석 사 학 위 논 문
Master's Thesis

강화학습 기반 자가 적응 사이버 물리 시스템 오브 시스템즈의 변칙 고려 적응 기법

Anomaly-Aware Adaptation Approach for Self-Adaptive Cyber-Physical System of Systems Using Reinforcement Learning

2022

조 은 호 (趙 恩 浩 Cho, Eunho)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

석 사 학 위 논 문

강화학습 기반 자가 적응 사이버 물리 시스템 오브 시스템즈의 변칙 고려 적응 기법

2022

조 은 호

한 국 과 학 기 술 원

전산학부

강화학습 기반 자가 적응 사이버 물리 시스템 오브 시스템즈의 변칙 고려 적응 기법

조 은 호

위 논문은 한국과학기술원 석사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2021년 12월 13일

심사위원장 배 두 환 (인)

심 사 위 원 고 인 영 (인)

심 사 위 원 지 은 경 (인)

Anomaly-Aware Adaptation Approach for Self-Adaptive Cyber-Physical System of Systems Using Reinforcement Learning

Eunho Cho

Advisor: Doo-Hwan Bae

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Daejeon, Korea
December 13, 2021

Approved by

Doo-Hwan Bae
Professor of Computer Science

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

MCS 조은호. 강화학습 기반 자가 적응 사이버 물리 시스템 오브 시스템즈의 변칙 고려 적응 기법. 전산학부 . 2022년. 31+iv 쪽. 지도교수: 배두환. (영문 논문)
Eunho Cho. Anomaly-Aware Adaptation Approach for Self-Adaptive Cyber-Physical System of Systems Using Reinforcement Learning. School of Computing . 2022. 31+iv pages. Advisor: Doo-Hwan Bae. (Text in English)

초 록

사이버 물리 시스템 오브 시스템즈 (CPSoS)는 물리적, 사이버 환경에서 소통하는 구성 시스템들로 이루어진 시스템으로 다양한 환경에서 작동하므로, 자가 적응성을 갖추는 것이 매우 중요하다. 하지만, CPSoS의 자가 적응성을 위해서 시간 제약과 변칙이라는 두 가지 과제가 있다. 시스템의 적응은 정해진 시간 제약 내에 이루어져야 하며, 기계적 결함, 사이버 공격, 혹은 창발적 특성 등에서 오는 시스템의 변화에서 생기는 변칙을 고려할 수 있어야 한다. 하지만, 존재하는 기법은 두 측면을 완벽하게 처리하지 못하고 있다. 이 연구에서는 런타임에 큰 시간을 소모하지 않고, 알려진 시스템 변칙을 처리하는 새로운 기법, A^4를 제시한다. 이 기법은 런타임 전에 알려진 시스템 변칙을 학습하고, 해당 변칙이 감지될 때, 그 영향을 완화하는 기법이다. 이 연구에서는 A^4와 다른 기법을 가상, 물리적 CPSoS에서 평가하였으며, A^4가 다른 기법과 비교해 충분히 효율적임을 보여준다.

핵 심 낱 말 사이버 물리 시스템, 시스템 오브 시스템즈, 자가 적응 시스템, 시스템 변칙, 강화 학습, 전이 학습

Abstract

Cyber-Physical System-of-Systems (CPSoS) is a system that is composed of multiple constituent systems that exist and interact with both physical and cyber environments. It is regarded as essential to reach the self-adaptivity to CPSoS because it is working on the uncertainty of various environments on both cyber and physical. Two challenges to achieving the self-adaptive CPSoS are time-constraint and system anomalies. The adaptation should be processed within the time constraint and consider the anomalies caused by the system's change due to mechanical faults, cyber-attacks, or emergent behaviors. However, existing adaptation approaches cannot fully handle both aspects. This research proposes the advanced approach, A^4, for the self-adaptive system that handles the known anomalies without enormous time in runtime. This approach learns the known anomalies before runtime and mitigates their impact when the anomalies are detected. The research evaluates the A^4 approach on the virtual and physical CPSoS and shows that A^4 is efficient enough rather than other approaches.

Keywords Cyber-Physical System, System of Systems, Self-Adaptive system, System anomaly, Reinforcement learning, Transfer learning

Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Research Background

The Cyber-Physical System-of-Systems (CPSoS) comprises multiple constituent systems that exist and interact with both physical and cyber environments. [3] With the improvement of modern software technologies like the Internet of Things or Cyber-Physical Systems, the concept, CPSoS came out to the reality by making connections between the physical and cyber environment in the complex system-of-systems. However, those connections increase the variety and complexity of the environment that CPSoS face. [4] Thus, without the proper strategy to mitigate the various uncertainty of the environment, it is almost impossible to design software that achieves the system goal reliably and safely for every possibility of the environment. [5]

The self-adaptive system is the one approach that enables the re-organization of the behavior and structure of the system to achieve the system's goal at runtime in response to the changing environment. The most adaptation process of the self-adaptive system is called MAPE [6], which continually repeat monitoring the environment, analyzing & planning the adaptation, and executing the adaptation tactic. Therefore, self-adaptation is regarded as essential to CPSoS to reduce the effort and improve the system's reliability. [7]

There are various existing adaptation approaches for the self-adaptive systems. One existing approach uses model checking (MC) techniques to verify whether the adaptation tactics meet the system's goal or not. [8, 9] These approaches compose the formal model like Markov Decision Process and verify the model with properties based on the system's goal. Another adaptation approach based on reinforcement learning (RL) finds the adaptation decision based on the learned knowledge by offline or online learning. [10, 11] With the reward configured by the system's goal, the RL model trains offline and online to understand the system's behavior and environment. Furthermore, based on the learning of the system and environment, the model selects the proper adaptation tactic for the given state of the system and environment.

However, there are two challenges for achieving self-adaptation in the CPSoS, and existing approaches cannot fully handle them. The one thing is the time constraint. The CPS should adapt to react to the environmental change within a certain time. The MC-based approaches require high verification costs like time or memory to verify the whole system. For complex systems like a CPSoS, the whole adaptation process might be failed because of time and memory constraints. Despite statistical model checking costing less than probabilistic model checking, it also needs to spend a long time verifying the tactics. [8]

Another challenge is the anomaly of the CPSoS. The anomaly of the CPS is the abnormal state or data that the system produces. The system's change causes those CPS anomalies due to mechanical failures, cyber-attacks [12], or emergent behavior, which can be discovered in system-of-systems. [13] The RL-based approaches which use offline learning cannot handle this problem. One possible solution is the system's online learning to re-train based on the changed system. However, the online RL model can also lower the efficiency when the anomaly remains during the re-training process, so the model should re-train twice to adapt the system with anomaly and without anomaly.

1.2 Motivation & Goal

The adaptation for CPSoS should be free from time-constraint problems.

The critical issue for applying the self-adaptive system in the real world is the time for each adaptation process. Every system should meet the needs of time constraints. However, the system is complex enough to verify the adaptation tactics hardly.

There are anomalies of a system that affect the adaptation approach's performance.

Existing adaptation approaches, mostly reinforcement learning-based approaches, cannot handle the system anomalies caused by changing the system's behavior due to various factors like mechanical failures or emergent behavior. Therefore, in this research, the adaptation approach will consider anomaly detection and verification of the tactics.

The contributions of this research are as summarized:

1. This research proposes the novel adaptation approach, A^4, for a self-adaptive system that is free from time constraints and considers the unintended behavior of the system.
2. This research applies and evaluates the proposed approach, A^4 to testbed to show the performance of the approach.
    - The evaluation is conducted with two self-adaptive testbeds, one of which is a physical testbed.

1.3 Thesis Outline

The remainder of this paper is organized as follows. Chapter 3 introduces the fundamental concepts of the requirements of understanding of this paper. The related works of self-adaptive systems are described in chapter 2. Chapter 4 shows the detailed approach of A^4, and it evaluates in chapter 5. Furthermore, chapter **??** shows the threats to the validity of this research, and chapter 6 concludes the paper.

Chapter 2. Related Work

There have been various existing approaches for the self-adaptive system. The researches can be divided into three possible solutions, improving and resolving the complexity of the model checking (MC) technique [8, 9, 14], utilizing the reinforcement learning (RL) technique [11, 15, 16], and using both model checking and reinforcement learning techniques to ensure both time constraint, and performance. [10, 17]

C. Stevens et al. (2020) proposed the approach and the tool that reduces the model's size for the MC technique by excluding tactics to concern. The approach defines the two models, structural and behavioral. The tool analyzes both models and extracts the reachable state and possible reward bounds information. The potential metric for each adaptation tactic is calculated based on the analysis. Furthermore, the approach excludes non-Pareto optimal adaptation tactics based on the metric values. As a result, the tool reduces the model size average by 60% and 20% for two self-adaptive system testbeds. [9]

M.A. Nia et al. (2020) also proposed an approach that approximates the verification process. This approach first constructs the formal model of the system and analyzes the strongly connected component that has a direct connection between states among the system states. Moreover, each strongly connected component is approximated independently based on the approximation algorithm. To sum up, although the adaptation performance is less than the non-approximated result, the cost for adaptation decreased an average of 25%. [14]

Y.J. Shin et al. (2021) proposed to use statistical model checking (SMC) instead of probabilistic model checking and a formal model of the system. Instead of using the formal methods, SMC requires the simulation and a sampled future environment. Based on the result of SMC, the system selects the proper adaptation tactic. This research shows that SMC can dramatically advantage over time rather than probabilistic model checking. [8]

The approaches that improve original MC techniques show that they can take advantage of costs. However, the evaluation of the approaches is limited to the simple self-adaptive system with a small number of tactics and no physical components. Y.J. Shin et al. (2021) evaluate the complex system. However, the approach also takes a long time to process the adaptation.

For utilizing reinforcement learning, D. Kim et al. (2009) is the first research utilizing reinforcement learning on the self-adaptive system. They proposed the Q-learning-based self-management system and showed the relationship between the performance and hyperparameters of the RL techniques. It also first proposed the online learning process of the RL model. [15]

A. Palm et al. (2020) organize the online RL approach for the self-adaptive system. In this research, they proposed the policy-gradient-based, online-learning RL approach. Every step, the approach monitors the state and gets its reward. The policy is updated based on the reward, system state, and last step's action. They showed that the approach could adapt to various sets of environments. [11]

Approaches that utilize RL show the potential of applying various machine learning techniques to self-adaptive system problems. RL-based approaches seem very familiar to adaptation approaches because solving a game in RL techniques is very similar to finding an optimal tactic of a self-adaptive system. However, the machine learning models cannot handle any drastic changes, and it is not yet proven to converge to the optimum.

F. Quin et al. (2019) proposed utilizing machine learning techniques to find the relevant adaptation

tactics that are likely to be optimal. Based on the MAPE-K loop, the analyzer turns on the machine learner to determine the set of optimal-like adaptation options. The machine learner returns the result, and the model checker verifies the adaptation options. After the verification ended, the machine learner improved based on the verification result. [17]

J. Cámara et al. (2020) proposed the approach for a self-adaptive IoT system that uses RL to select the tactic. The target system of the research has quality goals that have to satisfy. The pattern selector, which is the Q-Learning RL model, learns the adaptation tactic based on the given system state. The selected tactic is translated into the formal model and verified by the model checker. If the verification result satisfies the QoS goals, the tactic is selected. [10]

Both approaches used the RL model, but the method utilizing the RL model is slightly different. F. Quin et al. (2019) [17] used it for reducing the adaptation space, and it seems like the RL version of the approach of C. Stevens et al. (2020). [9] Therefore, the limitation of the approach, time complexity, might be a problem even it reduces the verification space. The approach of J. Cámara et al. (2020) [10] is more look like the RL-based approaches. However, it can use it only for the systems with QoS goals and the same limitation, anomalies.

Chapter 3. Background

3.1 Self-adaptive System

The term *adaptation* came from biology, which means the process of changing the organisms because of the changing environment. Especially, prefix *self-* in the term *self-adaptation* shows that the adaptation process of the system is executed without any help, support, or modification of human beings. [18] Thus, the self-adaptive system means the system that reconfigures itself autonomously to respond to environmental changes.

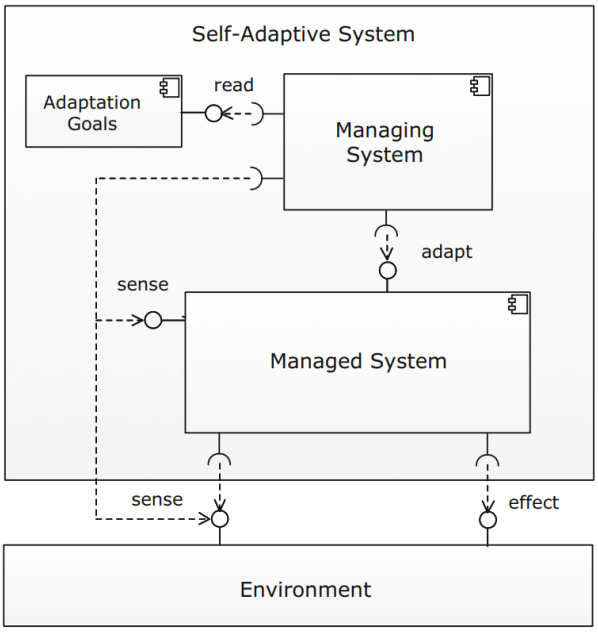

Figure 3.1: Conceptual model of self-adaptive system [1]

Figure 3.1 shows the conceptual model of a self-adaptive system with four fundamental parties: adaptation goals, managing system, managed system, and environment.

The environment is the external world of the self-adaptive system. A self-adaptive system tries to adapt to this environment by sensing the environment via sensors and affecting the environment via actuators. For example, road traffic should be the environment in the self-adaptive traffic light system. The traffic light system senses the traffic and actuates the environment by the traffic light duration.

Adaptation goal is the qualitative or quantitative software system quality factor that the system wants to achieve. The adaptation goal is quite intuitive, as the goal of a traffic light system is to reduce traffic, and the goal of a smart factory is to improve production efficiency without any safety crisis.

Based on this adaptation goal, the managing system manages the managed system. The managing system analyzes the environment and constructs the plan for the adaptation. For instance, a managing self-adaptive traffic light system monitors the traffic and calculates the traffic light's optimal duration. Then, this optimal adaptation plan is executed in the managed system.

3.1.1 Self-adaptive System Testbeds

This section introduces the details of testbeds for future evaluation in this study. In this research, two testbeds are used for the evaluation. Both testbeds represent the example of the cyber-physical system.

Self-adaptive Traffic Light

The flow of the cars changes as time goes by. There may be traffic congestion during the rush hour, and sometimes, there can be a traffic accident that disturbs other cars and make more traffic congestion. The self-adaptive traffic light system can be an excellent example for the CPS self-adaptive system that needs to be aware of anomalies like traffic accidents. The self-adaptive traffic light system predicts the number of cars based on the historical data. Based on the prediction of the future environment, the signal controller finds the optimal configuration of the traffic signal length that minimizes the number of waiting cars in the intersection.

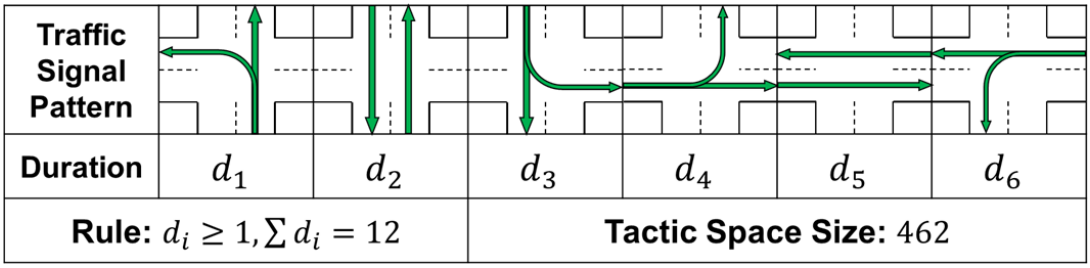

Figure 3.2: Traffic signal pattern for the self-adaptive traffic light

The self-adaptive traffic light system will be used as a virtual testbed in this research. Figure 3.2 shows the traffic signal pattern for the system. The one cycle of the traffic signal is composed of six component patterns. Each component must have at least one tick, which is, in this example, 10 seconds, and the sum of the duration of whole components must be 12 ticks, 2 minutes. In this configuration, the number of tactics must be 462.

Regarding 10 seconds as one tick, the simulation will be long for 24 hours, with 8640 ticks. Because the d_1 is more or equal than one, there is a time to adapt in one tick, 10 seconds. Therefore in this testbed, the reasonable time constraint is 10 seconds.

Self-adaptive Smart Warehouse

Smart Warehouse is the self-adaptive edge-computing-based system that targets to reduce the time to serve the order and exists as a physical testbed. Figure 3.3 shows the overview of the system. There are four types of items that the warehouse contains. The classification subsystem classifies the items based on their type and sends them to the repository subsystem. The repository subsystem holds the item and releases it when the order is received. The shipment subsystem sends the item to the destination.

The top view of the whole system is provided in figure 3.4. The item goes left in the right direction. The left-most device is the classification subsystem, the middle three are repository subsystem, and the rightmost device is the shipment subsystem. Each subsystem is consists of one edge server and one to three LEGO EV3 devices. The LEGO EV3 device controls the conveyor belt, which also performs the role of temporal storage of the item. The device communicates with the edge server, the raspberry pi, and follows the direction of the edge server. The edge servers collect the system state from devices and

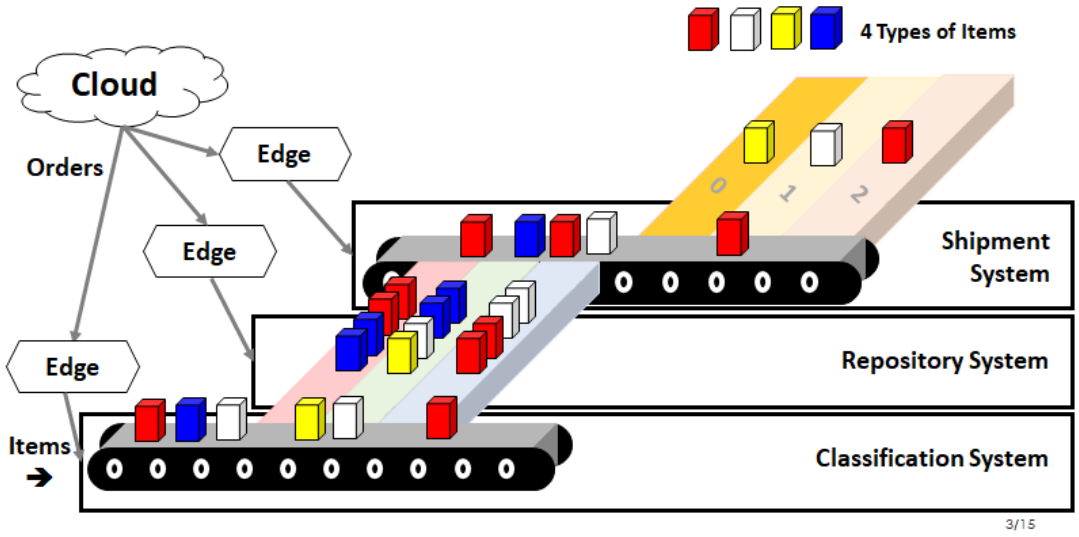

Figure 3.3: Overview of the self-adaptive smart warehouse system

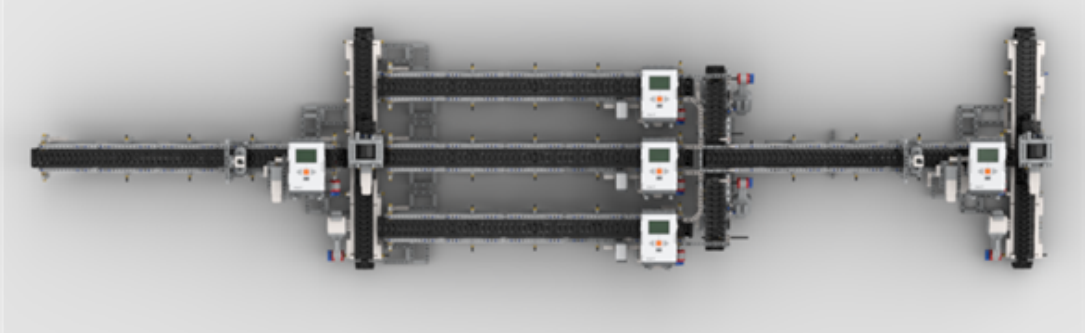

Figure 3.4: The top view of the Smart Warehouse System

other edge servers and decide based on the predefined rules. The decision-making of the edge server depends on the direction of the cloud server. The virtual customer sends the order to the cloud server. The cloud server examines the environment from the data of edge servers and selects the adaptations. Each communication is based on the HTTP post of API that edge or cloud server provides. The edge and cloud server is made with Python Django REST framework to efficiently respond to the communication and log the message and data.

The adaptation in the smart warehouse is based on the environment, orders. The orders are generated based on the rule with uncertainty. The smart warehouse can adapt the repository for each item type. For example, if there are three repositories, items 0 through 3 will store 0, 1, 2, 2 for each. Therefore there is a total of 81 adaptation tactics. The anomaly of the system is a stuck issue. The item naturally stopped middle of the conveyor belt because of the uncertainty of motors. In this research, only two types of stuck issues will be the target anomaly; the stuck on two side repository devices. The item can pass the repository subsystem after 50 seconds when the stuck has happened.

However, in this research, the overall reliability of physical components of the smart warehouse is not enough to evaluate with large amounts of orders and items. Therefore, the adaptation tactic changes into the single item level. It means that selecting the repository for each item will be the adaptation problem. Also, the anomaly will be artificially made in the system for a fair comparison. Moreover, to improve the reliability, the smart warehouse functions based on the discrete tick, which means, in one

tick, the classification subsystem and shipment subsystem can only serve one item, and the repository subsystem can only release one item per device.

Anomaly Generation

Since the comparison should be fair, the anomaly is generated artificially in this research. The cause of the anomaly of CPS is very different, and the distribution of the anomaly is also various. In this study, the anomaly assumes that it follows the wear-out failure rate.

$$p = 1 - e^{-t/C} \tag{3.1}$$

Equation 3.1 shows the probability of anomaly when the time after anomaly is t, and C shows the value that controls the distribution of the anomaly. The approximated version of the equation was used for the self-adaptive traffic light. With this anomaly probability equation, the simulator or the experiment system artificially makes an anomaly based on the given probability.

3.2 Reinforcement Learning (RL)

Reinforcement learning (RL) is a machine learning technique that targets an agent's actions based on the observed dynamic environment. The observed environment in the RL does not mean the same as the environment in the self-adaptive system. To the RL model, the environment is the environment of an external system and the system state. The RL model considers the set of state of the environment S, which is system states and external environment, set of actions A, and rewards $R_a(s, s')$, the reward after changing state from s to s' by doing action a. The final goal of the learning is that find the policy π that shows the probability of selecting actions at a certain state to maximize the accumulated reward. There are two kinds of techniques in RL, model-free, and model-based. Model-free algorithms are learned without any knowledge from the target environment, like the state transition function in Markov Decision Process. Generally, model-free algorithms learn based on the result of multiple trial-and-errors and repeatedly improve the policy. However, model-based algorithms can learn efficiently based on the information of reward functions. [19]

The following algorithms are common RL techniques:

- ***Deep Q-learning***

  Q-learning is a model-free RL technique that learns the potential reward Q of the action at the given state. With the number of results of trials, the value at selected action and state, $Q(s_t, a_t)$ is updated by the current rewards and estimated future rewards. [20] Deep Q-learning technique is originated by applying a deep learning algorithm to the original Q-learning algorithm. By implementing the convolutional neural network to RL and using replay memory. [21]

- ***Soft-Actor-Critic***

  Actor-Critic algorithm uses a two network, actor and critic. The actor network learns the fuction of probability of selecting the action π, and the critic network learns the value of the system state, V. [22] The Soft-Actor-Critic algorithm is applying the entropy on the training process. [23]

- ***Deep Deterministic Policy Gradient***

Policy Gradient algorithm is the policy-based algorithm that uses the gradient of the value function, Q in Q-learning. Policy π is defined as a stochastic function, and the reward for one episode establishes the differentiated policy's gradient direction. [24] Deep Deterministic Policy Gradient is a technique that applies actor-critic algorithm and deterministic policy function. The critic function assesses the value of the state and action and learns the policy based on the value of critic function by policy gradient method. [25]

In this research, reinforcement learning is used to react and select adaptation tactic by predicting the changing environment of the self-adaptive system. For the evaluation, the Deep Q-learning algorithm is selected and utilized to select the optimal adaptation tactic in a future environment and given anomalies.

3.3 Transfer Learning

Transfer learning technique saves the training time and cost and improves the machine learning model's performance by bringing the knowledge from the pre-trained model. Transfer learning is widely used for image classification tasks. [26]

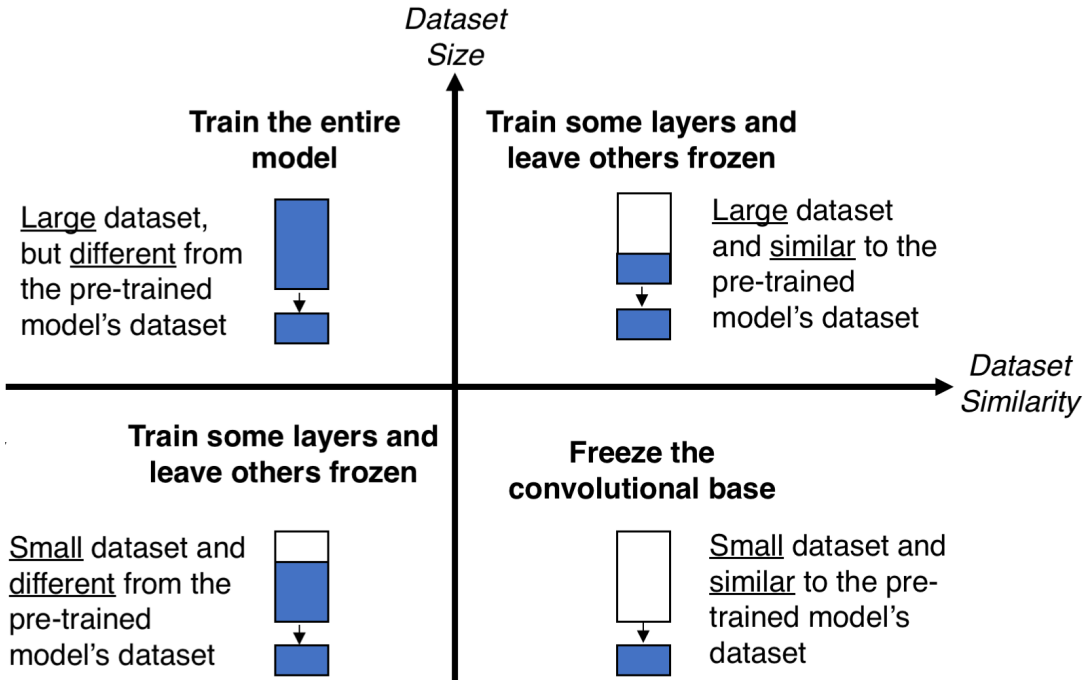

Figure 3.5: Transfer learning strategy from [2]

P. Marcelino (2018) proposes the strategies for applying the transfer learning technique based on the dataset size and similarity. Figure 3.5 shows the strategies. If the dataset is wealthy enough and the similarity is low, training the whole model is efficient. However, if the similarity is high and the size of the dataset is small, it is more efficient to train only the classifier or lowest layer. [2] In this research, the proposed approach applied the transfer learning to re-train the anomaly-free tactic planner for anomaly-specific tactic planners. The dataset size is enough because the research uses the modeled anomaly in the simulatable model. Therefore, in this research, the entire model will be re-trained. However, the speed and cost will spend less than training from scratch.

Chapter 4. Approach

4.1 Overview

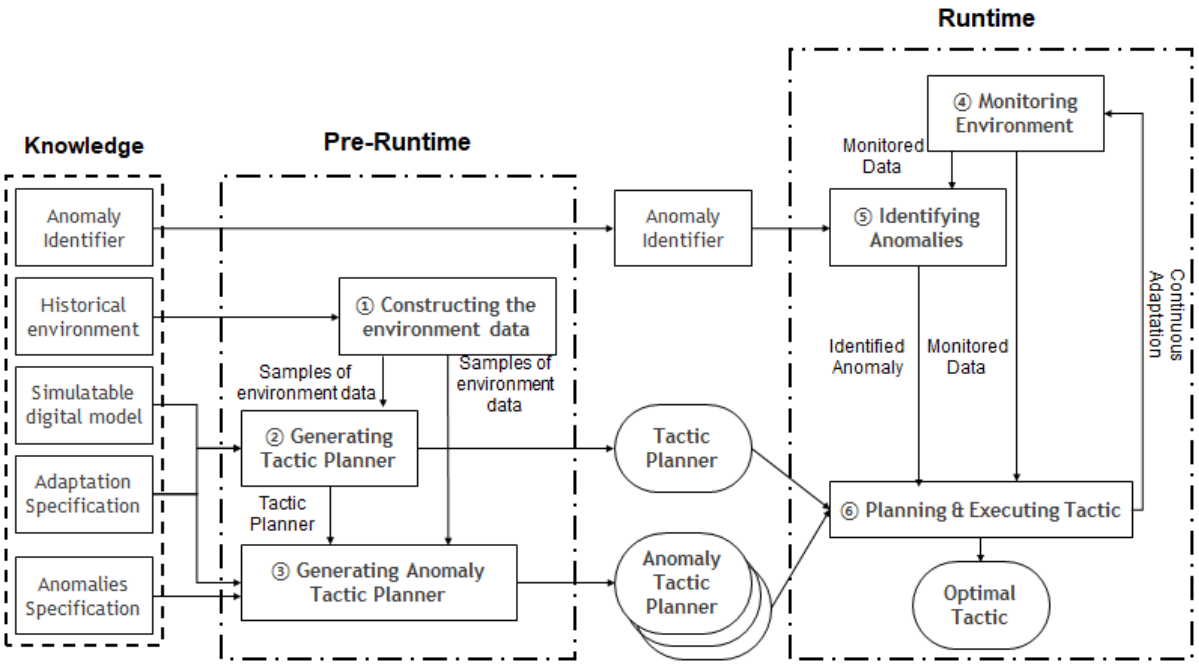

Figure 4.1: Overall A^4 process

This research proposes the A^4 approach, the Anomaly-Aware Adaptation approach. Figure 4.1 shows the overall adaptation approach of A^4. This approach aims to select the optimal tactic with a given system state while mitigating the effects of the potential CPS anomalies. The process is divided into pre-runtime and runtime. Before the process begins, A^4 requires knowledge of target anomalies and their specification, anomaly identifier, historical environment data, simulatable digital model, adaptation tactics, and goal specifications. In the pre-runtime phase, the tactic planner is generated. (Step 1) The generation process needs samples of the environment based on the historic environment data. (Step 2) The tactic planner generates reinforcement learning with the simulatable digital model and adaptation specifications. (Step 3) The generated tactic planner helps generate anomaly-specific tactic planners, which is the tactic planner for anomaly occurrence. By transfer learning, the anomaly tactic planner can be easily generated. (Step 4) In runtime, the system monitors the system and captures the environment. (Step 5) The anomaly identifier analyzes the system state and detects the anomaly. (Step 6) The generated tactic planner finds the optimal tactic with a given system state and identifies anomalies. Steps 4 to 6 will be repeated throughout the whole system runtime. The next sections will provide the detailed approach with the example based on the self-adaptive traffic light system.

4.2 Knowledge

Principle. The historical environment data is used to learn the system and environment in offline. Moreover, the A^4 requires the executable system model and an abstraction of the target cyber-physical self-adaptive system. The model can be written in any modeling language, and the model can contain any of the information that the engineer selected. However, it should be able to execute step by step and extract the system's state by logs. The simulatable model also contains the set of possible adaptation tactics, and each adaptation tactic can be executed by reconfiguring the system. The adaptation goal or reward shows the system's target, and it should be able to check based on the system state.

Table 4.1: Specifications for anomaly

	Principle	**Example** Self-adaptive traffic light system
Anomaly	Anomaly	Car accident
Occurrence	How many components occurs the same anomaly? How many types that anomaly has?	Four ways of the intersection
Behavior Change	How the system changes its behavior? How the anomaly disturbs the adaptation goal?	Reduce the outflow of the intersection
Average Impact	How much the anomaly effect the system?	Decrease the outflow by 50%
Average Duration	How long the anomaly effect the system?	1 hour (360 tick)
Identified by	How the system can identify the anomaly?	By cameras, police report
Identified after	How long the system can identify the anomaly?	After 10 seconds (1 tick)

The anomalies of the system are an essential part of the A^4. The anomaly of the CPSoS is the irregularly happened undesirable event that is caused by internal/external factors of CPSoS. The A^4 approach can target and mitigate the negative impact of the known anomalies of CPSoS when the engineer can define the specifications on table 4.1. The engineer should specify occurrence, behavior change, average impact, average duration, and identification of anomaly. Occurrence means the types of anomalies. The same anomaly can occur in different components with similar impacts. Behavior change and impact show how the system changes due to the anomaly. This factor needs to be implemented on the simulatable digital model. Moreover, the last thing is the identification process of anomalies.

Example. The main environmental factor for the self-adaptive traffic light system is the inflow of the intersection. The system can predict and sample the environment based on the historical data of the inflow of the intersection. The digital model simulates the number of waiting cars based on the inflow and length of the traffic light. There are 462 light traffic patterns as the system's possible adaptation patterns. The model can execute and generate the logs of each time unit. The goal is to minimize the

number of waiting cars.

The anomaly of this system is car accidents. This accident can occur on the four ways of the intersection. The anomaly changes the system's behavior by reducing the outflow of the intersection. On average, the anomaly decreases it almost 50%. This accident can be detected and identified by police call after a certain amount of time unit.

4.3 Constructing the Environment data

Principle. (Step 1) A^4 constructs the virtual environment data based on the historic environment data. This environment data is used for the environment of the simulatable digital model of system. The environment data should be non-deterministic, which means each sample should have differences and uncertainty. If the environment data is always the same, the tactic planner can be overfitted to the fixed environment. If the user have the average data as historical environment, the data-driven environment model generation approach can be useful. [27] Also, time-series forecasting techniques with some random initial value can be helpful to apply for sampling the environment, like using CNN or RNNs [28], injecting the random values based on random walk model [29]. The simple methods like injecting some random values or selecting the value with normal distribution can be another solution. The important thing is that the environment data should be realistic and non-deterministic. If enough amount of historical environment data already exists, this step can be skipped.

Example. The environment of the self-adaptive traffic light system is the inflow of cars. Therefore, the environment data is based on the historical car inflow data. There is the average data of historical environment on the open-source, it is not hard to collect the hour-based inflow data. Based on the data, the approach distribute the car inflow sample data of 24 hours for each tick, 10 seconds. There are periodic changes in car inflow. Therefore the periodic change is reflected by the sampling approach. The car inflow of one hour is divided into 10 seconds based on the distribution of total car inflow and randomly sampled based on that average flow. For example, if ten cars pass the intersection at noon, in the sample, ten cars will pass between 11:50 A.M and 12:10 P.M.

4.4 Generating Anomaly-Free Tactic Planner

Principle. (Step 2) A^4 generates the tactic planner by using reinforcement learning. Every reinforcement learning technique can be used for the generation, such as DQN [21], Deep Deterministic Policy Gradient[25] or any other else. Engineers can select the appropriate RL method based on the system's characteristics, adaptation tactics, adaptation goals, and anomalies. When A^4 generates the tactic planner, the simulatable system model performs a role of a simulation model of RL. With a sample of the environment, the system model gets the tactics from the RL model, changes the system state, and gives rewards and system state back to the model. For this time, the tactic planner does not consider any anomalies.

Example. The anomaly free tactic planner of the self-adaptive system is generated by the DQN [21] technique. It has a simple structure of three layers of the convolutional neural network and is optimized with SGD optimizer. The anomaly of the self-adaptive traffic light system is car accidents. It can occur in all of the four ways of the intersection.

4.5 Generating Anomaly-Specific Tactic Planners

Algorithm 1: A^4 Anomaly Tactic Planner Generation

Input : *targetAnomalies*, *anomalyFreeTacticPlanner*, *episodes*, *initialSystemState*, *systemModel*

Output: *anomalyTacticPlanners*

```
1  Procedure
2      anomalyTacticPlanners = [];
3      foreach anomaly in targetAnomalies do
4          anomalyTacticPlanner = RLModel(anomalyFreeTacticPlanner);
5          optimizer = optimizer(anomalyTacticPlanner.parameters);
6          foreach range(episodes) do
7              systemState = initialSystemState;
8              while isEnd(systemModel) do
9                  selectedTactic = selectTactic(systemState);
10                 reward, nextState = simulateOneStep(systemModel, systemState, selectedTactic,
                    anomaly);
11                 optimizer.optimize(systemState, selectedTactic, reward, nextState);
12                 systemState = selectTactic(nextState);
13             end
14         end
15         anomalyTacticPlanners.append(anomalyTacticPlanner)
16     end
17     return anomalyTacticPlanners
18 end
```

Principle. (Step 3) A^4 generates the anomaly tactic planner by using transfer learning. Algorithm 1 shows the pseudo-code of this step of the process. For each anomaly that A^4 targets, each anomaly tactic planner will be generated. In line 3, the anomaly is given to the simulation. These anomalies should be specified, and the simulatable digital model should be able to simulate the anomaly. In line 4 of algorithm 1, the anomaly-free tactic planner is imported. There are various transfer learning strategies, but the approach decided to re-train the whole model in this research because collecting the dataset is not hard with the simulatable model. However, the engineers can use other methods based on the situation. From lines 8 to 12, the model trains step by step based on the result of simulation with the specific anomaly. In this step, the simulation returns the result with the anomaly. This anomaly should remain the whole time during the training.

Example. Because there are four types of anomaly based on the location of the accident, the approach should generate four types of anomaly-specific tactic planners. Based on the anomaly-free tactic planner, the approach generates the anomaly-specific tactic planner. It imports the anomaly-free planner and re-train the model with remained parameters.

4.6 Monitoring and Identifying Anomaly

Principle. (Step 4) The system is on the runtime. It monitors the environment continuously and detects the needs of the adaptation. The environment can be monitored by the sensors that are connected with the physical parts of the system. The system state is also be monitored. (Step 5) The system detects the anomaly and identifies what exactly the anomaly is. There are various methods of anomaly identifications. Additional sensors for detecting anomalies or CPS anomaly detection algorithms can be used for physical anomalies like fault due to wear-out. Exceptions, assertions, or cyber-attack protection algorithms can be an example of anomaly identifications of cyber components.

Example. The traffic light system monitors the car inflow by using the camera that detects the car's movement. The system also can monitor the number of cars remaining at the intersection. The traffic light system detects the anomaly by the monitoring camera or the direct report from the police. The system also identifies the anomaly as a car accident, making the flow disturbance by 50%.

4.7 Planning and Executing the Tactic

Algorithm 2: A^4 Tactic Planning

Input : *environment*, *systemState*, *identifiedAnomaly*, *anomalyFreeTacticPlanner*, *anomalyTacticPlanners*

Output: *optimalTactic*

```
1 Procedure
2     if isAdaptationNeeded(environment) then
3         tacticPlanner = anomalyFreeTacticPlanner;
4         if isAnomalyHappened(identifiedAnomaly) then
5             tacticPlanner = anomalyTacticPlanner[identifiedAnomaly];
6         optimalTactic = tacticPlanner.selectTactic(environment, systemState);
7         return optimalTactic
8 end
```

Principle. (Step 6) Based on the monitored environment and system state, the A^4 approach decides whether there is an adaptation or not. Algorithm 2 shows the process of adaptation planning of the A^4 approach that dynamically changes the mode of tactic planner based on the identified anomaly. As shown in line 3, for the general cases without any anomalies, the anomaly-free tactic planner will plan the adaptation tactic and find the optimal tactic for the system. However, if there is an identified anomaly, the tactic planner is changed to an anomaly-specific tactic planner trained on the digital model with a particular anomaly, like in lines 4 to 5. The selected tactic planner then finds the optimal tactic based on the given environment and system state. Furthermore, the selected tactic is executed.

Example. Based on the monitored environment and system state, the adaptation process is triggered. The identifier shows an anomaly in the middle of the road of one way of the intersection. The approach finds the anomaly-specific tactic planner of that car accident and finds the optimal tactic. And then, the optimal tactic is executed.

Chapter 5. Evaluation

This section shows the evaluation of the A^4 approach to present the effectiveness among other adaptation approaches.

5.1 Research Questions

There are research questions that are related to the motivation and goal mentioned in section 1.2.

- Research Question 1 (RQ1): *Cost Efficiency.* How does A^4 handle adaptation within time constraints?
- Research Question 2 (RQ2): *Adaptation Effectiveness.* How does A^4 handle the anomalies, unlike other approaches?
- Research Question 3 (RQ3): *Relationship between Anomaly and Performance.* How adaptation effectiveness of A^4 changes for various anomaly occurrence probability?

5.2 Experimental Design

This section introduces the experimental design of the research. The testbeds mentioned in section 3.1.1 is used for the experiments. Table 5.1 shows the detailed descriptions of testbeds.

RQ1. Cost Efficiency

The A^4 approach is intended to handle the self-adaptive system's adaptation process with time-constraint adaptation limitations and huge adaptation space. This research experiments to compare the adaptation cost of the A^4 approach and other adaptation approaches in the runtime to check the cost-efficiency. Especially in this research, the time complexity between adaptation approaches will be shown.

The self-adaptive traffic light system will be used for this experiment. For each adaptation approach, the same environment - car inflow, the same anomaly will be given. The system will run for the whole 24 hours, and the system waits during the adaptation is planning. The time before and after adaptation will be measured, and the sum of every adaptation planning time will be the result. Every approach will experiment 20 times.

RQ2. Anomaly-Aware Effectiveness

When MC-based adaptation approaches handle the anomaly by fluctuating the system model, and the online RL adaptation approach learns the changed behavior, the A^4 handles the anomaly by preparing the anomaly-specific tactic planner based on the RL technique. For this research question, the adaptation performance is compared between the approaches. The system's goal or the reward of the system will be shown.

Table 5.1: Detailed descriptions for testbeds

Testbed	**Traffic Light**	**Smart Warehouse**
Environment	Car inflow to the target intersection	Recevied order to the warehouse
Environment Complexity	8 factors (Cars move from 4 ways of the road to 2 destinations - straight, and left.)	Single factor (Item types of the order)
Historical Environment	Open traffic data from Seoul, Republic of Korea [30] - Hour based daily traffic data	Order is randomly selected based on the rule - there must be each one of item in the group of orders
System Model	Cars move into the intersection based on the environment, go away based on the traffic light	Warehouse gets and serves the order by controlling the repository and shipment subsystem. The classification subsystem orders items for new inventory
System State	Number of cars waiting in the intersection	Number of orders left, Current inventory status
Sensors	Camera that senses traffic flow	Various sensors attached at device, Cloud can get the orders.
Actuators	Traffic lights	Item classifier that moves item from classification subsystem to repository
Adaptation Goal	Minimize the number of cars waiting in the intersection	Serves the orders as fast as it can
Adaptation Space	462	3
Adaptation Cycle	Adapt for every traffic light cycle	Adapt for each item
Reward Assessment	Number of cars waiting in the intersection	Reward that considers potential income of the warehouse, and the time for the order serves
Anomaly	Car accident	Jammed item
Anomaly Occurrence	Four ways of the intersection	three of two repository device
Behavior Change	Reduce the outflow of the intersection	slows the item flow
Anomaly Impact	Decrease the outflow by 50%	Item slowly moves from repository to shipment at 25% of normal speed
Anomaly Identifier	By sensors (cameras), police report	Sensors on the repository device
Anomaly Identified after	After one adaptation cycle	After 1 tick passed

The self-adaptive traffic light system and self-adaptive smart warehouse system are the testbeds for this experiment. Because this experiment measures anomaly-aware performance, each run of the experiment will have at least one anomaly at a time. The experiment process of the self-adaptive traffic light system is the same as mentioned in section 5.2. However, this experiment measures the reward - the number of cars waiting in the intersection.

The experiment on smart warehouse systems will be based on the digital model simulation and the physical testbed. The environment and anomalies of the self-adaptive smart warehouse system will be given equally between the approaches. Because this system should adapt the tactic at max one second, the RL-based approach and random approach will be only used for comparison. The experiment ends when the system serves all of the fixed numbers of orders. Predefined reward and the time until the experiment ends will be served as a result. The experiment will run 20 times for each approach.

Table 5.2: Experimental setups for testbeds

Testbed	**Traffic Light**	**Smart Warehouse**	
		Physical	**Virtual**
Experiment Ends	After 8,640 ticks (24 hours)	After all the orders completed to serve	
		Total 20 Orders	Total 10,000 orders
Rewards	-	Order complete: 30 Item removed: -70 Order waiting: -1	
Capacity of Conveyor Belt	-	5	20
Anomaly Constant	108,000	2	30,000,000
Anomaly Duration	360 ticks (1 hour) Anomaly does not reoccur until 720 ticks (2 hour) after the anomaly ends.	10 ticks	100 ticks
Execution CPU	Intel Core i7-7700	Cloud: Intel Xeon Gold 6140 Edge: ARMv7 Device: TI Sitara AM1808	AMD Ryzen 7 5800X
OS	Windows 10 Pro	Cloud: CentOS 8.4 Edge: Raspbian 10 Device: EV3 MicroPython	Windows 10 Pro
RAM	16GB	Cloud: 16GB Edge: 1GB Device: 64MB	32GB
Implemented	Python 3.9	Cloud: Python 3.9 Edge: Python 3.7 Device: Micropython 1.9.4 & pybricks 3.0	Python 3.9

RQ3. Relationship between Anomaly and Performance

Because the anomaly massively affects the system's performance, the anomaly occurrence rate can affect the performance of the adaptation approach. Because of the limitation of the testbed, the experiments for RQ1 and RQ2 are conducted with a larger anomaly occurrence rate than the actual rate. Therefore, RQ3 will show the adaptation effectiveness on various anomaly occurrence rates.

The experiment for this research question is on the self-adaptive traffic light. Also, only A^4, and the ORL approach will be executed. Other experiment designs will be the same as a section 5.2. However, the number of anomalies will be changed. For each approach and anomaly constant, the simulation will run 1,000 times.

Table 5.3: Experimental setups for A^4 and ORL approach

Input Size	**Traffic Light**: 9 (current tick, the number of cars for each inflow) **Smart Warehouse**: 10 (current tick, current inventory status for repository and shipment, current order remains for each item type)
Size of Hidden Layer	**Traffic Light**: 256 **Smart Warehouse**: 512
Structure of CNN	Linear regression (input to hidden layer) ReLU layer Linear regression (hidden to hidden layer) ReLU layer Linear regression (hidden to size of adaptation space)
Loss Function	Cross Entropy Loss
Optimizer	**Traffic Light**: RMSprop **Smart Warehouse**: SGD
Learning Rate	**Traffic Light**: 0.01 **Smart Warehouse**: 0.001
Number of Episodes	**Traffic Light**: 500 & 100 (Transfer Learning) **Smart Warehouse**: 150 & 50 (Transfer Learning)
Epsilon Decay	0.9 to 0.05 by decay rate 200
Gamma	**Traffic Light**: 0.8 **Smart Warehouse**: 0.9
Memory Size	10,000
Batch Size	**Traffic Light**: 64 **Smart Warehouse**: 128

5.3 Experimental Setups

The experimental setups are provided in table 5.2. This research also implemented statistical model checking (SMC) [8], online RL (ORL) [11], and RL with SMC method (RL-SMC) inspired from [17] to compare with A^4 approach. The RL-SMC methods first extract the best tactic and tactics that record the reward threshold, which is set by the reward of the best tactic. Moreover, the SMC verifies the tactic

Table 5.4: Experiment setup of anomaly constant

Testbed	**Anomaly Constant**	E(**time between anomalies**)	E(**number of anomalies**)
Traffic Light (RQ1 & 2)	**108,000**	1,426.82	3.45
Smart Warehouse (Physical)	**5**	2.87	1.55
Smart Warehouse (Simulation)	**30,000,000**	6,838.24	1.44

and chooses the best choice. As mentioned in section 5.2, for the traffic light system, the whole four approaches will be the target, and for the smart warehouse, only RL-based approaches, A^4 and ORL, and random approach will be the target approach.

The A^4 approach and ORL approach used the same CNN model with the Deep Q-learning algorithm [21] and the same hyperparameters to train the model. The pretraining process was also the same as the ORL approach. Therefore the only difference between the ORL model and anomaly-free tactic planner of A^4 is the online learning process. Table 5.3 shows the setups for the A^4 and ORL approaches. SMC approaches used simple monte carlo simulation algorithm and the number of samples was 10,000.

Table 5.4 shows the anomaly constant settings for each testbed. Each value is calculated based on the setups of the anomalies. The traffic light system has approximately 3 or 4 anomalies for each simulation to show the anomaly at various times like dawn, morning, afternoon, or night. All of the experiments should have at least one anomaly.

5.4 Results & Analysis

5.4.1 RQ1. Cost Efficiency

Table 5.5: Average adaptation time for each approach

Approach	**Average Adaptation Time**
A^4	0.584ms
ORL	1.887ms
RL-SMC	25.49s
SMC	40.30s

This section shows the result of the RQ1. Table 5.5 shows the average adaptation time for each adaptation approach on the self-adaptive traffic light system. The results show that the A^4 and ORL approaches take less than one second for the single adaptation process, while other MC-based approaches take more than ten seconds. RL-SMC approach successfully reduces the size of the adaptation. However, it costs more than 20 seconds to adapt, violating the time constraint of this testbed. The cost of SMC can be reduced when if it uses the SMC approach like simple Monte-Carlo simulation [31]. However, it will sacrifice the performance of the approach rather than the SPRT algorithm. ORL approach slightly takes longer than the A^4 approach because of the online learning process. However, it is not critical to violate the time constraint. A^4 shows the best result among other approaches.

5.4.2 RQ2. Anomaly-Aware Effectiveness

Self-adaptive Traffic Light

Table 5.6: Average number of cars waiting for each approach

Approach	**Average number of cars waiting** (cars, 20runs)
A^4	54.51
ORL	60.89
RL-SMC	54.74
SMC	54.07

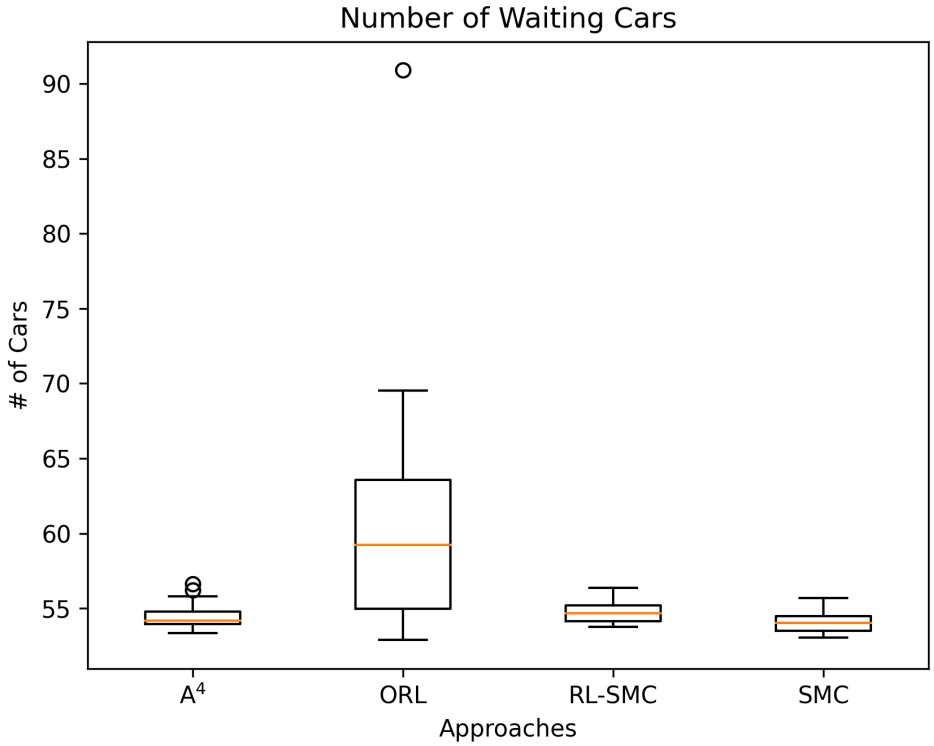

Figure 5.1: Box plotted result of average number of cars waiting

This section shows the result of the RQ2. Table 5.6 shows the average number of cars waiting at the intersection for each adaptation approach on the self-adaptive traffic light system. Figure 5.1 also shows the box plotted result of the effectiveness on the self-adaptive system. All of the four approaches showed 50 to 60 cars in the average of 24 hours simulation. Especially, A^4, RL-SMC, and SMC approach all showed a similar result in average reward values and showed very stable results on every run. Although the RL-based approaches and the A^4 approach have not proven the convergence yet, based on the results in the traffic light system, it can argue that the proposed approach has competitiveness among other approaches. ORL approach shows approximately 60 cars on average. However, it shows large dispersion and records the highest value among the approaches. The difference between other approaches and the ORL approach seems not that high. However, this difference is mainly caused by anomalies.

Figure 5.2 is an one-run example result of the experiment of RQ2. The red zone shows that the anomaly happens at that time. In this example, there were three anomalies at almost 4 A.M., 11 A.M.,

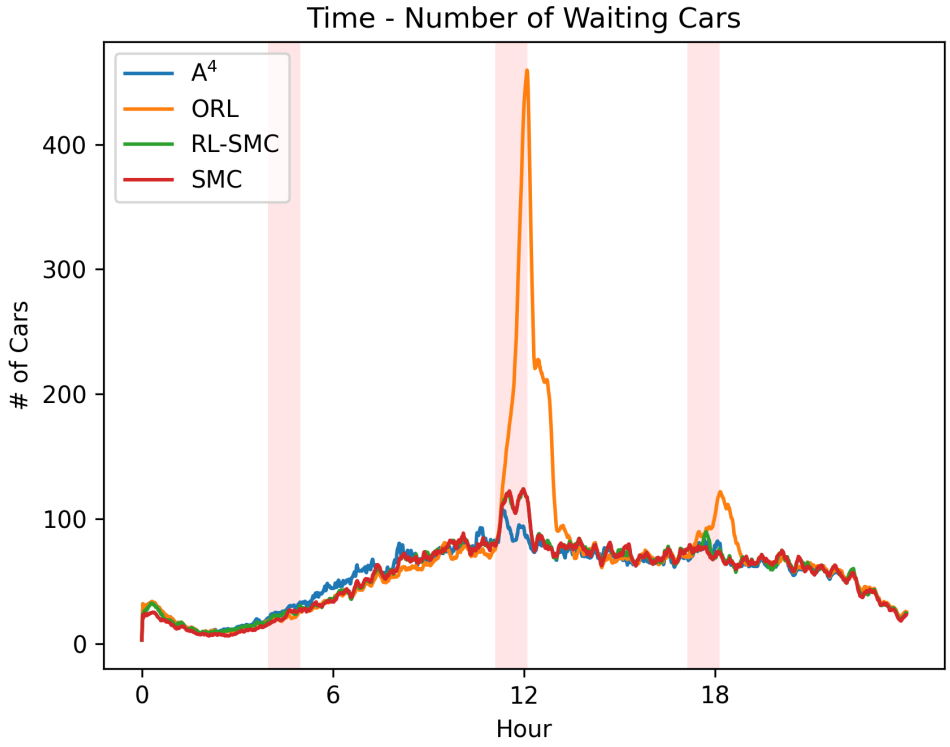

Figure 5.2: Example result of the traffic light system

and 5 P.M. The anomaly does not affect the performance at 4 A.M. when the cars do not pass a lot. However, at 11 A.M. and 5 P.M., the anomalies do affect the performance of ORL. The online RL algorithm is good to adapt the model by the changing system and environment continuously. However, the online RL algorithm needs time to change itself by collecting enough data. In this experiment, the RL algorithm cannot collect enough data for online learning. It can only collect 30 minutes of data. Moreover, the online RL algorithm has another disadvantage when the anomaly is resolved. The system changes once more, and the online RL algorithm gets damaged because of the memory of anomalies.

Self-adaptive Smart Warehouse

Table 5.7: Average performance of simulation of smart warehouse for each approach

Approach	Average reward	Average processed time
A^4	202,275.4	10,012.8
ORL	169,765.6	10,016.1
Random	140,883.4	10,019.8

This section is the result of an experiment of RQ2 that runs on the smart warehouse system testbed. Table 5.7 shows the effectiveness of each approach on the simulated version of the smart warehouse system. Moreover, figures 5.5 and 5.6 shows the box plot of the result. The A^4 approach dominates the ORL and random approach. The ORL approach records higher rewards than the random and also records similar values on some runs. However, the average result was much lower than the A^4 approach. The processing time seems not that different among the approaches. Nevertheless, the A^4 approach

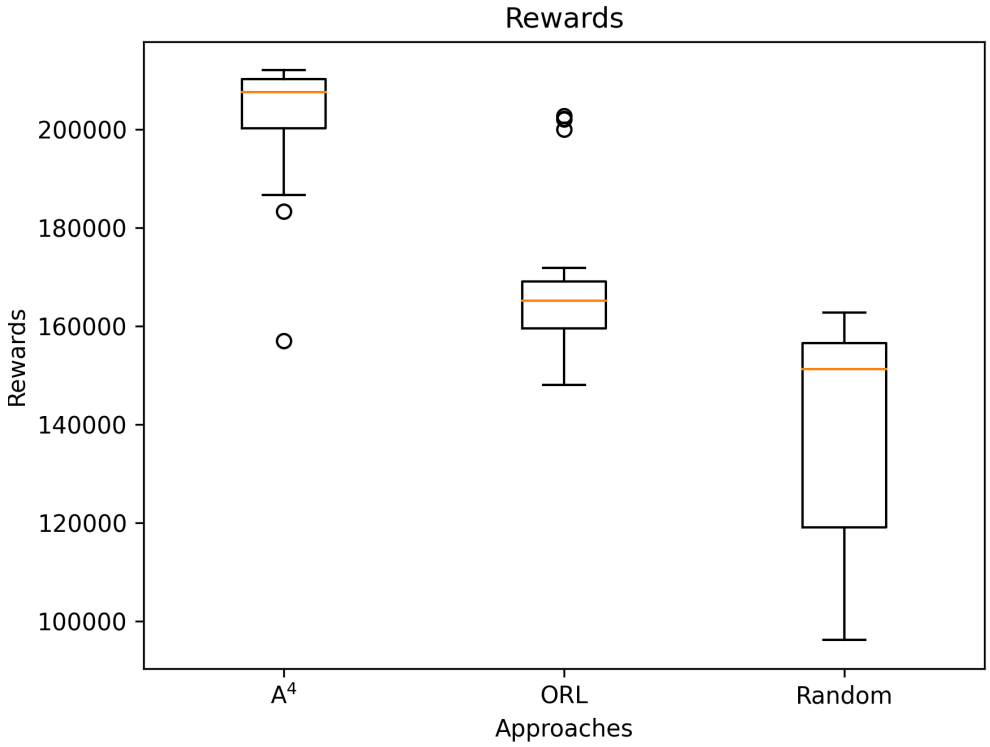

Figure 5.3: Box plotted result of reward of the simulation of smart warehouse system

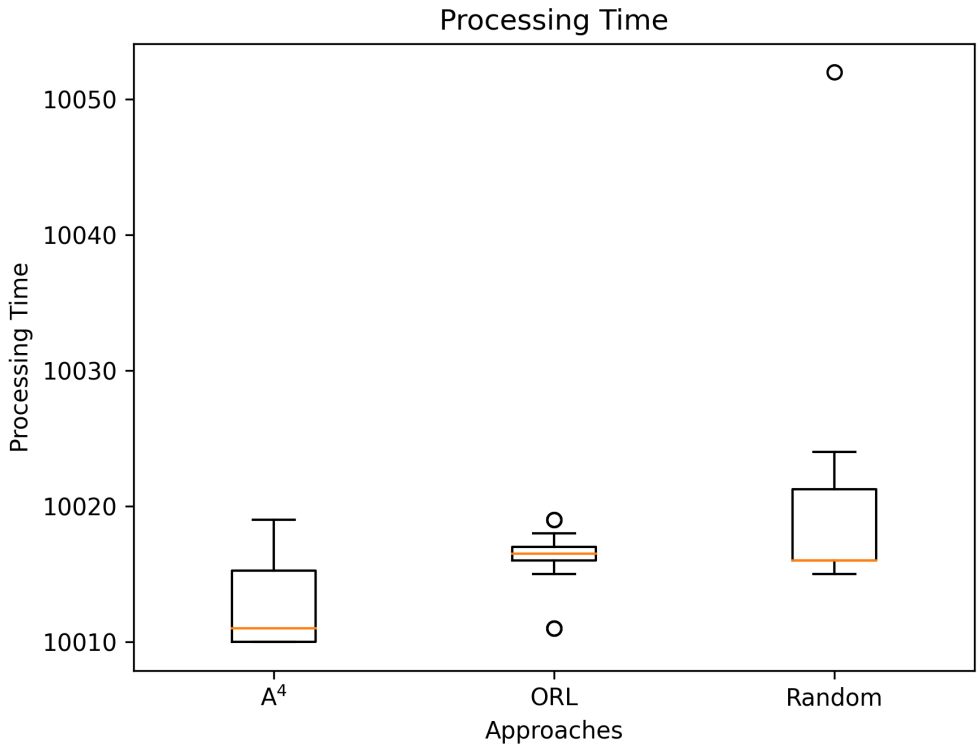

Figure 5.4: Box plotted result of processed time of the simulation of smart warehouse system

seems better result than others.

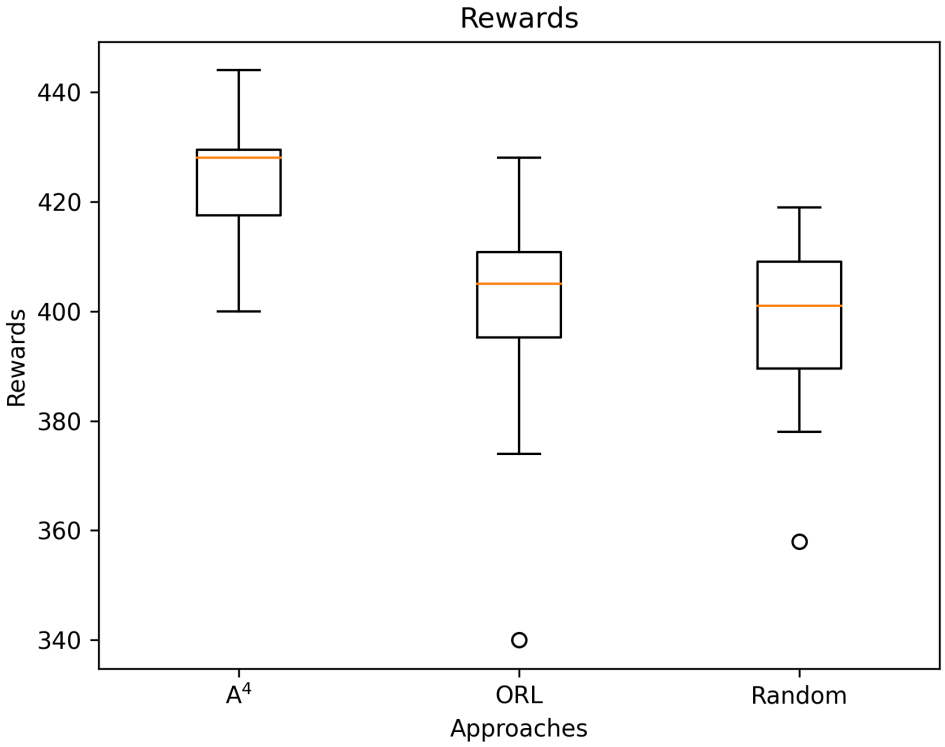

Figure 5.5: Box plotted result of reward of the simulation of smart warehouse system

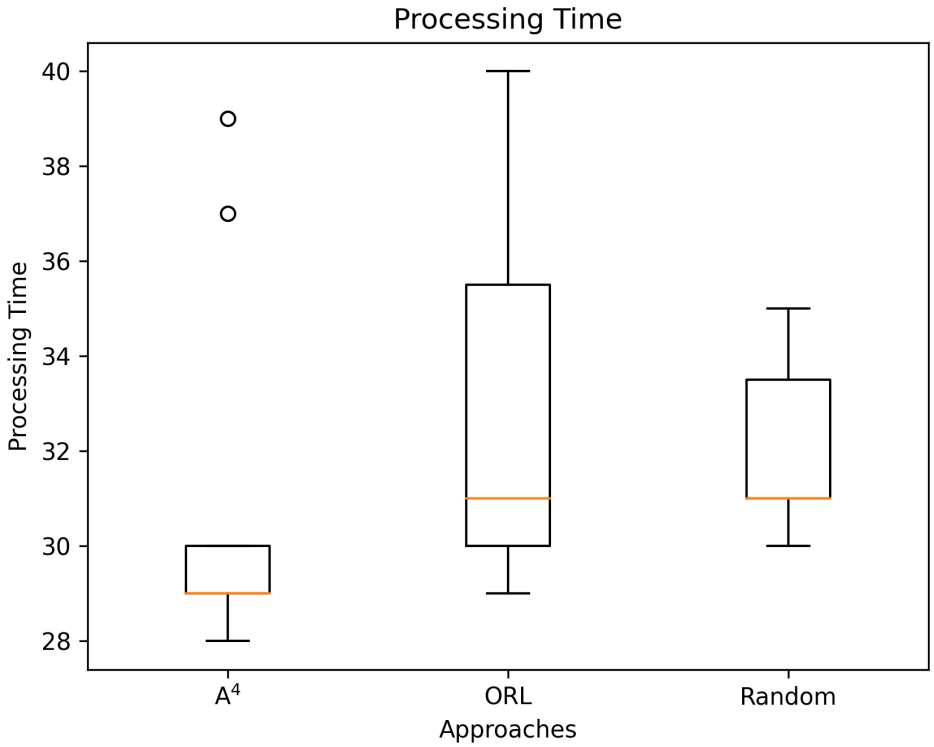

Figure 5.6: Box plotted result of processed time of the physical smart warehouse system

Table 5.8: Average performance of physical warehouse testbed for each approach

Approach	Average reward	Average processed time
A^4	424.33	30.33
ORL	385.53	32.93
Random	397.40	32.13

Based on the result of the simulation session, the experiment was conducted on the real smart warehouse. Table 5.8 shows the result of the experiment on the real system, and figures 5.5 and 5.6 shows the box plot result. Because the size of the experiment is smaller than the simulation, the difference between approaches does not seem clear. However, the A^4 constantly records the high rewards and low processing time based on the box plotted result. It means that the A^4 approach is effective not only in the simulation but also in the real world.

5.4.3 RQ3. Relationship between Anomaly and Performance

Table 5.9: Average number of cars on different number of anomalies

# of Anomalies	A^4	ORL
0	53.23	52.75
1	53.92	56.75
2	54.59	59.03
3	54.94	61.27
4	55.47	63.75
5	56.33	65.25
6	56.59	68.45
7	56.89	70.82

This section is the result of the experiment of RQ3. Table 5.9 shows the average number of cars for each number of anomalies. Figures 5.7 also show the number of cars change based on the anomaly. As predicted, the average number of cars increased for both approaches as the anomaly frequency increased. The ORL approach shows a good result when there is no anomaly. However, the A^4 approach shows that the increased values are not that high as the ORL approach. A^4 approach increased only four cars when the ORL approach increased almost twenty cars on seven anomalies. It shows that the A^4 approach is better than the ORL approach to manage the anomaly and mitigate the performance degradation. Moreover, the A^4 approach can manage the situation when there are many anomalies.

5.5 Threats to Validity

This section shows the threats to validity of this research, and provides the methods that prevent the threats. One threat is the RL model selection of A^4. This research is conducted based on the Deep Q-learning algorithm which is action-reward function-based and supports discrete tactic space. This algorithm has a limitation that does not support the continuous tactic space. This threat is reduced by the fact that many of self-adaptive systems has discrete set of adaptation tactics. In addition, other RL algorithms also can be utilized instead of the Deep Q-learning.

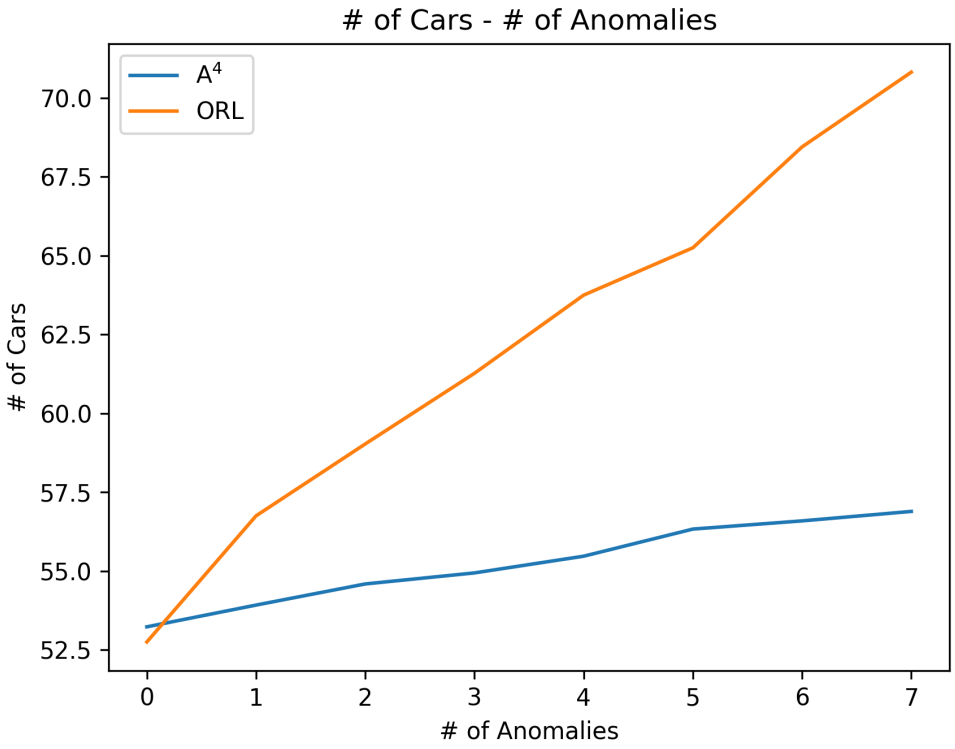

Figure 5.7: Number of cars change based on the number of anomalies

Another threat is the fair comparison between the A^4 and baseline approaches such as online RL, or statistical model checking. In this research, the implementation of baseline approaches are followed the approach from the following studies. [8, 11, 17] Moreover, the complexity, environment, anomalies, and pre-trained RL model was equally given through the baselines.

Moreover, the experiment is conducted on small size of CPSoS. In this research, the single traffic light, and smart warehouse is given as the experiment subject. However, the A^4 approach can be applied to any of CPSoS that has known anomalies, and the subjects are selected to show the comparison between approaches with large amount of experiments. Therefore, it is highly sure that the A^4 approach can show the similar experiment result with this evaluation.

Chapter 6. Conclusion

The environment that CPSoS face is various and keeps changing and evolving. The self-adaptation is the key to achieving the CPSoS's goal in various environments by adapting the system's behavior. Model-checking and reinforcement learning is the existing works to handle the uncertainty of the environment and achieve the goal. However, in the CPSoS, the approaches cannot fully handle the time constraints and anomalies.

The A^4, Anomaly-Aware Adaptation Approach, is proposed to solve both time and anomaly problems in this research. The A^4 generates anomaly-specific tactic planners using transfer learning and plans the adaptation when the anomaly happens. The A^4 shows that it is fast enough not to violate any time constraints of the testbed, and the adaptation performance on various anomalies is also shown in the physical testbeds.

Limitations

One limitation is the hardness of the application of A^4. This approach needs various knowledge, such as specifications of anomalies and anomaly identifiers. It is nearly impossible to know all of the anomalies that can happen in the system. Moreover, identifying anomalies in the field that is started to be actively researched recently and many technologies have not yet been developed. It is also challenging to develop a digital model that imitates the anomalies.

Another limitation is the inefficiency of anomaly-specific tactic planners. Making a separate tactic planner may not be difficult with using transfer learning. However, if there are many anomalies and the engineer should consider the situation that the multiple anomalies happen together, the number of anomaly-specific tactic planners grows exponentially like the state explosion problem.

The proof of convergence is also the limitation. Although the evaluation result of this research seems good enough to use, it is unknown that the RL models - tactic planners will converge for every self-adaptive system.

Future Work

The A^4 approach shows enormous potential for utilizing the self-adaptive system with anomalies. However, it also shows various potential future works to validate and improve the A^4 approach. Future work plans to solve the limitation of tactic planners by generating the integrated anomaly tactic planner by encoding a system state as a few multiple vectors and adding anomaly as another vector. Another future work is the integrated approach with the anomaly identifier using deep learning. For example, the deep-learning-based anomaly identifier identifies and sends the result to the anomaly-specific tactic planner. The experiment on various level of CPSoS is also planned to overcome the threat to validity of the evaluation. In this research, it is experimented on the single traffic light. For future work, it is planned to experiment on multiple traffic lights or a city block.

Bibliography

[1] D. Weyns, "Software engineering of self-adaptive systems," in *Handbook of Software Engineering*. Springer, 2019, pp. 399–443.

[2] P. Marcelino, "Transfer learning from pre-trained models," *Towards Data Science*, 2018.

[3] Y.-J. Shin, L. Liu, S. Hyun, and D.-H. Bae, "Platooning legos: An open physical exemplar for engineering self-adaptive cyber-physical systems-of-systems," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2021, pp. 231–237.

[4] F. D. Macías-Escrivá, R. Haber, R. Del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems with Applications*, vol. 40, no. 18, pp. 7267–7279, 2013.

[5] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 1–32.

[6] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[7] M. D'Angelo, A. Napolitano, and M. Caporuscio, "Cyphef: a model-driven engineering framework for self-adaptive cyber-physical systems," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 2018, pp. 101–104.

[8] Y.-J. Shin, E. Cho, and D.-H. Bae, "Pasta: An efficient proactive adaptation approach based on statistical model checking for self-adaptive systems," *Fundamental Approaches to Software Engineering*, vol. 12649, p. 292, 2021.

[9] C. Stevens and H. Bagheri, "Reducing run-time adaptation space via analysis of possible utility bounds," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1522–1534.

[10] J. Cámara, H. Muccini, and K. Vaidhyanathan, "Quantitative verification-aided machine learning: A tandem approach for architecting self-adaptive iot systems," in *2020 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2020, pp. 11–22.

[11] A. Palm, A. Metzger, and K. Pohl, "Online reinforcement learning for self-adaptive information systems," in *International Conference on Advanced Information Systems Engineering*. Springer, 2020, pp. 169–184.

[12] J. Plasse, J. Noble, and K. Myers, "An adaptive modeling framework for bivariate data streams with applications to change detection in cyber-physical systems," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2017, pp. 1074–1081.

[13] H. Kopetz, A. Bondavalli, F. Brancati, B. Frömel, O. Höftberger, and S. Iacob, "Emergence in cyber-physical systems-of-systems (cpsoss)," in *Cyber-physical systems of systems*. Springer, 2016, pp. 73–96.

[14] M. A. Nia, M. Kargahi, and F. Faghih, "Probabilistic approximation of runtime quantitative verification in self-adaptive systems," *Microprocessors and Microsystems*, vol. 72, p. 102943, 2020.

[15] D. Kim and S. Park, "Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software," in *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 2009, pp. 76–85.

[16] M. Zhang, J. Li, H. Zhao, K. Tei, S. Honiden, and Z. Jin, "A meta reinforcement learning-based approach for self-adaptive system," *arXiv preprint arXiv:2105.04986*, 2021.

[17] F. Quin, D. Weyns, T. Bamelis, S. S. Buttar, and S. Michiels, "Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2019, pp. 1–12.

[18] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Engineering self-adaptive systems through feedback loops," in *Software engineering for self-adaptive systems*. Springer, 2009, pp. 48–70.

[19] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[20] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[22] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[23] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[24] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[26] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[27] Y.-J. Shin, Y.-M. Baek, E. Jee, and D.-H. Bae, "Data-driven environment modeling for adaptive system-of-systems," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 2019, pp. 2044–2047.

[28] B. Lim and S. Zohren, "Time-series forecasting with deep learning: a survey," *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2194, p. 20200209, 2021.

[29] F. Spitzer, *Principles of random walk*. Springer Science & Business Media, 2013, vol. 34.

[30] *Research data of traffic in Seoul, 2020.* Seoul Metropolitan Government, 2021. [Online]. Available: https://news.seoul.go.kr/traffic/files/2012/02/6058855d14fa49.45283783.pdf

[31] B. K. Aichernig and R. Schumi, "Statistical model checking meets property-based testing," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST).* IEEE, 2017, pp. 390–400.

Acknowledgments in Korean

카이스트, 대전에 온지 이제 7년이 지나, 석사 과정을 마치고 학위 논문을 마무리하게 되었습니다. 석사 논문 심사를 마치고 되돌아 보니, 지금까지의 제 여정은 제 역량만이 아니라 많은 분들의 도움과 지도로 이루어졌음을 느끼게 됩니다.

먼저, 학부 2년, 석사과정 2년, 합쳐 4년간 저를 지도해주신 배두환 교수님께 깊은 감사의 뜻을 전합니다. 학부생때 처음 만났 뵈었을 때, 처음 만난 제게 넘치는 기회를 주시고, 제게 연구란 무엇인지 성심성의껏 가르쳐주신 교수님을 학부생 때부터 지도 교수님을 만난건 제게 큰 복이었음을 느끼게 됩니다. 연구적으로, 인간적으로 절 지도해주시고, 제게 연구자란 무엇인가 알게해주셨습니다. 교수님과 연구 미팅하고 대화했던 그 시간이 너무나 즐거웠고, 감사했습니다. 교수님의 이름에 부끄럽지 않은 제자가 되도록 노력하겠습니다.

그리고, 제 석사 논문 심사에 참여해주시고, 좋은 조언과 지도를 해주신 고인영 교수님께도 감사의 뜻을 전합니다. 앞으로 있을 박사과정 기간에 교수님의 지도를 받게 되어 더욱 기대가 됩니다. 앞으로도 잘 부탁드립니다. 논문 지도와 심사를 해주신 지은경 연구교수님께도 감사합니다. 논문 심사 직전까지도 바쁘신 와중에 제게 조언을 아끼지 않아주셔서 너무 감사합니다. 박사과정에 진학해서도 잘 부탁드립니다.

소프트웨어 공학 연구실의 선배, 동기, 후배님들께도 모두 감사합니다. 4년간 함께 연구하고, 조언해주었던 용준이형 정말 고맙습니다. 대선배로서의 조언과 동시에 멋진 사진을 남겨주셨던 영민이형, 제 옆자리에서 응원해주셨던 지영이 누나, 대화할 때마다 즐겁고 편안한 Zele, 지금은 연구실에 없지만 작년 내내 함께하고 즐거워했던 수민이형, 급할 때 붙잡고 말해도 언제나 좋은 조언을 해주었던 상원이형, 석사 선배로 다양한 꿀팁을 줬던 Lingjun, 동기로 2년... 1년 반동안 함께했던 May, 1년간 제가 너무 많은 일을 시키고 힘들었을 거 같아 미안한 한수씨, 미현씨, 그리고 언급하지 못했지만, 여기서 만난 모든 연구실 가족 분들 감사합니다. 좋은 사람들을 만나서 함께했기에, 제가 연구실 생활에서 힘을 얻고 즐겁게 연구를 진행할 수 있었습니다. 정말 감사합니다.

항상 저를 위해 응원하고, 기도해주시는 우리 부모님과 가족들, 친구, 지인분들께도 정말 감사드립니다. 부모님의 일이 힘들고 괴로운 가운데에서도, 혹여나 혼자 있을 아들이 힘들지 않을까, 어렵지 않을까 걱정하시며 기도하고 응원해주셨기에 제가 석사과정 생활을 잘 마무리할 수 있었습니다. 항상 기도를 아끼지 않으시고 저를 응원해주시는 할머니, 할아버지, 그리고 지면에 다 담지 못했지만, 절 위해 기도해준 가족들, 그리고 친구들에게 정말 감사합니다.

마지막으로, 주님께서 제 주님이 되셔서 이 모든 것을 가능케 하시고, 이 모든 사람들을 만나게 해주셨기에 여기까지 올 수 있었습니다. 석사 과정동안 저와 동행하신 하나님 아버지께 감사드립니다.

2022년 1월. 조은호 올림.

Curriculum Vitae in Korean

이 름: 조 은 호

생 년 월 일: 1997년 07월 11일

전 자 주 소: ehcho97@gmail.com

웹 사 이 트: https://eunhocho.github.io

학 력

2013. 3. – 2015. 2. 고등학교 (2년 수료)

2015. 2. – 2020. 2. 한국과학기술원 전산학부 (학사)

2020. 3. – 2022. 2. 한국과학기술원 전산학부 (석사)

경 력

2017. 9. – 2018. 2. (주)카카오페이지 인턴

2018. 7. – 2018. 8. Institute for Energy Technology (IFE) 인턴

2018. 9. – 2021. 8. 한국과학기술원 전산학부 조교

학 회 활 동

1. Y.-M. Baek, **Eunho Cho**, Y.-J. Shin, and D.-H. Bae, *A Modeling Method to Represent Geographical Information of a System-of-Systems*, 16th Annual System of Systems Engineering Conference (SoSE) 2021, Sweden, June., 2021.
2. **Eunho Cho**, J. Yoon, D. Back, D. Lee, and D.-H. Bae, *DNN Model Deployment on Distributed Edges*, International Workshop on Big data driven Edge Cloud Services (BECS) 2021, France, May., 2021.
3. Y.-J. Shin, **Eunho Cho**, and D.-H. Bae, *PASTA: An Efficient Proactive Adaptation Approach Based on Statistical Model Checking for Self-Adaptive Systems*, 24th International Conference on Fundamental Approaches to Software Engineering (FASE) 2021, Luxembourg, March., 2021.
4. **Eunho Cho**, S. Park, L. Liu, C. K. K. Anthony and D.-H. Bae, *시스템 오브 시스템즈의 런타임 검증 속성 명세 유형 분석 및 적용*, Korea Software Congress (KSC) 2020, Korea, December., 2020.
5. **Eunho Cho**, Y.-J. Shin, E.-K. Jee, and D.-H. Bae, *결함-공격 트리 기반 안전성 및 보안 분석 기법 비교*, Korea Software Congress (KSC) 2019, Korea, December., 2019.
6. T.-H. Kim, **Eunho Cho**, Y.-J. Shin, and D.-H. Bae, *데이터 기반 교통 흐름 시스템 다이내믹스 환경 모델 생성 및 추론 기법*, Korea Software Congress (KSC) 2018, Korea, December., 2018.