acts on the system instead of the remote controller. For detection methods, see Section 2. In the sequel, we focus on the controller proxy agent training.

The agent can only observe the communications between the controller and the system. There are two types of communication. The first type of communication is from the controller to the system. This communication contains commands to the system. The system executes the commands and responds with the resulting state. As a function of the new state, the controller sends a new set of commands in the second communication type. To faithfully imitate the controller, the controller proxy needs to reliably predict the new set of commands based on the reported system state. Therefore, it must learn a policy that reflects the controller’s decision-making.

Our reinforcement learning architecture reflects this setting. There are two types of scenarios. The first type of scenario is stateless. The controller consistently makes the same decision given the same input. The second type of scenario is stateful. In this case, in addition to the system’s current situation, the state also needs to capture the trends of the ongoing activity. For example, a water tank may have a changing target level. A controller’s decisions for a water tank collecting water differ from the commands for a tank leaking water. Because they are more general, we focus on stateful physical systems.

The input to the agent is a series of observations of the communication flow on the channel between the controller and the system. Let a command from the controller to the system have n parameters. Moreover, let a system state be a vector of m parameters. With i denoting the number of interactions between the controller and system, i. e., one command and a corresponding state vector, the input to the agent has length $i \cdot (m + n)$ parameters. The agent is limited to the actions that the controller takes. Since the agent should imitate the controller, the commands it receives have the same dimension as the commands produced by the controller. Furthermore, the agent’s output is also command vectors of n parameters, i. e., its choice of actions according to the policy acquired with reinforcement learning. The size of the reinforcement learning action space grows exponentially with the number of commands available to the controller. The growth depends on two factors. The first is the number of possibilities for each action. The second factor is the number of actions included in a command vector. The combination of

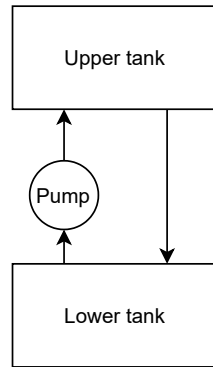


Figure 2: Two-tank evaluation scenario.

possible commands, i. e., actions, grows with the possible combinations that the controller can send. When receiving an input, the agent predicts the chosen action of the controller. When it correctly predicts the new parameters, it receives a positive reward. Otherwise, it receives a negative reward, i. e., a penalty.

We use a neural network for the core of the agent. Within these parameters, the neural network has $i \cdot (m + n)$ input neurons and n output neurons. In addition, the neural network can have one or more hidden layers of varying sizes. Their number depends on the scenario. The agent should accurately imitate the controller’s logic. This implies that the agent should not explore new action paths to optimize its policy. The agent concentrates on exploitation.

The training of the agent results in a policy that reflects the controller’s logic. If needed, it makes it capable of replacing the controller in case of an attack and managing the system according to the observed behavior. In contrast to predictive control with a finite time horizon, the controller proxy can operate while the system is disconnected for an arbitrarily long period.

5. Evaluation

In this section, we demonstrate the controller proxy concept. We assess the feasibility of our method and its performance versus the number of observations made by the controller proxy. To do so, we choose a concise yet realistic setting: a two-water tank control problem. The plant comprises two tanks while the controller maintains target water levels. The controller is remote to the plant in the cloud. Collocated with the plant, we place a self-learning controller proxy. The quality of service is evaluated with variable conditions.

The process works as follows. The controller proxy learns by passively observing the communication between entities inside the plant and one or multiple external controllers. After observing the state of the monitored plant, the internal controller decides and compares its decision to the command received from the external controller in the next step. When the external controller gets disconnected, the internal controller proxy takes over and mimics its behavior to keep the plant in an optimal state.

For our evaluation, we consider the following scenario. The plant consists of two water tanks, one being on top of the other. Two pipes connect the water tanks. One pipe has an in-line water

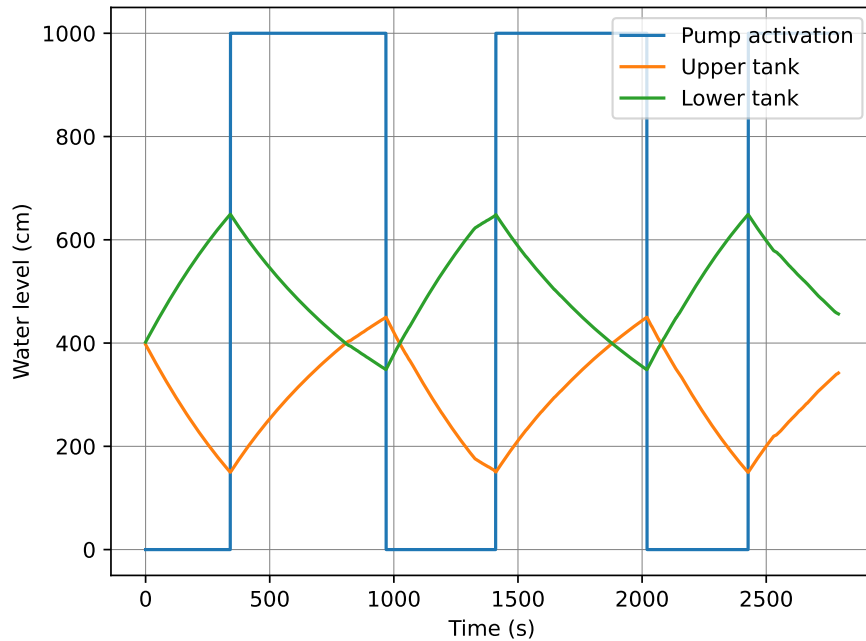


Figure 3: Cyclic behavior of the plant over time. The horizontal axis represents time (seconds). The vertical axis corresponds to the pump state (on or off) or tank water level (cm). The blue line tracks the pump's on/off (low/high) state. The green (orange) line plots the water level in the lower (upper) tank.

pump that can be turned on or off. When turned on, the pump moves water from the lower tank to the upper one. While the pump moves the water to the upper tank, the water level in the lower tank decreases. The water from the upper tank flows through the second pipe from the upper tank to the lower tank. When the pump is off, the water level in the upper tank decreases and the level in the lower tank increases. The pump turns on when the upper tank is emptied to a certain threshold. It remains on until the upper tank reaches the maximum threshold and is turned off. The pump is reactivated when enough water from the upper tank has flown to the lower tank. Figure 2 shows the setup.

Figure 3 shows this deterministic cyclic behavior to be learned by observation and imitation. The horizontal axis corresponds to the time in seconds. The vertical axis represents the water level in tanks in cm. The green curve denotes the water level of the lower tank. The orange curve corresponds to the level of the upper tank. The blue line represents the state of the pump; the low level is off, high status is on. For this scenario, the sampling period is five seconds.

This scenario is implemented and simulated using the Virtual State Layer (VSL) middleware [22]. VSL enables rapid prototyping and realistic evaluation of the scenario. Building on the VSL micro-service architecture concept, one service is created for each entity in the scenario. The resulting system consists of six services. One service for each tank, the valve, pump, and the external controller. In addition, there is a plotter service recording measurements. The pump is modeled to fill the upper tank with a fixed flow of 500 cm^3 per second. The outflow of the upper tank depends on the water level and is simulated using the differential equations introduced in Section 3. The initial water levels of both tanks are 400 cm. The upper tank has

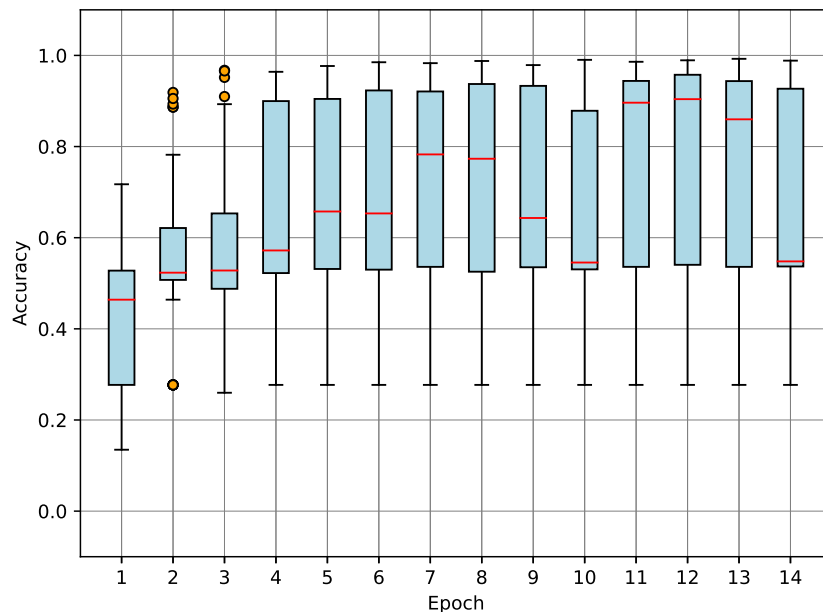


Figure 4: Accuracy vs. number of epochs. The horizontal axis corresponds to the number of epochs used for training, one to 14. The vertical axis represents the accuracy (on a scale of zero to one). We use statistical boxplots. For every box, the central (red) mark indicates the median. The bottom and top edges of the box indicate the 25th and 75th percentiles. The whiskers extend to the most extreme data points not considering outliers. Outliers are represented by the orange circles.

a maximum water level of 500 cm. When reaching 90% of the maximum level, the external controller turns off the pump. It turns the pump on again when the level of the upper tank drops below 30% of the maximum level.

We train a Deep Q-learning (DQN) agent for this task on the observing controller proxy. The input to the DQN is the current level of each water tank and the status of the pump. The action space switching the pump on or off in the next step. The DQN outputs a prediction for the activation of the pump in the next step. It also observes the command of the external controller that it predicted and uses it as label to evaluate the prediction. We implement the DQN in a separate Python program using PyTorch. The agent is connected via interfaces to the VSL middleware such that it can observe the activities in the simulation.

There are various parameters that can affect the result of the training. For our evaluation, we focus on the number of epochs for training, and the number of observations at the proxy controller. Both affect the amount of data that the proxy controller has for learning.

In a first experiment, we evaluate the quality of proposed actions by the local controller proxy depending on the number of trained epochs and observations. We assess the performance by comparing DQN predictions against the actual commands issued by the external controller.

We train multiple DQNs while varying amounts of data and epochs. Each DQN has random weights at initialization. Then, we train each DQN for 1, 2, ..., 14 epochs. We repeat this experiment with different amounts of training data. We use sequential data starting from the beginning of the simulation. Depending on the amount of provided data, a DQN is trained with

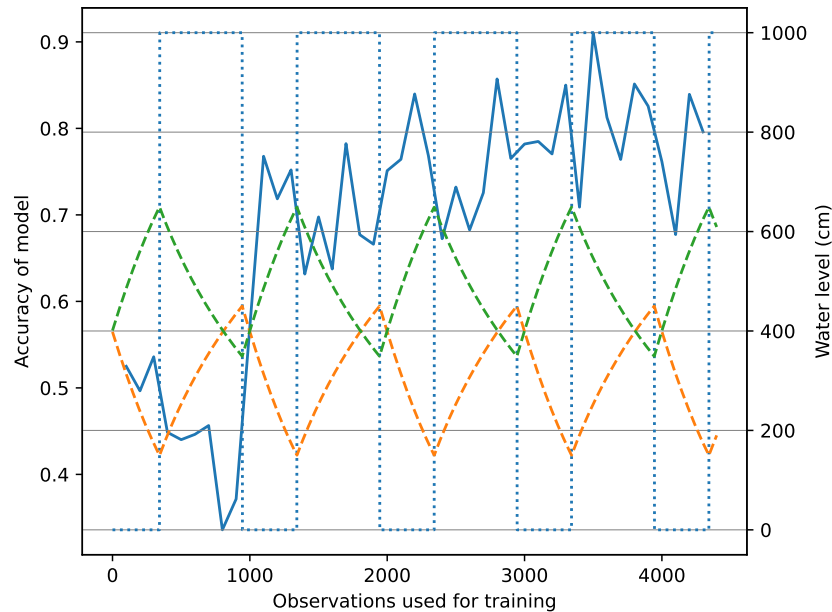


Figure 5: Performance vs. training instances. The horizontal axis corresponds to the number of observations used for training. It ranges from 100 to 4300 observations. The left vertical axis indicates the accuracy of the model together with the blue solid line. The blue dotted line tracks the pump's on/off (low/high) state. The right vertical axis corresponds to the tank water level (cm). The green (orange) line plots the water level in the lower (upper) tank.

a partial observation of a cycle, a complete, or multiple cycles. The smallest amount of data used for training is the first 100 measurements. For the number of epochs, the DQNs are trained with 100, 200, 300, . . . , 4300 observations.

Figure 4 shows the accuracy of DQNs as a function of the number of trained epochs averaged over the different amounts of training. The accuracy is defined as correctly predicted labels over all samples. The accuracy rises until epoch seven. DQNs trained for eight to 10 epochs have decreasing accuracy. DQNs trained for 12 epochs have the highest accuracy. Considering the trade-off between training time and accuracy, we use seven epochs for further evaluation. The accuracy gain at 12 epochs is only around 5%, while the training time rises by about 170%.

Using the fixed number of seven epochs, the following experiment evaluates the impact of the number of observations on a model's accuracy. We, therefore, trained multiple DQNs while varying the amount of sequential training data. Figure 5 shows that the performance of trained models correlates with the quantity of training data. The dashed lines show the observations used to train the model. Models that are only trained on observations where the pump is turned off perform poorly. The performance increases significantly after observing one cycle in which the pump was turned off and turned on. The maximum accuracy is achieved at 3500 observations. The results are averaged over ten iterations of the experiment.

We implemented in VSL the internal controller proxy to verify the training accuracy. We used a learned model on live data. Based on the results of the former evaluation, the controller proxy implements a model trained for up to seven epochs, using 3500 observations. The results

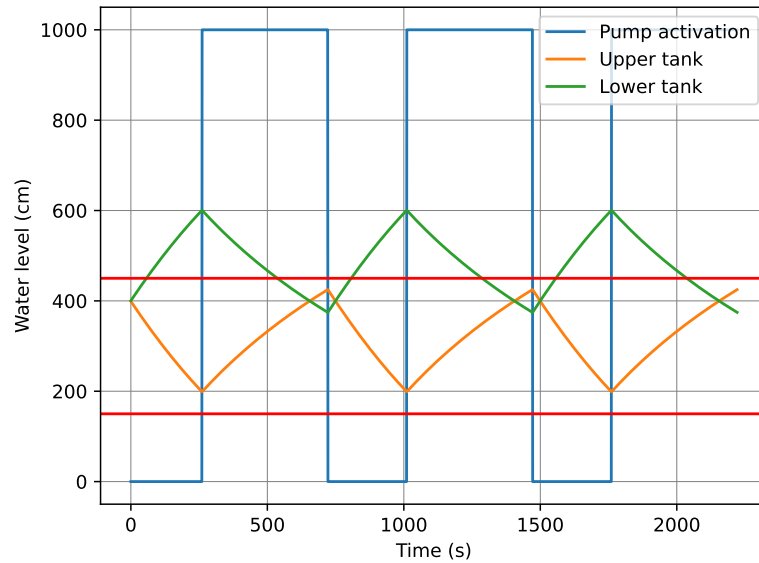


Figure 6: Proxy controller applied to live data. The red lines represent the limits of the upper tank.

are shown in Figure 6. The red lines mark the upper and lower limits implemented in the original controller. Compared to Figure 3, it shows that the internal controller proxy learned the behavior of the external controller. The internal controller keeps the upper tank level within the limits of the external controller.

The previous evaluation shows the feasibility of the approach. Our local proxy controller became a self-learned digital twin of the remote controller. It showed that the approach works in our limited scenario. Our evaluation setting fulfills our assumptions of a local CPS that is controlled from the outside. Therefore, It can be expected that the approach is also fitting for bigger settings. We plan to evaluate this in the future.

6. Conclusion

This paper showed how a remote controller could be learned locally, resulting in a digital twin of the controller. It showed how the digital twin acts as a trust anchor, enabling anomaly detection and mitigation from attacks by taking over control in case of an attack.

This work shows the feasibility of the idea by describing the approach and evaluating it at a representative scenario. In the future, we plan to focus on more complex systems to evaluate the performance of the approach.

7. Acknowledgments

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC). This research took part in the context of the industrial chair Cybersecurity for Critical Networked Infrastructures (CyberCNI.fr) with the support of the FEDER development

fund of the Brittany region. This work was also supported by the German Federal Ministry of Education and Research under funding number 16KIS1221 (SKINET),

References

- [1] R. Smith, A decoupled feedback structure for covertly appropriating networked control systems, *IFAC Proceedings Volumes 44* (2011) 90–95. 18th IFAC World Congress.
- [2] R. Smith, Covert Misappropriation of Networked Control Systems: Presenting a Feedback Structure, *IEEE Control Systems* 35 (2015) 82–92. doi:10.1109/MCS.2014.2364723.
- [3] B. Ramasubramanian, M. Rajan, M. G. Chandra, Structural resilience of cyberphysical systems under attack, in: 2016 American Control Conference (ACC), IEEE, 2016, pp. 283–289.
- [4] A. Chapman, M. Mesbahi, Security and infiltration of networks: A structural controllability and observability perspective, in: *Control of Cyber-Physical Systems*, Springer, 2013, pp. 143–160.
- [5] C. Barreto, A. A. Cárdenas, N. Quijano, Controllability of dynamical systems: Threat models and reactive security, in: *International Conference on Decision and Game Theory for Security*, Springer, 2013, pp. 45–64.
- [6] J. Rubio-Hernan, L. De Cicco, J. Garcia-Alfaro, Revisiting a watermark-based detection scheme to handle cyber-physical attacks, in: *Availability, Reliability and Security (ARES)*, 2016 11th International Conference on, (Best Paper Award), IEEE, 2016, pp. 21–28. URL: <http://dx.doi.org/10.1109/ARES.2016.2>. doi:10.1109/ARES.2016.2.
- [7] J. Rubio-Hernan, L. De Cicco, J. Garcia-Alfaro, Adaptive control-theoretic detection of integrity attacks against cyber-physical industrial systems, *Trans. Emerging Telecommunications Technologies* 32(09) (2017). URL: <http://dx.doi.org/10.1002/ett.3209>. doi:10.1002/ett.3209.
- [8] J. Rubio-Hernan, L. De Cicco, J. Garcia-Alfaro, On the use of watermark-based schemes to detect cyber-physical attacks, *EURASIP Journal on Information Security* 2017 (2017) 8. URL: <http://dx.doi.org/10.1186/s13635-017-0060-9>. doi:10.1186/s13635-017-0060-9.
- [9] C. Schellenberger, P. Zhang, Detection of covert attacks on cyber-physical systems by extending the system dynamics with an auxiliary system, in: 2017 IEEE 56th Annual Conference on Decision and Control (CDC), 2017, pp. 1374–1379. doi:10.1109/CDC.2017.8263846.
- [10] M. Barbeau, F. Cuppens, N. Cuppens, R. Dagnas, J. Garcia-Alfaro, Resilience estimation of cyber-physical systems via quantitative metrics, *IEEE Access* 9 (2021) 46462–46475.
- [11] D. E. Quevedo, E. I. Silva, G. C. Goodwin, Packetized predictive control over erasure channels, in: 2007 American Control Conference, 2007, pp. 1003–1008.
- [12] G. Franzè, F. Tedesco, D. Famularo, Model predictive control for constrained networked systems subject to data losses, *Automatica* 54 (2015) 272 – 278. URL: <http://www.sciencedirect.com/science/article/pii/S0005109815000710>. doi:https://doi.org/10.1016/j.automatica.2015.02.018.
- [13] G. Franzè, F. Tedesco, W. Lucia, Resilient control for cyber-physical systems subject to replay attacks, *IEEE Control Systems Letters* 3 (2019) 984–989.
- [14] A. K. Tangirala, *Principles of System Identification: Theory and Practice*, CRC Press, 2014.

- [15] B. D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, *Robotics and autonomous systems* 57 (2009) 469–483.
- [16] A. Billard, S. Calinon, R. Dillmann, S. Schaal, Robot programming by demonstration, in: *Springer handbook of robotics*, Springer, 2008, pp. 1371–1394.
- [17] S. Schaal, Is imitation learning the route to humanoid robots?, *Trends in cognitive sciences* 3 (1999) 233–242.
- [18] S. Ross, G. Gordon, D. Bagnell, A reduction of imitation learning and structured prediction to no-regret online learning, in: *Proceedings of the fourteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings*, 2011, pp. 627–635.
- [19] Y. Duan, M. Andrychowicz, B. Stadie, O. J. Ho, J. Schneider, I. Sutskever, P. Abbeel, W. Zaremba, One-shot imitation learning, in: *Advances in neural information processing systems*, 2017, pp. 1087–1098.
- [20] A. Y. Ng, S. Russell, et al., Algorithms for inverse reinforcement learning., in: *ICML*, volume 1, 2000, p. 2.
- [21] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, An algorithmic perspective on imitation learning, *Foundations and Trends® in Robotics* 7 (2018) 1–179. URL: <http://dx.doi.org/10.1561/23000000053>. doi:10.1561/23000000053.
- [22] M.-O. Pahl, S. Liebold, Information-centric IoT middleware overlay: VSL, in: *2019 International Conference on Networked Systems (NetSys) (NetSys'19)*, Garching b. München, Germany, 2019, pp. 1–8.