UNIVERSITY OF CALIFORNIA,
IRVINE

Control System Design Automation Using Reinforcement Learning

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Hamid Mirzaei Buini

2018

DEDICATION

I dedicate this dissertation to my wonderful wife, Mona, who has been incredibly patient and supportive along my journey. Her love and encouragement have been true blessings for me. I also dedicate this dissertation to my lovely parents, who quietly and patiently waited for me to achieve my goals.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my PhD advisor, Professor Tony Givargis, for continuous support and encouragement. My sincere thanks also goes to Professor Peter Stone who provided me an opportunity to collaborate with their team as a visiting student.

I would like to express my gratitude to the member of my committees, Professor Eli Bozorgzadeh and Professor Ian Harris.

CURRICULUM VITAE

Hamid Mirzaei Buini

EDUCATION

Doctor of Philosophy in Computer Science	**2018**
University of California, Irvine	*Irvine, California*
Master of Science in Electrical Engineering	**2004**
Sharif University of Technology	*Tehran, Iran*
Bachelor of Science in Electrical Engineering	**2001**
Sharif University of Technology	*Tehran, Iran*

ABSTRACT OF THE DISSERTATION

Control System Design Automation Using Reinforcement Learning

By

Hamid Mirzaei Buini

Doctor of Philosophy in Computer Science

University of California, Irvine, 2018

Professor Tony Givargis, Chair

Conventional control theory has been used in many application domains with great success in the past decades. However, novel solutions are required to cope with the challenges arising from complex interaction of fast growing cyber and physical systems. Specifically, integration of classical control methods with Cyber-Physical System (CPS) design tools is a non-trivial task since those methods have been developed to be used by human experts and are not intended to be part of an automatic design tool. On the other hand, the control problems in emerging Cyber-Physical Systems, such as intelligent transportation and autonomous driving, cannot be addressed by conventional control methods due to the high level of uncertainty, complex dynamic model requirements and operational and safety constraints.

In this dissertation, a holistic CPS design approach is proposed in which the control algorithm is incorporated as a building block in the design tool. The proposed approach facilitates the inclusion of physical variability into the design process and reduces the parameter space to be explored. This has been done by adding constraints imposed by the control algorithm.

Furthermore, Reinforcement Learning (RL) as a replacement for convection control solutions are studied in the emerging domain of intelligent transportation systems. Specifically, dynamic tolling assignments and autonomous intersection management are tackled by the state-of-the-art RL methods, namely, Trust Region Policy Optimization and Finite-Difference Gradient Descent. Additionally, Q-learning is used to improve the performance of an embedded controller using a novel formulation in which cyber-system actions, such as changing control sampling time, is combined with the physical action set of the RL agent. Using the proposed approach, it is shown that the power consumption and computational overhead of the embedded control can be improved.

Finally, to address the current lack of available physical benchmarks, an open physical environment benchmarking framework is introduced. In the proposed framework, various components of a physical environment are captured in a unified repository to enable researchers to define and share standard benchmarks that can be used to evaluate different reinforcement algorithms. They can also share the realized environments via the cloud to enable other groups perform experiments on the actual physical environments instead of currently available simulation-based environments.

Chapter 1

Introduction

1.1 Motivation

Conventional digital control methods has been successfully employed in various industrial application domains during previous decades. To address the shortcomings of the conventional methods as a result of increasing complexity of interaction between cyber systems - including hardware/software systems and communication networks - and the physical environments, the new area of Cyber-Physical Systems (CPS) has been developed in the recent years. The problems addressed in CPS include but are not limited to hard real-time constraints, effective verification and efficient design process. However, in most cases, the focus has been on what follows after the control design step during the whole process. This makes the design space exploration a time-consuming task since in many cases changing the cyber parameters requires re-tuning of the control parameters and thus, the space of control parameters should be explored too To address this shortcoming, in the first part of this dissertation, a holistic PCS design approach is proposed in which the control parameter tuning is incorporated as a building block in the design tool. Additionally, using the proposed method, physical parameter space can be explored to find the best setting since the control parameters are updated by the design tool in response to physical parameter changes.

On the other hand, emerging CPSs such as autonomous vehicles, intelligent transportation systems and Artificial Intelligence(AI)-enabled robots, demand a replacement for classical control methods due to the physical environment complexity, high levels of uncertainty, continuous environment variations and challenging robustness and reliability requirements. Recently, Reinforcement Learning (RL) has shown to be a promising alternative for conventional control methods. In this dissertation, it has been shown that RL can be applied to some of the new CPS application domains. Furthermore, an open physical environment benchmark framework is introduced to evaluate RL (or AI in general) methods. Existence of this kind of physical benchmarks becomes more and more important as the new area of AI-based control systems advances.

1.2 Contributions

In this section, an overview of the contributions of this dissertation is provided. The first subsection includes the contributions made in the CPS design automation area and in the next subsection the contributions related to RL applications in CPS are discussed.

1.2.1 Incorporating Physical Variability into CPS Design

Physical variations in a CPS require recalculation of the control algorithm parameters. In most of the current CPS design solutions this recalculation is not done in the design tool but rather it is assumed that control algorithms should be revised manually by an expert after changes to the physical parameters or cyber parameters. To address this issue, in this dissertation, a design approach is proposed in which the control algorithm is made a building block of the design to reduce the complexity of design process and shrink the design space which should be explored. More specifically, the contributions of this part are summarized below:

- We ensure rigorous parametric description following the equation derivation approach of the physical subsystem.
- Settings of the control algorithm are determined as a function of the parameters of the physical system and the cyber-system for every iteration of the design space exploration, which eliminates time consuming search for suitable configurations.
- The DSE tool instantiates pre-compiled parameterized executable models of the CPS, which can be parallelized for the search of superior system configurations.

1.2.2 CPS Design Automation Using Reinforcement Learning

Adaptive Embedded Control

In this dissertation, it has been shown that RL can be leveraged to improve the dynamic performance of an embedded controller, while it can be used for the phyiscal control task. In particular, the sampling time of an embedded controller, which is traditionally assumed to be an invariant parameter, is tuned in real-time to enhance the efficiency of the CPS in terms of computational overhead and power-consumption.

Furthermore, a cloud-based co-simulation framework has been introduced to evaluate the performance of RL-based adaptive embedded control approach.

Autonomous Intersection Management

Currently, most of intersection management research (AIM) is focused on replacing traffic lights in the intersections of main roads experiencing high levels of traffics. In this dissertation, an RL method is proposed to solve the intersection management problem for local roads which are mainly controlled by stop signs. The following are the main contributions of this dissertation for the AIM problem:

- Definition and formulation of the AIM problem for local road settings where vehicles are coordinated by fine-grained acceleration commands.
- Employing the TRPO approach proposed in [79] to solve the formulated AIM problem.
- Incorporating collision avoidance constraint in the definition of RL environment as a safety mechanism.

Tuning Micro-tolling for Traffic Management

Micro-tolling has been shown to be possible solution that might be used in future to optimize traffic flow in large traffic networks of big cities. The main idea of Micro-tolling is to dynamically assign and change toll for all the main streets and highways during the day as a function of the traffic condition. A recent work [81] shows that Δ-*tolling*, i.e., tolling proportional to the difference between free-flow travel time and real travel time, yields significant improvement in terms of total travel time of all vehicles. However, the proportionality factor and the tolling time-constants is invariant in the proposed scheme. In this dissertation, the following contributions are made to enhance Δ-*tolling*:

- Different proportionality factor and time-constants are assigned for each individual link in the network to account for different properties of the links, such as width, fan-in/fan-out and number of lanes.
- A finite-difference RL solution is proposed to optimize the link-based parameters of the network.

- Different variations of local and global parameterization, e.g., a single global time-constant and local proportionality factor per link, are empirically studied on some example networks.

Physical Environment Benchmark

Currently, the benchmarking solutions for RL and AI methods are either simulation environments or expensive real-world physical environments available to a few organizations. One of the contributions of this dissertation is to propose a low-cost physical environment benchmarking framework to enable a larger group of researchers to develop AI methods that are applicable to real-world problems. Using the proposed framework, physical benchmark developers can share the hardware and software specifications of the physical environment through an open platform. They can also share their actual implementation via a web-based software on the cloud to be used by others for research and education purposes.

1.3 Dissertation Roadmap

The rest of the dissertation is organized as following: Chapter 2, includes a summary of prior research on related work. In Chapter 3, the proposed CPS design on physical variability is outlined. In Chapter 4 a short overview of RL and the methods that are used in the remaining chapters is given. Chapter 5 includes the methods to use RL in adaptive embedded control. In Chapter 6, the RL-based autonomous intersection management framework is explained. Tuning Micro-tolling via finite-difference RL is described in Chapter 7. Finally, the proposed open physical environment benchmark is introduced and evaluated with an example implementation in Chapter 8.

Chapter 2

Literature Survey

2.1 CPS Design Automation

Recent research has made tremendous advances in the development of frameworks for CPS that support the allocation and automatic evaluation of physical components in their CPS context [17, 71]. However, those frameworks operate on a high level of granularity by selecting preconfigured components but do not support parameterization of the physical components.

In this section we describe today's model-based design flows for cyber-physical systems. Specifically, we outline the state-of-the-art in automation of key steps of the design. The input to a CPS design flow is the CPS problem definition, which conventionally consists of the specification of the physical system, design objectives like the actual CPS task, and nonfunctional requirements like control quality metrics and power efficiency. In model-based design flows [44], generally, first the physical system is modeled, a control algorithm is designed, and a cyber system is architected. The system can finally be implemented after successful validation. The following steps outline the design flow in greater detail.

1) Physical Modeling: Physical modeling concerns the expression of the physical system consisting of plant and actuators in mathematical or logical terms. The physical model can be used for two main purposes: first to build a simulation model of the CPS, and second to design a control algorithm based on this model. The wide selection of applicable models for this purpose spans from conventional models like transfer function or state space models to complex probabilistic models for large-scale systems. Besides system identification methods, equation derivation has been applied to obtain the model parameter values of the physical system. In particular, the derivation of equations that govern the system using laws of physics has been applied in CPS development frameworks in practice [93]. Modeling of physical systems, even using those frameworks, still requires experienced domain experts and, to date, automation tools are unavailable or provide limited benefits. However, blueprints of physical models can be reused from a repository as applied in [17] and [71], but without parameterization.

2) Control Algorithm Design: With the aim to find a correct control algorithm for the CPS, the complex physical model needs to be simplified so that control design methods can be applied. Similar to the physical models, control algorithms can be selected from a repository to match the system properties and design objectives [17]. control algorithms have a range of parameters that influence the overall control quality. For example, PID controllers have three gain parameters. While for some specific use cases such as building control, an automatic control selection and parameterization was proposed [62], in general the selection and parameterization of the control algorithm still have to be performed manually, which limits the use for automated design.

3) Cyber System Design: The design of a suitable computation system that implements the designed control algorithm is a non-trivial task, since many cyber system design decision may affect the quality of the control algorithm. For example, the control algorithm has to be discretized, a sampling time has to be selected, and the delay and jitter of the processing elements have to be considered. To cope with the gap between control and cyber system, control engineers might specify the acceptable ranges for some parameters such as delay and jitter in a contract-based approach

[26]. Alternatively, the cyber system and the control algorithm can be co-designed with certain jitter and delay in mind [18, 38]. These approaches are valuable for a good cyber system/control algorithm co-design but neglect possible variability of the physical system. While verification and calculus methods [88] exist to validate the correctness of specific systems, typically, designers rely on the simulation to validate the cyber system design.

4) Simulation: Applying the models of the physical system, the cyber system, and the control algorithm, simulation-based analysis can be executed to evaluate the system under development. The main objective of the simulations, which can be performed by capable off-the-shelf tools such as Simulink and Modelica [34], is to evaluate and validate the performance of the CPS design. If all the requirements are met, the design can advance to the next step, namely implementation. Otherwise, reiteration of earlier steps in the design flow are required, which makes faults identified during simulation very expensive to fix, in particular if domain experts are required to manually refine the control algorithm or the physical system. Simulations are integral part of several approaches [17, 69, 71] to systematically analyze CPS design spaces. For instance Muhleis et al. [69] proposed a design space exploration solution based on co-simulation of control and cyber system, assessing design objectives such as cost or energy consumption for the whole CPS.

2.2 Adaptive Embedded Control

The impact of design decisions of the (Embedded Control System) ECS to the overall CPS system performance has been discussed in a range of works [16, 19, 71]. For instance [16] and [71] applied holistic cyber-physical design-space exploration to show the existence of optimal design points regarding control quality and power consumption. Specifically [16] showed that long sampling rates might increase the power consumption of the physical part of the system, while short sampling rates increase the average power consumption of the ECS. However, the works do not consider multi-mode system or time-variance yet.

Time-varying control has been discussed directly [19, 77] or in form of schedule planning [42, 85]. For instance, the optimal sampling time assignment in feedback controllers was studied in [19]. The result is that online sampling period assignment can deliver significantly better control performance than the state-of-the-art static period assignment. However, computing the optimal rate either requires complex online computations or large look-up tables which in turn reduces the power efficiency of the system. Sala [77] realized time-varying sampling periods and delayed actuation by time-varying observers and Kalman-filter based state-feedback controllers. While the approach demonstrates the feasibility of variable sampling periods, the implementation requires complex application-specific control knowledge to be feasible.

The works in [42] and [85] extended the idea to develop an optimal scheduling strategy for multiple control tasks on a shared computation platform that uses feedback from the physical system to optimize the control quality. While [42] still requires a complex application-specific online optimization strategy, the work in [85] added a dedicated feed-back scheduler to control the computation resources. On a higher system level, multi-rate control and time-varying sampling was investigated in [5] and [8]. However, the aim of the work in [5] is not to improve the efficiency of the ECS, but more generally to cope with variable sampling times as they occur in distributed and wireless sensor systems. The motivation in these works is to cope with timing uncertainty of sampled values, while our work aims to add time-variance in order to improve computation performance without degradation of the control quality of the CPS.

Non-uniform sampling time in digital-only control is studied in [49] and the proposed method is applied for tracking control of a linear actuator. Khan [49] applies non-uniform sampling time for a lower average sampling rate to achieve a lower average processing time. The proposed method is based on an adaptive change of digital controller coefficients as the sampling time changes, while a number or simplifying assumptions have been made such as linearity and time-invariance of the physical system dynamics.

In [4], different non-uniform sampling schemes, such as variable sampling period, non-synchronous sampling and multi-rate sampling are discussed for heterogeneous sensor systems. The solution, however, relies on linear dynamics in the physical system, which is not applicable to many cyber-physical systems.

In [50], authors have shown that optimal adaptive control algorithms can be developed using RL. The authors also have used practical examples that RL based controller can achieve desirable results in real-time. While our work does not aim to optimize the control quality as shown in [50], we apply the RL technique to control the properties of the ECS.

Event-triggered control and its variants also can realize non-uniform sampling in control systems [64]. However, in event-triggered control, a number of restricting assumptions have to be made such as explicit modeling of the physical system and existence of the Lyapunov control function. We have used an event-triggered method as the baseline solution in one of the case studies in Section 5.3.

2.3 Autonomous Intersection Management (AIM)

Advances in autonomous vehicles in recent years have revealed a portrait of a near future in which all vehicles will be driven by artificially intelligent agents. This emerging technology calls for an intelligent transportation system by redesigning the current transportation system which is intended to be used by human drivers. One of the interesting topics that arises in intelligent transportation systems is AIM. Dresner et al. have proposed a multi-agent AIM system in which vehicles communicate with intersection management agents to reserve a dedicated spatio-temporal trajectory at the intersection [28].

In [66], authors have proposed a self-organizing control framework in which a cooperative multi-agent control scheme is employed in addition to each vehicle's autonomy. The authors have proposed a priority-level system to determine the right-of-way through intersections based on vehicles' characteristics or intersection constraints.

Zohdy et al. presented an approach in which the Cooperative Adaptive Cruise Control (CACC) systems are leveraged to minimize delays and prevent clashes [94]. In this approach, the intersection controller communicates with the vehicles to recommend the optimal speed profile based on the vehicle's characteristics, motion data, weather conditions and intersection properties. Additionally, an optimization problem is solved to minimize the total difference of actual arrival times at the intersection and the optimum times subject to conflict-free temporal constraints.

A decentralized optimal control formulation is proposed in [63] in which the acceleration/deceleration of the vehicles are minimized subject to collision avoidance constraints.

Makarem et al. introduced the notion of fluent coordination where smoother trajectories of the vehicles are achieved through a navigation function to coordinate the autonomous vehicles along predefined paths with expected arrival time at intersections to avoid collisions.

In all the aforementioned works, the AIM problem is formulated for only one intersection and no global minimum travel time objective is considered directly. Hausknecht et al. extended the approach proposed in [28] to multi-intersection settings via dynamic traffic assignment and dynamic lane reversal [41]. Their problem formulation is based on intersection arbitration which is well suited to main roads with a heavy load of traffic.

In Chapter 6, for the first time, we introduce fine-grained acceleration control for AIM. In contrast to previous works, Our proposed AIM scheme is applicable to local road intersections. We also propose an RL-based solution using Trust Region Policy Optimization to tackle the defined AIM problem.

2.4 Optimal Traffic Control

The approach suggested in Chapter 7 for solving the micro-tolling assignment problem builds on two previously presented algorithms: Δ-tolling, and Finite Difference policy Gradient Reinforcement Learning (RL). In the next subsections, the previous works related to these methods will be described.

2.4.1 Delta-tolling

It is well known that charging each agent an amount equivalent to the cost it inflicts on all other agents, also known as *marginal-cost tolling*, results in optimal social welfare [74].

Applying a marginal-cost tolling scheme, when differentiable latency functions are not assumed, requires knowing in advance the marginal delay that each agent will impose on all others. This, in turn, requires knowledge of future demand and roadway capacity conditions, as well as counterfactual knowledge of the network states without each driver.

Δ-*tolling* [81, 82] was recently suggested as a model-free scheme for evaluating marginal cost tolling. It requires observing only the latency (travel time) on each link and makes no assumption on the underlying traffic model. Δ-*tolling* involves charging a toll on each link proportional to its delay (the difference between observed and free-flow travel times). Δ-*tolling* requires tuning of only two parameters: a proportionality constant (β), and a smoothing parameter (R) used to damp transient spikes in toll values.

Algorithm 1 describes the toll value update process of Δ-*tolling*. For each link, Δ-*tolling* first computes the difference (Δ) between its current latency (l_e^i) and its free flow travel time (denoted by T_e). We use i to denote the current time step. Next, the toll for link e at the next time step (τ_e^{i+1}) is updated to be a weighted average of Δ times beta and the current toll value. The weight

Algorithm 1: Updating tolls according to Δ-*tolling*.

```
1 while true do
2     for each link $e \in E$ do
3         $\Delta \leftarrow l_e^i - T_e$
4         $\tau_e^{i+1} \leftarrow R(\beta\Delta) + (1-R)\tau_e^i$
5     $i \leftarrow i + 1$
```

assigned to each of the two components is governed by the R parameter ($0 < R \leq 1$).

The R parameter determines the rate in which toll values react to observed traffic conditions. When $R = 1$ the network's tolls respond immediately to changes in traffic on the one hand but leave the system susceptible to oscillation and spikes on the other hand. By contrast, as $R \to 0$ the tolls are stable, but are also unresponsive to changes in traffic conditions.

Sharon et al. [81, 82] showed that the performance of Δ-*tolling* is sensitive to the values of both the R and β parameters. Their empirical study suggests that values of $\beta = 4$ and $R = 10^{-4}$ result in the best performance. However, they do not present a procedure for optimizing these parameters and relay on brute force search for finding the optimal values through trial and error.

2.4.2 Policy gradient RL

Policy gradient RL is a general purpose optimization method that can be used to learn a parameterized policy based on online experimental data. While there are several different methods for estimating the gradient of the policy performance with respect to the parameters [73], one of the most straightforward, and the one we use in this dissertation, is *Finite Difference Policy Gradient RL* (FD-PGRL) [52] which is based on finite differences. In this subsection we review the methods and formulations presented in [52].

FD-PGRL is presented in Algorithm 2. Under this framework, the policy is parameterized using the parameter vector $\pi = [\theta_1, \dots, \theta_N]^\intercal$. The algorithm starts with the initial parameters $\pi^0 = [\theta_1^0, \dots, \theta_N^0]^\intercal$ (line 1). At each step k, the policy gradient is estimated by running a set of randomly generated policies $\Pi^k = \{\pi_1^k, ..., \pi_M^k\}$ (lines 5- 7) where each policy is defined as:

$$\pi_m^k = [\theta_1^{k-1} + \delta_{1,m}^k, \dots, \theta_N^{k-1} + \delta_{N,m}^k]^\intercal, \tag{2.1}$$

where $\delta_{n,m}^k \in \{-\epsilon_n, 0, \epsilon_n\}$. The generated policies in (2.1) are obtained by randomly changing each parameter from the previous policy by a small ϵ_n, relative to θ_n. The cost of each newly created policy, π_m^k, is observed and denoted by c_m^k (lines 8- 9).

To estimate the policy gradient, the policy set in (2.1) is partitioned to three subsets (lines 11- 14) for each dimension depending on whether the change in the policy in that dimension is negative, positive or zero, that is the three subsets are:

$$\pi_m^k \in \begin{cases} \Pi_{-\epsilon,n}^k = \{\pi_m^k : \delta_{n,m}^k = -\epsilon\} \\ \Pi_{0,n}^k = \{\pi_m^k : \delta_{n,m}^k = 0\} \\ \Pi_{+\epsilon,n}^k = \{\pi_m^k : \delta_{n,m}^k = \epsilon\}. \end{cases} \tag{2.2}$$

The average costs of above policy subsets are denoted by $\overline{c}_{-\epsilon,n}^k$, $\overline{c}_{0,n}^k$ and $\overline{c}_{+\epsilon,n}^k$ (lines 15- 17). The adjustment vector $A^k = [a_1^k, \dots, a_N^k]^\intercal$ can be constructed by the following equation for each dimension (lines 18- 21):

$$a_n^k = \begin{cases} 0, & \text{if} \quad \overline{c}_{-\epsilon,n}^k < \overline{c}_{0,n}^k \text{and} \quad \overline{c}_{+\epsilon,n}^k < \overline{c}_{0,n}^k \\ \overline{c}_{+\epsilon,n}^k - \overline{c}_{-\epsilon,n}^k & \text{otherwise} \end{cases} \tag{2.3}$$

The adjustment vector A^k is normalized and multiplied by a constant step size η to update the parameter vector at the end of each step k (lines 22- 23).

Unlike other policy gradient methods that rely on within-episode reward signals to search for an optimal policy, or those in which the agent must learn the policy with no prior knowledge of a reasonably-performing starting policy (for example [31] and [56]), in the method employed in Chapter 7, the policy is parameterized with a finite set of parameters and the overall system performance at each episode is optimized using an empirical estimate of the policy gradient based on finite differences. This approach is well-suited for the traffic optimization problem for two reasons. First, the agent can leverage an existing policy with reasonable system performance. Second, the agent is required to proceed towards the optimal policy only by slight changes of the policy parameters in contrast to approaches in which randomized exploration policies can be executed more freely. Our empirical study suggests that considering such slight changes results in a total cost that is within an acceptable bound. Furthermore, using other RL methods to learn actual tolls in real-time instead of Δ-*tolling* parameters requires modeling traffic as Markov Decision Process which is a challenging task (see [10]).

2.5 RL Physical Benchmarks

In this section, we first review existing literature about solving real-world problems using AI algorithms. Next, we review recent simulation-based AI benchmarks that are widely used in academia. Finally, we review the related research projects to provide real-world benchmarks in robotic applications.

Using RL as a replacement for conventional control theory is an emerging trend in Cyber-Physical systems. In [70] an RL algorithm is proposed to autonomously navigate a humanoid Nao robot into a docking station used for recharging. An RL model is proposed in [57] to learn hand-eye coordination for grasping objects in an environment of robotic manipulators. In [65], RL methods have been applied on an actual cart-pole system to balance the pole. Researchers are exploring AI algorithms as a way to simplify and speed up the programming of industrial robots in factories.

Fanuc [48], the world's largest maker of industrial robots, has used RL methods to train robots to precisely pick up a box and put it in a container. In the automotive industry, authors in [72] have proposed an RL-based approach to control robot morphology (flippers) to move over rough terrains that exist in Urban Search and Rescue missions.

Access to these physical environments (hardwares/robots) is not feasible for a lot of research groups. This hinders partnerships and cooperation between academia and industry. In Chapter 7, for the first time, we propose the idea of providing low-cost and easy-to-construct physical environments that allow researchers and students to implement, evaluate and compare their AI algorithms on standardized benchmarks.

In a dynamic AI problem, the state of the environment depends on the actions that are chosen by the agent. This makes it almost impossible to store the environment as a fixed dataset similar to the supervised machine learning paradigm. Therefore, to facilitate reproducible research and accelerate the pace of education, researchers in this community are trying to design a standard programming interface for reinforcement-learning experiments.

One of the earliest efforts to design a standard tool is RL-Glue [87] which has been used for RL courses at several universities to create experiments for scientific papers. A more recent effort, RLPy [36], is a software framework written in python that has focused on value-function-based methods with linear function approximation using discrete actions. ALE [12] is another software framework designed to make it easy to develop agents that play different genres of Atari 2600 games.

OpenAI Gym [13] is the most recent and comprehensive toolkit for developing AI algorithms. It provides a diverse suite of environments that range from classic control to 2D and 3D robots. It is designed to let the users evaluate the proposed AI algorithms with little background in AI. Researchers can compare the performance of their proposed algorithm with other approaches' scores reported on the scoreboard. These solutions are very effective in advancement of research and education within simulated environments because it is usually expensive and more challenging to implement AI algorithms in real-world scenarios.

Most similar to our work is [84] that has proposed an open hardware design for academic and research robots. They have leveraged 3D printing technology to allow users to create all required components except electronics parts. All basic code and libraries have been released under the GNU General Public License. Authors in [22] have made their research on aquatic swarm robots reproducible by providing the 3D printing models, CNC milling files and the developed software on Raspberry Pi. In Chapter 8, we propose a framework that can be used to produce an arbitrary number of physical environments, not limited to robots. Contrary to the mentioned works where a specific physical environment is introduced, a unified benchmark framework is proposed in Chapter 8 to integrate a variety of physical environments. In other words, research groups can contribute by sharing their physical environment blueprints using the proposed framework.

Algorithm 2: Finite Difference Policy Gradient RL

```
1  $\pi^0 \leftarrow [\theta_1^0, \ldots, \theta_N^0]^\intercal$;
2  $k \leftarrow 0$;
3  while improving do
4      $k \leftarrow k + 1$;
5      generate $\Pi^k = \{\pi_1^k, ..., \pi_M^k\}$,
6          $\pi_m^k = [\theta_1^{k-1} + \delta_{1,m}^k, \ldots, \theta_N^{k-1} + \delta_{N,m}^k]^\intercal$,
7          $\delta_{n,m}^k \sim Uniform\{-\epsilon_n, 0, \epsilon_n\}$;
8      for each $m \in \{1, \ldots, M\}$ do
9          $c_m^k \leftarrow run(\pi_m^k)$;
10     for each $n \in \{1, \ldots, N\}$ do
11         partition $\Pi^k$ to
12             $\Pi_{-\epsilon,n}^k = \{\pi_m^k : \delta_{n,m}^k = -\epsilon\}$,
13             $\Pi_{0,n}^k = \{\pi_m^k : \delta_{n,m}^k = 0\}$,
14             $\Pi_{+\epsilon,n}^k = \{\pi_m^k : \delta_{n,m}^k = \epsilon\}$;
15         $\overline{c}_{-\epsilon,n}^k \leftarrow average(c_m^k : \pi_m^k \in \Pi_{-\epsilon,n}^k)$;
16         $\overline{c}_{0,n}^k \leftarrow average(c_m^k : \pi_m^k \in \Pi_{0,n}^k)$;
17         $\overline{c}_{+\epsilon,n}^k \leftarrow average(c_m^k : \pi_m^k \in \Pi_{+\epsilon,n}^k)$;
18         if $\overline{c}_{-\epsilon,n}^k < \overline{c}_{0,n}^k \& \quad \overline{c}_{+\epsilon,n}^k < \overline{c}_{0,n}^k$ then
19             $a_n^k \leftarrow 0$;
20         else
21             $a_n^k \leftarrow \overline{c}_{+\epsilon,n}^k - \overline{c}_{-\epsilon,n}^k$;
22     $\pi^k \leftarrow \pi^{k-1} - \eta \frac{A^k}{|A^k|}$,
23         $A^k = [a_1^k, \ldots, a_N^k]^\intercal$;
```

Chapter 3

Physical Model Variablity in CPS Design [1]

3.1 Introduction

Generally, a cyber-physical-system (CPS) is one that combines computational and physical entities in a unified design effort. The design of CPSs needs good understanding of both subsystems, as small changes in the physical subsystem (PS) or the cyber subsystem (CS) may have significant consequences with respect to the overall system performance. For example, it is well known that the weight and size of physical objects like pendulums [92] directly influence the performance requirements for the CS but also that scheduling decisions on the CS may affect the timing jitter of an otherwise correct control application so that the control process fails [6]. Therefore, a good CPS design methodology must carefully model and account for physical attributes of a system.

Traditional design tools are well suited in addressing design constraints and optimization objectives of the CS, but often fall short adequately to consider physical and control aspects of a CPS. Parameters of physical components include, for instance, setting the diameter of a pipe, the strength of

[1]This chapter is mainly reprinted from: Buini, Hamid Mirzaei, Steffen Peter, and Tony Givargis. "Including variability of physical models into the design automation of cyber-physical systems." In Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE, pp. 1-6. IEEE, 2015. IEEE, Copyright (2015), with permission from IEEE.

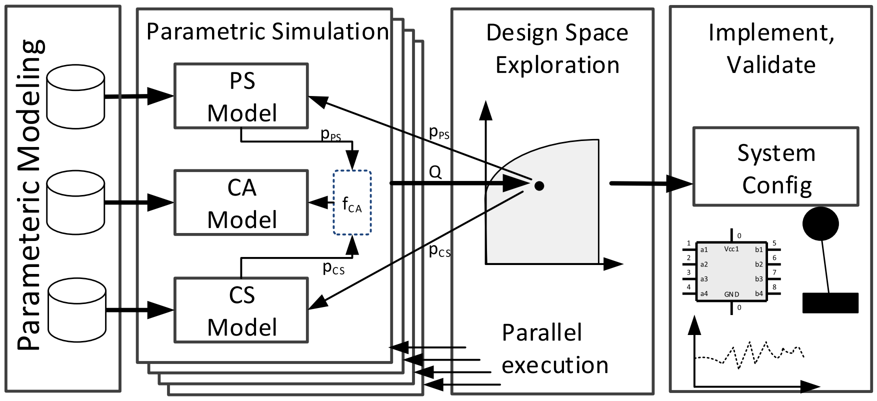


Figure 3.1: Proposed CPS design framework: Parametric models are instantiated in a parameterizable simulation. The DSE tool invokes simulations parameterized with properties of the PS and the CS. Aim is to identify superior design points which can be validated and implemented.

a spring, or the size of mechanical parts. Another important aspect that has not been addressed in current CPS design automation frameworks is the consideration of the impact of the changes in the control subsystem. The control algorithm (CA) specifies how measurements from the PS are processed and how actuation commands are generated so that the CPS can achieve its objectives. As such, the CA is the logical bridging element between the CS and the PS, and needs to be designed and adapted when subsystems, including physical aspects, undergo changes.

The goal of this work is the development of a design space exploration (DSE) framework that considers the variabilities of the PS and the required adaptations of the CA. Our proposed design flow is shown in Figure 3.1. The basic idea is to derive parameterizable models of the PS, the CA and the CS. The parameterizable models then can be instantiated by the DSE tool that, for each design point, invokes an executable simulation model to evaluate the quality of the CPS design. After an evaluation of the designs, a system configuration is proposed for implementation, containing parameters for the physical and the cyber subsystems. Our design flow is facilitated by the

The encapsulated control knowledge and the efficient simulation enable CPS design analysis, available for system engineers even without strong control background. We show the steps, necessary to prepare the models, and the application of the models in a design space exploration including a non-trivial trade-off between control quality and energy consumption. We evaluate our approach for a rotary inverted pendulum system, for which we explored the design space that included a selectable length of the pendulum and a flexible sampling time in the CS. The results, which we validated in a reallife setup, show that our approach can automatically identify superior cyber-physical configurations which would have been ignored using existing design space exploration techniques.

3.2 Parametric CPS Design Flow

In this section we introduce our parametric design flow, which is based on the standard MBD flow discussed in the previous section. As shown in Figure 3.1, our flow relies on parameterizable models for the PS, the CA, and the CS, which have to be crafted first, before they can be instantiated in simulation and for the DSE.

1) Parametric Physical Modeling: To craft the parameterizable model of the PS we propose the Equation-Based Modeling (EBM) [15] method. In contrast to system identification or learning techniques, EBM is not focused on finding a fixed set of parameters for a single instance of PS, but facilitates the derivation of the equations symbolically. The result is a parameterized model that describes the physical system behavior as a set of differential equations, E_{ps}, and their parameters P_{Ps}, so that the model can be expressed as:

$$M_{ps} = (P_{ps}, E_{ps}) \tag{3.1}$$

The representation of the PS as M_{ps} is rewarding for two reasons: First, it allows composition of smaller physical subsystem to larger PSs [93], and second, M_{ps} is reusable and can be directly instantiated during the simulation step. However, it should be noted that no general guidelines are available how to craft M_{ps}, and the result becomes complex already for small systems like the example we discuss in Section IV.

2) Parametric Control Algorithm Design: As mentioned in Section II, generally CAs are already designed in parametric form, so that the CA models M_{ca} can be expressed as:

$$M_{ca} = (P_{ca}, E_{ca}) \tag{3.2}$$

where E_{ca} is the set of equations describing the CA, and P_{ca} is the set of CA parameters. Like the physical models, M_{ca} can be directly applied as parametrizable model in the simulations. A major issue is that the parameters P_{ca} depend on the PS as well as the CS and therefore must reflect changes in those subsystems. A complex search for fitting parameters is not feasible for complex DSEs, because the search is required for each design point. Therefore, we propose to follow a derivation technique similar to the EBM approach discussed for the PS. The idea is to express P_{ca} as a mapping from properties of the models from PS and CS, and the design objective parameters, P_{obj}. This mapping $\mathbf{f}_{ca}$ is expressed as:

$$P_{ca} = \mathbf{f}_{ca}(P_{ps}, P_{cs}, P_{obj}). \tag{3.3}$$

Since $\mathbf{f}_{ca}$ computes the actual controller parameters, $\mathbf{f}_{ca}$ does not need to be symbolic, or in a closed form, but can also be represented numerically, as demonstrated in the example in Section II. Crafting of $\mathbf{f}_{ca}$ requires the application of control theory methods for each type of controllers manually. The result is a transformation of the general model (3.2) to

$$M_{ca} = (\mathbf{f}_{ca}(P_{ps}, P_{cs}, P_{obj}), E_{ca}), \tag{3.4}$$

which can be instantiated in simulations, with all parameters directly based on existing knowledge in the system.

3) Simulation and Design Space Exploration: The simulation tool we introduce next, instantiates

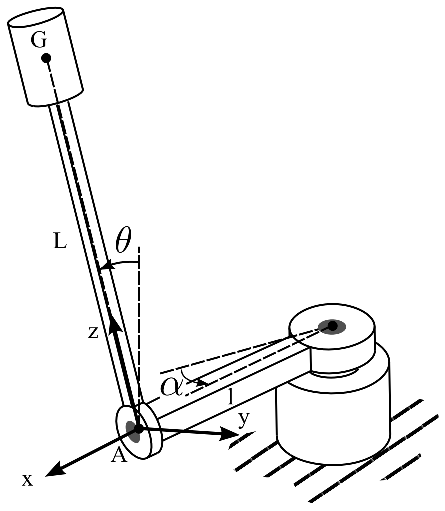


Figure 3.2: One degree of freedom inverted pendulum

the building blocks that were described in the previous steps. Since the idea is to run the simulation for each parameter setting in the design space, we are interested in a short simulation time. A standard approach is to obtain the performance values of interest (Q) by instantiating the parametrized models in a interpreted simulation environment such as Simulink. This approach is technically possible, but requires significant amount of time for each simulation. To accelerate the process we considered compiled simulation models, which is supported by Simulink and Modelica. However, compiling parameterized models still required to recompile the simulation model for each setting. To avoid unnecessary recompilations, we instead expose the parameters from the parametrizable models of the PS (3.1) and the CA (3.4), and only compile the unparametrized models, so that the simulation is expressed as:

$$Q = \mathrm{SIM}_{M_{ps},M_{cs},M_{ca}}(P_{ps}, P_{cs}, P_{ca}) \tag{3.5}$$

Hence, the simulation model is compiled only once for each model allocation, but recompiling is not required for changing model parameters in this tool. In Modelica, for instance, we only need to set the parameter values in an XML file and execute the same binary file for all the parameter settings. The result is a native executable simulation code which can be invoked by design exploration tools which only have to provide the parameters of the PS, the CS and the design objectives.

```
Algorithm 3: DSE for parametrized physical models
   Input: Simulate()                          ▷ The simulation code called as a procedure
       $\mathbf{f}_{ca}$                              ▷ The parametric CA design function
       $D \subset \mathbb{R}^m$                                           ▷ Design space
   Output: $J : D \mapsto \mathbb{R}^n$                   ▷ Evaluated Variables-of-interest
1  Procedure EVALVOI $(P_{ps}, P_{cs}, P_{obj})$
2      $P_{ca} \leftarrow \mathbf{f}_{ca}(P_{ps}, P_{cs}, P_{obj})$;
3      Q ← Simulate($P_{ps}$, $P_{cs}$, $P_{ca}$);
4      Put $Q$ in the queue $U$;
5  forall the $(P_{ps}, P_{cs}, P_{obj}) \in D$ {in parrallel} do
6      EvalVOI($P_{ps}, P_{cs}, P_{obj}$);
7  Sort $U$ to find the mapping $J$;
```

To study the design space, the pre-compiled simulations can be invoked in generic design space exploration (DSE) algorithms. As one example, Algorithm 3 demonstrates the feasibility of DSE that supports physical parameters, by systematical evaluation of all physical and cyber configurations in the search space (line 5, 6). Subfunction EVALVOI computes P_{ca}, invokes the simulation, and stores Q. The result is a set of evaluated variables-of-interest for each design point in the design space. From this space, the system configuration is selected evaluating the stored results. A benefit of the proposed algorithm is that procedure EVALVOI is "embarrassingly parallel", so that it can easily utilize parallel multi- and many-core platforms to accelerate the search process. Notably, Algorithm 3 does not utilize design space pruning techniques like they are required for larger design spaces. However, the proposed algorithm can be applied for smaller CPSs, such as the rotary inverted pendulum case study presented next.

3.3 Case Study: Rotary Inverted Pendulum

In this section we use the rotary inverted pendulum example [92] as a case study to demonstrate the steps of framework described in the previous sections. The goal of the system, shown in Figure 3.2, is to produce appropriate actuator angle α to keep the pendulum in upright position, i.e, $\alpha \approx 0$. In following subsections we discuss of the design flow including the crafting of the models (A-C) and the execution of the DSE, including a trade-off analysis and practical validation (D-E).

3.3.1 Parametric Physical Modeling

As explained in previous section, a parametric model of the PS is fundamental for the subsequent design steps. The block diagram of the complete CPS is shown in Figure 3.3. The controller should output the motor voltage in order to balance the pendulum. At the same time, the controller must track the reference command for the angle α. Here, we assume that the main output of the system is α and we want to move the weight to the commanded angle α_r. For this purpose, we assume that all the state variables, $\mathbf{x}$, are measured and provided to the controller.

The PS is shown using a dashed box in Figure 3.3. The PS consists of the actuator and the pendulum dynamics subsystems. First, the model of each subsystem is obtained and next they are combined to build the PS model. In the following paragraphs the EBM of these two subsystems is explained. We use the following symbols throughout this chapter for the physical parameters and variables:

α_m	Motor angle (before gearbox)
B_m	Motor viscous friction coefficient
i_a	Motor winding current
J	Motor, Gearbox and actuator arm moment of inertia
K_b	Motor back-emf constant

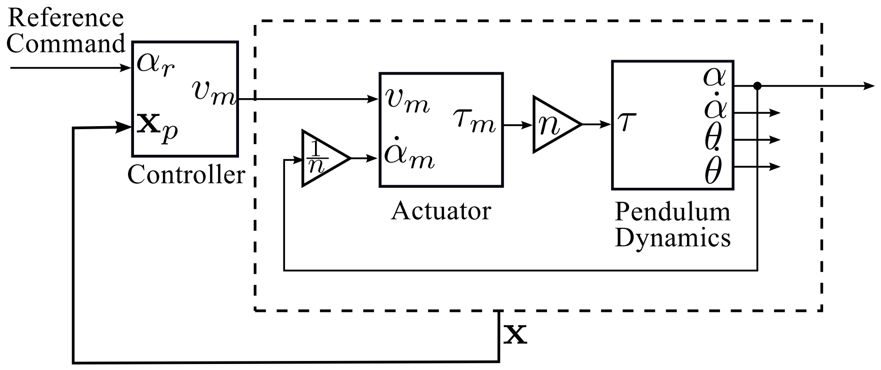


Figure 3.3: Block diagram of the inverted pendulum example: A closed loop control system consisting of Controller, Actuator, and Pendulum Dynamics.

K_t	Motor torque constant
l	Actuator arm length
L	Pendulum length
L_a	Motor winding self inductance
m	Mass of the pendulum weight
μ_m	Actuator viscous friction coefficient
μ_p	Pendulum joint friction coefficient
n	Gearbox ratio
R_a	Motor winding resistance
τ	Torque applied by the actuator
τ_m	Torque applied by the motor (before gearbox)
v_m	Voltage applied to the motor terminal

a) Pendulum Dynamic Equations: Dynamics modeling is required because we want to know how the pendulum angle changes when a torque is applied by the actuator. Assuming the mass of pendulum rod is negligible, we write the Newton-Euler equations for the rigid body consisting of pendulum weight and rod in the $\{xyz\}$ coordinate frame as follows [37]:

$$\sum \mathbf{F} = m\mathbf{a}_G \tag{3.6}$$

$$\sum \mathbf{M}_A = \mathbf{r}_{G/A} \times m\mathbf{a}_G + \frac{d}{dt}\mathbf{H}_A, \tag{3.7}$$

Expanding these equations we obtain following equation that relates the angular variables and their time derivatives:

$$-\mu_p\dot{\theta} + mgL\sin\theta = mL^2\ddot{\theta} - mL^2\dot{\alpha}^2\sin\theta\cos\theta - mLl\ddot{\alpha}\cos\theta \tag{3.8}$$

To involve the torque applied by the actuator in a new equation, torque equation around the motor axis can be used. The torque equation results in:

$$J\ddot{\alpha} = \tau - lF_y\cos\theta + lF_x\sin\theta - \mu_m\dot{\alpha}. \tag{3.9}$$

where $F_x = -l\dot{\alpha}^2 + L\sin\theta\ddot{\alpha}$ and $F_y = l\cos\theta\ddot{\alpha} - L\ddot{\theta} + L\sin\theta\cos\theta\dot{\alpha}^2$. If we solve equations (3.8) and (3.9) for $\ddot{\alpha}$ and $\ddot{\theta}$ we will get:

$$\ddot{\alpha} = f_\alpha(\alpha, \dot{\alpha}, \theta, \dot{\theta}, \tau) \tag{3.10}$$

$$\ddot{\theta} = f_\theta(\alpha, \dot{\alpha}, \theta, \dot{\theta}, \tau) \tag{3.11}$$

These equations are used to define a nonlinear state space (SS) model for the pendulum as with the state variables $\mathbf{x}_p = \begin{pmatrix} \alpha & \dot{\alpha} & \theta & \dot{\theta} \end{pmatrix}^T$ and input variable $u_p = \tau$ as

$$\dot{\mathbf{x}}_\mathbf{p} = \mathbf{f}(\mathbf{x}_p, u_p) \tag{3.12}$$

This SS model is used inside the bigger simulation model of the whole system in the next step of our design flow.

The next step concerns the linearization of (3.12) with the aim to design the LQR Control method. For this purpose, we find the Jacobean of vector function $\mathbf{f}$ with respect to $\mathbf{x}_p$ and u_p and evaluate it in a reference solution of the (3.12), $(\mathbf{x_{p0}}, u_{p0})$. One reference solution is $\mathbf{x_{p0}}(t) = \mathbf{0}$, $u_{p0}(t) = 0$, which means the system is at rest with all the variables set to zero. So, the linearized system can be written as:

$$\dot{\mathbf{x}}_p = A_p \mathbf{x}_p + B_p u_p, \tag{3.13}$$

$$A_p = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}_p}\right|_{\mathbf{x}_p = \mathbf{x_{p0}} = \mathbf{0}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-\mu_m}{J} & \frac{glm}{J} & -\frac{l\mu_p}{JL} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{l\mu_m}{JL} & \frac{g(ml^2+J)}{JL} & -\frac{\mu_p(ml^2+J)}{JL^2 m} \end{pmatrix} \tag{3.14}$$

$$B_p = \left.\frac{\partial \mathbf{f}}{\partial u_p}\right|_{u_p = u_{p0} = 0} = \begin{pmatrix} 0 & \frac{1}{J} & 0 & \frac{l}{JL} \end{pmatrix}^T \tag{3.15}$$

These matrices are the first part of the linearized PS model of our example. The model will be concluded with the actuator SS matrices which are explained next.

b) Actuator Modeling: The actuator in our case study is assumed to be a geared DC motor. The linear equation based model is:

$$L_a \dot{i}_a + R_a i_a = v_m - K_b \dot{\alpha}_m \tag{3.16}$$

$$\tau_m = K_t i_a - B_m \dot{\alpha}_m \tag{3.17}$$

Defining the actuator state and input as $x_a = i_a$ and $\mathbf{u}_a = \begin{pmatrix} v_m & \dot{\alpha}_m \end{pmatrix}^T$, the resulting SS model of the actuator is:

$$\dot{x}_a = A_a x_a + B_a \mathbf{u}_a = -\frac{R_a}{L_a} x_a + \begin{pmatrix} \frac{1}{L} & -\frac{K_a}{L_a} \end{pmatrix} \mathbf{u}_a \tag{3.18}$$

$$y = C_a x_a + D_a \mathbf{u}_a = K_t x_a + \begin{pmatrix} 0 & -B_m \end{pmatrix} \mathbf{u}_a \tag{3.19}$$

Also, assuming the gearbox attached to the DC motor is ideal we will have $\tau = \tau_m n$ and $\alpha = \alpha_m / n$.

3.3.2 Parametric Control Algorithm Design

In LQR, the design objective is expressed by a quadratic cost function. In our case study this cost function is:

$$J = \int_0^\infty \left(\mathbf{x}^T Q \mathbf{x} + R v_m^2 \right) dt = \int_0^\infty \left(q ||\alpha - \alpha_r||^2 + v_m^2 \right) dt \tag{3.20}$$

where $\mathbf{x} = (\alpha - \alpha_r \quad \dot{\alpha} \quad \theta \quad \dot{\theta} \quad i_a)^T$ and we have chosen $Q = q\mathbf{v}\mathbf{v}^\mathbf{T}$ where $\mathbf{v} = (1 \quad 0 \quad 0 \quad 0 \quad 0)^T$ and $R = 1$.

The above cost function tries to balance the energy consumption and control performance. The first component in (3.20) , $\int_0^\infty ||\alpha - \alpha_r||^2 dt$, defines control performance metric as the mean-squared error (MSE) of angular command tracking. The parameter q in (3.20) is the relative weight of control performance to power consumption which is described by the second component $\int_0^\infty v_m^2 dt$.

In LQR, the control law is the state feedback which can be described using the following equation:

$$v_m = -K\mathbf{x}, \tag{3.21}$$

The State vector $\mathbf{x}$ is the combination of the pendulum dynamics and actuator SS model state vectors, with one modification, to subtract the reference command α_r from α.

Algorithm 4: The Control Algorithm

```
Input: $K, \alpha_r$                          ▷ LQR gain and angular command
1 forall the sampling times do
2     Measure $\alpha, \dot{\alpha}, \theta, \dot{\theta}$ and $i_a$;
3     $v_m \leftarrow -K(\alpha - \alpha_r \quad \dot{\alpha} \quad \theta \quad \dot{\theta} quad i_a)^T$;
4     Apply voltage $v_m$ to motor;
```

By solving the algebraic Ricatti equation [51], the optimal gain K in (3.21) is calculated. This gain optimizes the cost function (3.20) guaranteeing that the closed loop system will be stable. Solving Ricatti equation to obtain optimal gain, K, corresponds to (3.3) in the proposed design flow. Therefore, the concrete form of (3.3) is:

$$K = \mathbf{f}_{lqr}(L, q) \tag{3.22}$$

For the control we only have to choose a sampling time, h, compute and apply the control signal in successive sampling periods. The CA is shown in Algorithm 4.

3.3.3 Simulation

We built the executable specification model of Figure 3.3 in Modelica, instantiating the pendulum equation (3.12), actuator equations (3.18) and (3.19) and the discrete-time state feedback (3.21). By compiling the Modelica model we have the simulation executable code. The inputs to this code are pendulum length L, sampling time h and controller gain K. The output is the objective variable which is MSE of tracking error $||\alpha - \alpha_r||$. At the end of this step, the general form of simulation procedure explained in Section 3.2 is:

$$MSE = \text{SIM}_{\text{Inv Pendulum}}(L, h, K), \tag{3.23}$$

while K is the result of (3.21) and can be computed externally using the control Python package.

3.3.4 Design Space Exploration

1) Setup and Execution: The DSE in our case is an implementation of Algorithm 3 in Python, instantiating the simulation executable code of (3.23). The design objective parameter q is fixed to 10 $(1/rad^2)$ in this case study. We studied pendulum lengths from 0.01 (cm) to 60 (cm) and sampling times from 0.01 (s) to 0.14 (s). The parallelism of the DSE and the pre-compiled simulation allowed us to finish the exploration within is 87(sec) on a 48-core Intel Xeon @ 2.7GHz platform. The naive approach without parallelization and using Simulink in normal mode results in 2360 (sec). Using existing DSE flows with no parametric CA design, the running time would be approximately 86 days if we want to try only 5 values for each element of gain parameter K.

2) Control Quality: Figure 3.4 shows the stability region of the DSE. This plot is generated by comparing the computed MSE with a fixed threshold of 0.035 (rad^2). A system with MSE below the threshold is considered stable. Figure 3.4 also compares our parametric control approach to static reusable controllers. The stability regions for fixed-controllers designed for 6 (cm) and 34 (cm) pendulums are shown on top of the region for our approach. Evidently, the static controllers only cover small range of the design space, while our approach provides good design points for all physical settings.

3) Power Consumption: The second important metric is power consumption. The total power of the system is the sum of the power for the PS and the CS: $P_{CPS} = P_{PS} + P_{CS}$. P_{PS} is the required power of the actuator ($P_{PS} = v_m i_m$), which is part of the actuator model (3.16). P_{CS} can be computed from the processing time (t_{prc}) and the average power consumption of the microcontroller for sleep (P_{slp}) and run (P_{run}):

$$\bar{P}_{CS} = \frac{t_{prc}}{T} P_{run} + \left(1 - \frac{t_{prc}}{T}\right) P_{slp} \tag{3.24}$$

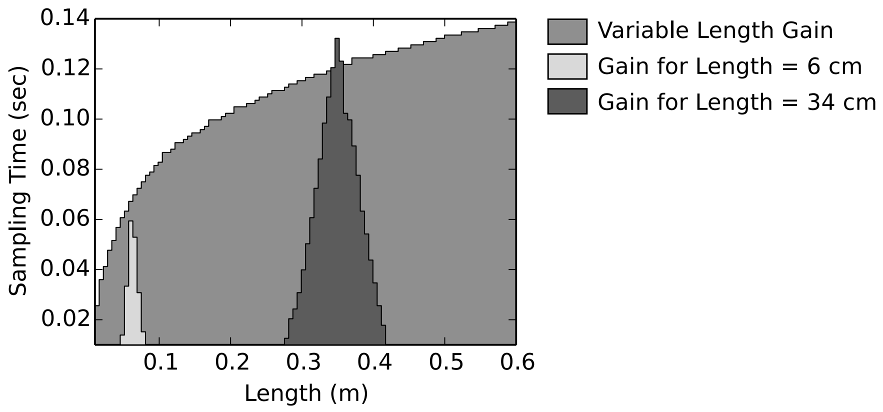


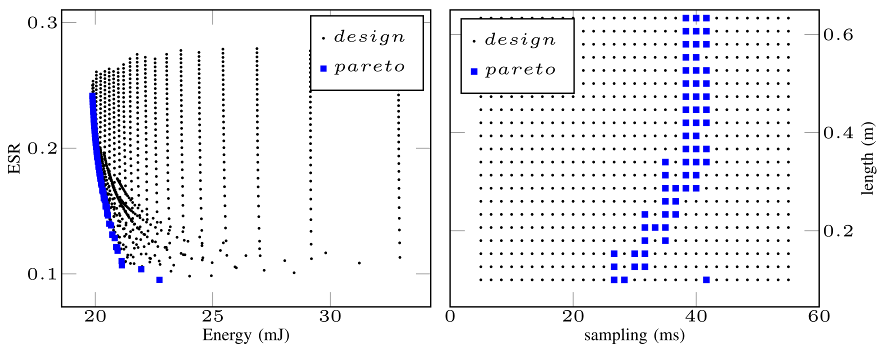
Figure 3.4: Stability region for different CA gains.

The power consumption for a fixed pendulum length is shown in Figure 3.7. It is visible that smaller h reduces energy due to better control performance, but for very small h the required processing power outweighs the savings. The result is an interesting trade-off between the control and power performance of the whole system, which we study next.

4) Control-to-Energy Trade-Off: Figure 3.5 (a) shows the scatter graph for power consumption and control quality for the discussed design space. An ideal system would be located in the bottom left. The Pareto front (in blue) shows all potentially beneficial design points. For each system, that is not part of the Pareto front, at least one system exists with better power consumption and better control quality. Figure 3.5 (b) shows the highlighted points in the design space of sampling rate and length. The results deliver a set of superior design points, and confirm that the design space does not contain a superior pendulum length, sampling rate, or controller setting that would dominate the entire design space, which confirms the importance of the holistic DSE in order to identify superior design points.

3.3.5 Practical Evaluation

To validate the results for the modeling and the controller design, we implemented a pendulum system that supports a range of physical configurations. As examples, Figure 3.6 shows two pen-



(a) (b)

Figure 3.5: Simulation results for energy and control quality (a), and the highlighted Pareto set in the design space (b).

dulums with a length of 6(cm) and 34(cm). For each experiment, we could use different pendulum lengths, LQR gains, and sampling rates. The control program was implemented on an Cortex-M3 ARM development board. Like the DSE in the previous subsection, all the experiments could be performed by a system engineer without manually revisiting the control algorithm. We used our framework to return the energy-optimal system and parametrization for a given set of invariant system parameters (either size or sampling rate). After applying the computed configuration values to the experiment, we expected that each pendulum length has the expected stability. The experiments confirmed the assumption to be correct, since each system ran stable for at least one minute. As cross-validation we applied the controller setting of the small pendulum (6cm) for the long pendulum (34cm) and vice versa. As a result the short and long pendulums tilted after four and one seconds, respectively, which confirmed the simulations results shown in Figure 3.4.

A comparison of the measured energy values and the simulation results is shown in Figure 3.7. Notably, the actual power consumption is about 10(mW) above the values obtained in the simulation. We suspect the higher power consumption is caused higher friction in reality than considered in our PS model. However, more important is that the practical measurements validate our assumption of a cyber-physical power consumption minimum. The measurements can identify this minimum as 27(ms), which is well in range of the estimated optimum point of 30(ms) for the system.

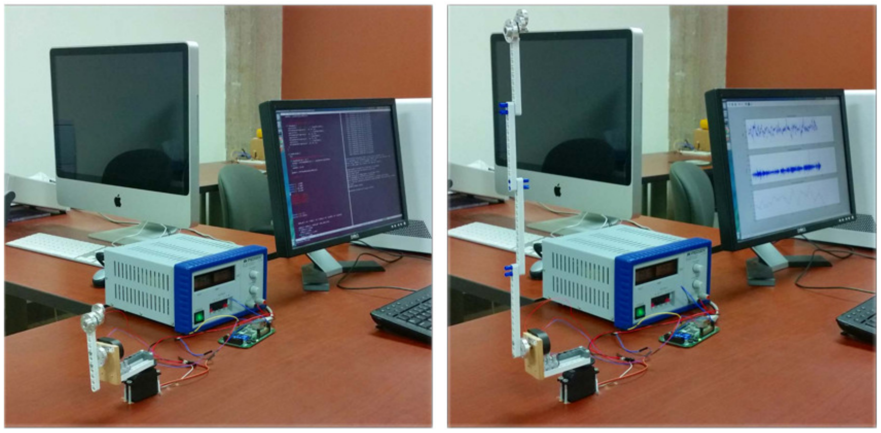


Figure 3.6: Pendulum setup using 6cm (left), and 34cm (right) lengths.

3.4 Conclusions

The results of the experiments presented in the previous section underline the importance of tailored control design in a holistic design process of CPSs. Variabilities of the physical subsystem have to be reflected in the control algorithm in order to find superior design points. We presented a DSE framework that instantiates parameterized models of the physical system, the control algorithm and the cyber system to find those design points. The applied EBM approach for the physical models opened the design space for the physical subsystems, and the derived configuration of the control algorithm parameters helped to reduce the design space to superior designs only. The actual DSE then is based on parameterized executable simulations, generated in Modelica, which facilitate a highly parallel DSE. The DSE in the pendulum use case enabled toolsupported managemant of the trade-off between control quality and energy consumption - even for non control experts.

Evidently, the modeling steps in our design flow are still complex and require knowledgeable control engineers and physical domain experts. However, following our design methodology, resulting models can be packaged as reusable parameterizable components which in turn enable systematic DSE of CPSs as part of design automation tools.

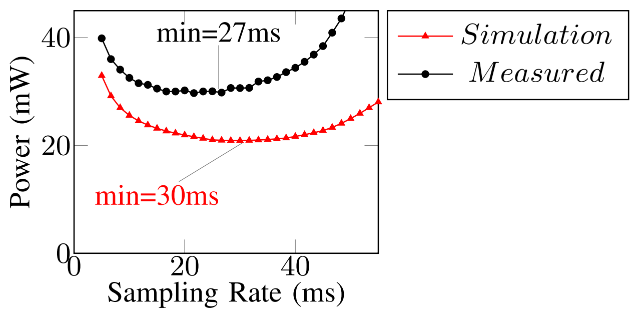


Figure 3.7: Measured and simulated energy consumption for the 34cm pendulum.

Chapter 4

Overview of Reinforcement Learning

4.1 Introduction

In this Chapter we briefly review RL and introduce the notions used in the remainder of this dissertation. In Figure 4.1 the agent-environment model of RL is shown. The "agent" interacts with the "environment" by applying "actions" that influence the environment state at the future time steps and observes the state and "reward" in the next time step resulting from the action taken. The "return" is defined as sum of all the rewards from the next steps to the end of current "episode":

$$G_t = \sum_{i=t+1}^{T} r_i \tag{4.1}$$

where G_t is the return at time t, r_i are future rewards and T is total number of steps in the episode. An "episode" is defined as a sequence of agent-environment interactions. In the last step of an episode the control task is "finished." Episode termination is defined specifically for the control task of the application.

For example, in the cart-pole balancing task, that we discuss in more detail in Section 5.3, the agent

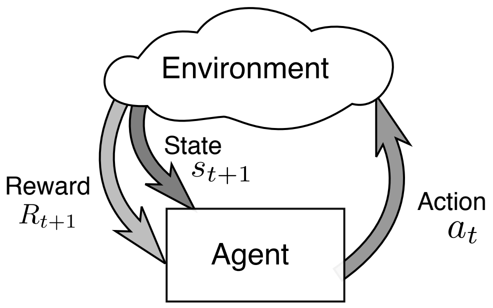


Figure 4.1: Agent-Environment interaction model in RL

is the controller, the environment is the cart-pole physical system, the action is the force command applied on the cart, and the reward can be defined as $r = 1$ as long as the pole is nearly in upright position and a large negative number when the pols falls. The system states are cart position, cart speed, pole angle and pole angular speed. The agent task is to maximize the expected return G_t, which is equivalent to preventing pole from falling for the longest possible time duration.

In RL, a control policy is defined as a mapping of the system state to the actions:

$$a = \pi(s) \tag{4.2}$$

where a is the action, s is the state and π is the policy. The expected return function which is the expected value of 'return' defined in (4.1) can be written as:

$$v_\pi(s_t) = \mathop{\mathbb{E}}_{a_\tau \sim \pi, \tau \geq t} \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \right] \tag{4.3}$$

This equation is defined for infinite episodes and the constant $0 < \gamma < 1$ is introduced to ensure that the defined expected return is always a finite value, assuming the returns are bounded.

An optimal policy is one that maximizes the expected return for all the states, i. e.:

$$v_{\pi^*}(s) >= v_\pi(s), \quad \text{for all } s, \pi \tag{4.4}$$

where v is the expected return (value) function. (4.4) means that the expected return under optimal policy π^* is equal or greater than any other policy for all the system states.

Another important concept in RL is the action-value function, $Q_\pi(s, a)$ defined as the expected return (value) if action a_t is taken at time t under policy π:

$$Q_\pi(s_t, a_t) = \mathop{\mathbb{E}}_{a_\tau \sim \pi, \tau > t} \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \right] \tag{4.5}$$

This function is related to the optimal value function introduced in (4.4) by the following equation:

$$v_{\pi^*}(s) = \max_a Q_{\pi^*}(s, a) \tag{4.6}$$

To develop algorithms to find the optimal policy, π^*, the environment dynamics need to be modeled. Contrary to most of the control design methods, many RL algorithms do not require the models to be known beforehand. The elimination of the need of modeling the system under control is a major strength of RL.

4.2 Markov Decision Process

The main assumption about the environment is that it has the *Markov Property*. A system has the Markov property if at a certain time instant, t, the system history can be captured in a set of *state variables*. Therefore, the next state of the system has a distribution which is only conditioned on the current state and the taken action at the current time, i.e.:

$$s_{t+1} \sim P(s_{t+1} | s_t, a_t) \tag{4.7}$$

The Markov property holds for many cyber-physical system application domains and therefore MDP and RL can be applied as the control algorithm. We can also define the stochastic policy which is a generalized version of (4.2) as a probability distribution of actions conditioned on the current state, i.e.:

$$a_t \sim \pi(a_t|s_t) \tag{4.8}$$

By the Markov property assumption, the RL problem can be expressed as *Markov Decision Process* (MDP). The MDP problem can be modeled with the following conditional property:

$$\begin{aligned} &p\{s(t+1), r(t+1)|\,\text{system history up to time}\, t\} \\ &= p\{s(t+1), r(t+1)|s(t), a(t)\}, \end{aligned} \tag{4.9}$$

which means that the reward and the next state only depend on the current state and action. Markov property holds for many of CPS application domains and therefore MDP and RL can be applied as the control algorithm.

The MDP problem for the optimal policy, π^*, can be solved by [86]:

- Dynamic Programming
- Monte-Carlo Methods
- Temporal-Difference (TD) Learning

The key idea in Dynamic Programming is to back up value function in a state to the previous state using Bellman equation and (4.9). In contrast to Dynamic Programming methods where the MDP model described by (4.9) should be known, in Monte-Carlo and TD methods the prior knowledge of model is not required. Monte-Carlo Method is based on running a large number of episodes from some starting state s and use the averaged total expected return obtained for all the episodes as an estimate of $v(s)$.

One important disadvantage of Monte-Carlo methods is that we have to wait until the end of an episode to update value function of the initial state. In applications with long episodes it could be impractical to implement the Monte-Carlo control algorithm. Temporal-Difference (TD) learning methods addresses this issue by an on-line, incremental approach.

In TD method, as the agent interacts with the environment it bootstraps current knowledge of the value function at a visited state to evaluate or build a policy and then after observing the actual reward from the environment, the value function of that visited state is updated. It has been shown that under certain assumptions the value function obtained by TD methods converge to one can be obtained by Monte-Carlo methods.

4.3 Q-learning

Q-learning [90] is an important example of RL solution methods and it is used in this dissertation. The Q-learning control algorithm is shown in Algorithm 5. In Q-learning, to find the optimal policy, we start from an arbitrary initial action-value function Q_0 and update it at each step of MDP by observing the reward gained by the taken action. Therefore, the optimal policy can be learned by the agent just by interacting with the environment. At each learning algorithm step, the greedy policy π_Q^* corresponding to learned action-value Q function can be defined as

$$\pi_Q^*(s) = \arg\max_a Q(s,a) \tag{4.10}$$

As the learning algorithm proceeds, the learned $Q(s,a)$ function converges to the optimal $Q^*(s,a)$ and therefore the greedy policy converges to optimal policy π^* defined in (4.4). However, to explore the action-state space for the optimal solution, an exploratory policy should be used. One example of such policies is ϵ-greedy policy defined as

Algorithm 5: Q-Learning control

```
Data: $Q_0(s,a)$                                ▷ Initial action value function
$s_0$                                           ▷ Initial State
Result: $Q^*(s,a)$                              ▷ Optimal Action-Value function
1 $Q \leftarrow Q_0$;
2 repeat
3     $s \leftarrow s_0$;
4     while $s$ is non-terminal do
5         $a \leftarrow \pi_{\epsilon,Q}(s)$;
6         Take action $a$ observe new state $s'$ and reward $r$;
7         $Q(s,a) \leftarrow Q(s,a)+$
                  $\alpha\,(r + \max_{a'} Q(s',a') - Q(s,a))$;
8         $s \leftarrow s'$;
9 until convergence;
```

$$\pi_{\epsilon,Q}(s) = \begin{cases} fa_i,\ i \in \{1,\ldots,N_a\} & \text{with probability } \epsilon/N_a \\ \pi^*_Q(s), & \text{with probability } 1-\epsilon \end{cases} \tag{4.11}$$

where N_a is the number of available actions. (4.11) means that a random action is picked instead of the greedy action with small probability ϵ in ϵ-greedy policy. In CPS applications, we choose ϵ depending on the uncertainty level in the application domain. For example, in our case-study a very small value is chosen for ϵ since the system is deterministic.

In Q-learning (Algorithm 5), the agent starts from the initial state, takes action using ϵ-policy and observes the reward. The incremental optimal policy learning is done in line 7. α is the step size parameter used to update current action-value function towards greedy target value at each iteration.

4.4 Linear Approximation of Continuous Value-Functions

The Q-learning algorithm described in Algorithm 5 is applicable to a discrete state space where the Action-Value function can be defined in a tabular representation. However, in most CPS applications the state space is continuous and cannot be expressed by a finite number of states. To be able to use methods mentioned in previous subsection, one approach is to approximate the continuous space with a linear combination of feature functions of the state variable. Coefficients of the mentioned linear combination can be expressed as the parameter vector:

$$\boldsymbol{\theta} = \begin{pmatrix} \theta_1 & \dots & \theta_{N_f} \end{pmatrix}^\intercal. \tag{4.12}$$

In this case, the Action-Value function can be expressed as:

$$Q(s,a) = \boldsymbol{\theta}_a^\intercal \boldsymbol{f}(s) = \boldsymbol{\theta}_a^\intercal \begin{pmatrix} f_1(s) & \dots & f_{N_f}(s) \end{pmatrix}^\intercal \quad a \in \mathcal{A} \tag{4.13}$$

where $f_i(s)$ are the feature functions of the continuous state space, N_f is the number of functions and $\mathcal{A}$ is the finite action set. Here we assume that only the state variables are continuous and the action space is still discrete and finite.

The objective of the learning algorithm is to estimate the parameter vector θ. The Gradient-Descent method can be used for this purpose. Assuming that f_i are binary functions, the Q-learning for linear approximate value function can be described as shown in Algorithm 6.

An example of binary features is Tile Coding [86] where each dimension of the continuous state space is divided into a number of disjoint intervals. For example, if the state space has d dimensions and each dimension is divided into k intervals, the whole space is divided into d^k hyper-cubes (tiles). Each tile i defines a feature as:

$$f_i(s) = \begin{cases} 1 & s \text{ resides in tile } i \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

Algorithm 6 with Tile Coding function approximation require low computational overhead and can be readily implemented in an embedded controller because we update the parameters towards the greedy value only for the activated features (lines 8 and 10 of Algorithm 6).

4.5 Trust Region Policy Optimization

There are two main categories of methods to find the optimal policy. In the first category, $Q_\pi(s,a)$ is parameterized as $Q^\theta_\pi(s,a)$ and the optimal action-value parameter vector θ is estimated in an

Algorithm 6: Q-Learning control for Linear approximated Value function and binary features.

```
   Data: $\boldsymbol{\theta_0}$                                  ▷ Initial parameter vector
   $s_0$                                                          ▷ Initial State
   Result: $Q^*(s,a)$                                             ▷ Optimal Action-Value function
1  $\boldsymbol{\theta_a} \leftarrow \boldsymbol{\theta_{a0}} \quad \forall a \in \mathcal{A}$;
2  repeat
3      $s \leftarrow s_0$;
4      while $s$ is non-terminal do
5          $a \leftarrow \pi_{\epsilon,Q}(s)$;
6          Take action $a$ observe new state $s'$ and reward $r$;
7          if $s'$ is terminal then
8              $\theta_{ai} \leftarrow \theta_{ai} + \alpha(r - Q(s,a)) \quad \forall i \in \{i | f_i(s) = 1\}$;
9          else
10             $\theta_{ai} \leftarrow \theta_{ai} + \alpha(r + \max_{a'} Q(s',a') - Q(s,a))$
                   $\forall i \in \{i | f_i(s) = 1\}$;
11         $s \leftarrow s'$;
12 until convergence;
```

Algorithm 7: High-Level description of Trust Region Optimization

```
  Data: $\mathcal{S}$                              ▷ Actual system or Simulation model
  $\pi^\theta$                                     ▷ Parameterized Policy
  Result: $\theta^*$                               ▷ Optimal parameters
1 repeat
2     Use $\mathcal{S}$ to generate trajectories of the system using current $\pi^\theta$;
3     Perform one iteration of policy optimization using Monte Carlo method to get $\theta_{new}$ ;
4     $\theta \leftarrow \theta_{new}$
5 until no more improvements;
6 return $\theta$
```

iterative process. The optimal policy can be defined implicitly from $Q_\pi(s, a)$. For example, the greedy policy is the one that maximizes $Q_\pi(s, a)$ in each step:

$$a_t = \arg\max_a \{Q_\pi(s_t, a)\} \tag{4.15}$$

Q-learning which is discussed in Section 4.3 is one of example of this category of methods.

In the second category, which is called policy optimization and has been successfully applied to large-scale and continuous control systems [29], the policy is parameterized directly as $\pi^\theta(a_t|s_t)$ and the parameter vector of the optimal policy θ is estimated. The Trust Region Policy Method (TRPO) [79] is an example of the second category of methods that guarantees monotonic policy improvement and is designed to be scalable to large-scale settings. In each iteration of TRPO, a number of MDP trajectories are simulated (or actually experienced by the agent) and θ is updated to improve the policy. A high level description of TRPO is shown in Algorithm 7.

Chapter 5

Adaptive Embedded Control of Cyber-physical Systems Using RL [1]

5.1 Introduction

In a cyber-physical system (CPS), most generally, a physical system is controlled by an embedded control system. The embedded control system (ECS) is the cyber part of the CPS. The ECS contains the control program that periodically processes sensor inputs and generates actuator outputs to achieve the stability, quality and performance goals of the CPS. The performance of the ECS is determined by hardware decisions, such as the applied computation platform, but also by software-defined decisions such as sampling rate and resource allocation.

Most of today's embedded control design approaches assume the ECS parameters to be fixed quantities, which are set at design time. Using classical control theory, the system parameters are set to work in the most challenging (worst case) scenario, for which the designer validates the stabil-

[1]This chapter is mainly reprinted from: Buini, Hamid Mirzaei, Steffen Peter, and Tony Givargis. "Adaptive embedded control of cyber-physical systems using reinforcement learning" IET Cyber-Physical Systems: Theory & Applications 2, no. 3 (2017): 127-135. IET, Copyright (2017), with permission from IET.

ity of the system. Such over-engineering results in resource usage inefficiency, for example when the sampling rate designed for temporary high-bandwidth disturbances or non-linear dynamics exceeds the required value for the current system state.

In this chapter we investigate the feasibility and the effect of on-line adaptation of ECS parameters to improve the resource utilization and energy consumption of the ECS and the entire CPS. Adaptive parameters have already been applied in isolated cyber systems, for instance to dynamically tune the voltage and frequency of a system [45]. However, the approaches do not consider the effect of the changes on the physical part of the CPS. Different sampling rates and computation settings influence the stability and correctness of the CPS, as well as its overall power consumption, with non-trivial trade-offs [16].

Therefore one of the main challenges of adaptive ECSs is to model and understand the effects of parameter tuning of the ECS on the physical system dynamics and control performance. Existing approaches [19, 77] rely on classical control theory and require complex application-specific models. To reduce the modeling complexity, this work relies on Reinforcement Learning (RL). In RL methods, the prior knowledge about the system dynamics is not required because RL can learn the optimal control policies just by experiencing the environment and observing the reward signal. Therefore, RL is a promising candidate to control time-varying and non-linear systems with uncertainties in the model or system states. RL has already been successfully demonstrated to control real-world physical systems, such as autonomous transportation [32], smart grids [46] and robotics [59], however, without considering the effects of the ECS parameters.

In our work, we investigate if the benefits of RL are applicable not only to learn properties of the control part of the system, but to adapt attributes of the ECS at run-time to improve usage of system resources. Specifically, we present the Adaptive ECS (A-ECS) framework that utilizes RL to control the sampling time depending on the system state, i.e. at each sampling time the controller determines the next sampling time. Using A-ECS we show that processing time and consumed energy can be reduced by 20% for the classical cart-pole example, compared to an optimal implementation with fixed controller settings. At the same time A-ECS improves the control quality in presence of model uncertainties, compared to fixed controllers and event-triggered controllers (ETC). Our results are obtained with a novel cloud-based co-simulation framework. Theoretical and experimental results for two benchmark applications indicate the practical suitability of RL to control on-line parameters of ECSs as part of CPSs.

The chapter is structured as follows. We review related work in Section 2.2 and introduce RL in Chapter 4. In Section IV we present our framework for the on-line adaptation of ECS properties. Section V and VI present experimental setups and our results, before we conclude the chapter.

5.2 Adaptive ECS based on Reinforcement Learning

In this section, first, we explain our proposed Adaptive ECS (A-ECS) framework to extend the adaptive control concept to change embedded system parameters (e.g. sampling time or voltage) in real-time based on RL methods. We apply the term *adaptive* not only for the controller, but in a broader sense, that is changing ECS system parameters, such as sampling time or memory allocation, based on the online system state. We also introduce the specific variable sampling time ECS (VS-ECS) as an example of A-ECSs where the controller sampling time is changed in real-time to realize more efficient embedded control in CPS applications.

In the second part of this section, we discuss our cloud-based evaluation framework as an extension to facilitate simulation-based design and evaluation of A-ECS. Further, we explain the steps to apply our methods on a generic CPS application.

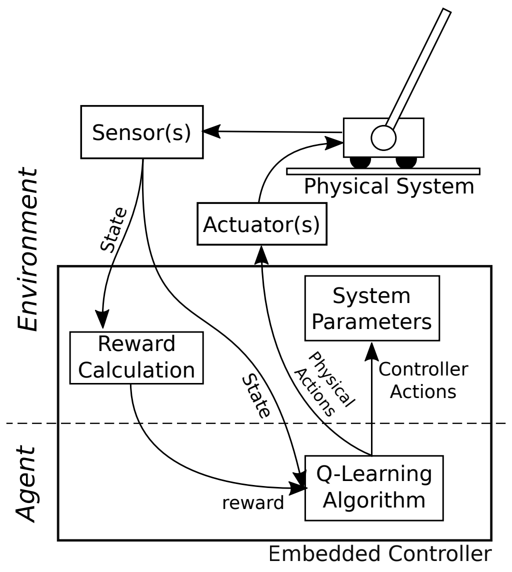


Figure 5.1: Proposed RL based Adaptive ECS

5.2.1 A-ECS Reinforcement Learning Environment and Actions

The A-ECS can be realized by following two extensions to the conventional RL based control algorithm.

- Since, RL does not require prior assumptions about the environment and the available action set, RL can be used on a broader definition of an environment that includes the physical system along with elements of the embedded controllers.
- We can extend the action set to include actions that change ECS parameters such as sampling time and memory allocation in real-time.

Once the extended RL control problem is solved, the optimal policy will include the ECS parameter real-time adaptation and physical system control commands at the same time. This idea is outlined in Figure 5.1 where the environment/agent boundary is crossing the ECS so that system parameters of ECS are included in the environment.

To realize the explained A-ECS, we augment the action vector by parameter change actions. Formally, we can represent RL action vector as:

$$\boldsymbol{a} = (\boldsymbol{a_p}^\intercal \quad \boldsymbol{a_c}^\intercal)^\intercal \tag{5.1}$$

where a_p is a vector containing the actions that influence the physical system such as force applied to cart in cart-pole balancing task and a_c is the vector of actions that change the ECS parameters, such as sampling time. The remaining RL elements correspond to conventional RL approaches, as discussed in Chapter 4. Therefore, we can define a reward so that the agent optimize the objectives of CPS system using conventional RL algorithms. Hence, the reward calculation based on the system state and the online learning algorithm can be integrated in the ECS.

Now, we can describe variable sampling time ECS (VS-ECS) as an example of A-ECS described above. In VS-ECS, the sampling time is changed in a fine grained manner, i.e. in each sample time the controller decides about the very next sampling time. The controller can choose from a limited number of available sampling times. Using a variable sampling time scheme, we expect to reduce processing time and system power by decreasing the sampling rate whenever fast sampling is not required to stabilize the system. There is a trade-off in selecting number of available sampling times. If this number increased we have more flexibility and possibly better performance. On the other hand, larger number of selections degrade the performance of the learning algorithm due to the optimization problem that needs to be solved in each time step which scales exponentially with the number of possible actions.

For the VS-ECS the only controller parameter is sampling time h. Therefore, the action vector defined in (5.1) can be rewritten as:

$$\boldsymbol{a} = (\boldsymbol{a_p}^\intercal \quad h)^\intercal \tag{5.2}$$

In Algorithm 8, the RL algorithm for VS-ECS based on RL and Q-learning is described.

Algorithm 8: Variable Sampling-Time Embedded Control

```
1 $s \leftarrow s_0$;
2 while true do
3     $(\boldsymbol{a_p}^\intercal \quad h)^\intercal \leftarrow \pi_Q^*(s)$;
4     apply $\boldsymbol{a_p}$ to the physical system (actuator);
5     wait for $h$ seconds ;
6     observe new state $s'$ ;
7     evaluate the reward based on observed $s'$;
8     Perform one step of Algorithm 6 to learn $\boldsymbol{\theta}^*$ (lines 7 to 10);
9     $s \leftarrow s'$;
```

5.2.2 Cloud-Based Evaluation Framework

While A-ECS can be used to develop algorithms to control the physical system and change ECS parameters online with no prior modeling of the system, model-based simulations help to learn preferable parameters and policies, and test identified settings before deployment. Furthermore, with efficient simulation models we can speed up the learning process.

To improve the performance of those simulations, we propose a parallel cloud-based evaluation process using a simulation model of the physical system, the ECS and the RL algorithm. The approach helps to find a superior policy by running multiple instances of the simulation and picking the learned parameters of the instance with maximum performance. In all discussed cases, the RL based ECS can apply the learned parameters to change ECS parameters online.

Our simulation approach is shown in Figure 5.2. While the Q-learning algorithm is an iterative and therefore a sequential algorithm inherently, we still can leverage recent cloud-based parallel platforms to run multiple instances of the simulation model for statistical evaluation of overall CPS performance. To realize this requirement, we require the physical model, ECS and Q-learning algorithm expressed as ordinary differential equations (ODE). ODEs can be efficiently solved utilizing

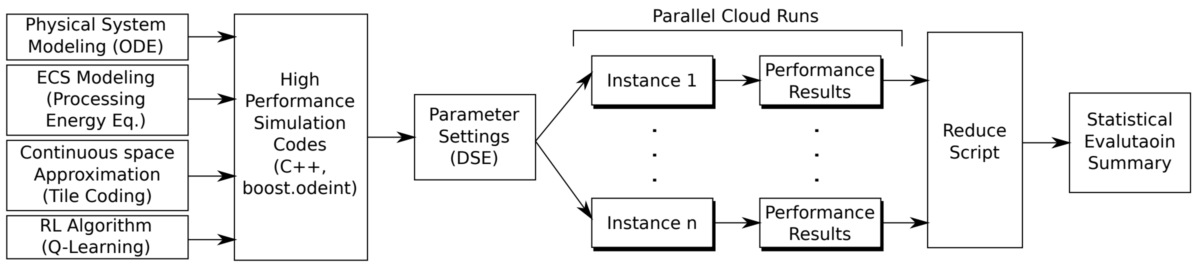

Physical System Modeling (ODE)
ECS Modeling (Processing Energy Eq.)
Continuous space Approximation (Tile Coding)
RL Algorithm (Q-Learning)
High Performance Simulation Codes (C++, boost.odeint)
Parameter Settings (DSE)
Parallel Cloud Runs
Instance 1
Instance n
Performance Results
Performance Results
Reduce Script
Statistical Evalutaoin Summary

Figure 5.2: Cloud-based evaluation framework

C++ and the Boost odeint library [3] to create a native executable binary file for the simulation. We can launch multiple instances to run the simulation with different random seeds. Then we can run a "reduce" script that aggregates simulation results of multiple instances,i.e. training curves (RL return vs training step number) and episode trajectories. The reduce script also generates evaluation statistical results. For example we can pick the learned parameters of the instance that achieves the maximum performance. Figure 5.2 shows the flow of the evaluation framework. In this figure, some examples are given for each block inside the parenthesis.

Although the described evaluation framework is not strictly "model free", for many complex systems it is a considerably easier tasks to build simulation models instead of explicit models needed in conventional control design methods. Even if we can develop explicit or the analytical models of challenging systems (e.g. non-linear, hybrid, time-varying, etc.), control algorithm design is not trivial using conventional methods.

5.2.3 A-ECS Development Workflow

To summarize this section, we provide a list of required steps to apply our framework for a CPS. The steps are:

1. Identify the RL elements in the CPS, i.e. environment and actions. Especially, it should be decided which parts of the ECS can be changed in real-time and what are the actions that apply these changes.

2. Design a reward formulation based on the performance objectives. In contrast to conventional control theory we can address actual design objectives directly, by rewarding the agent (controller) proportional to the most important performance metrics and penalize it with large negative rewards in case of failures.

3. Choose an RL algorithm to learn the optimal policy. For example, Q-learning algorithm explained in section 4.3 can be used.

4. Choose an approximation method for the continuous state space. For example, the Tile coding described in Section 4.4 is one of the possible approaches. Recent deep neural network representation method is an alternative for more complex systems [59].

5. Develop the simulation codes for the physical system and ECS. Also, implement the RL algorithm and reward calculations. Next, integrate all the mentioned components and performance measure output generation codes.

6. Choose system and RL parameters based on available heuristics or by iterative design space exploration using proposed framework. Some example of these parameters are ϵ, α, number of tilings and number of grids for each continuous state space dimension.

7. Build executable of the simulation. launch independent parallel instances to run the simulation models for different random seeds.

8. Use the “reduce” script to extract statistical information such as average or maximum performance.

5.3 Case Study 1: Cart-Pole Swing up task

In this section, we follow the steps listed in A-ECS workflow for the cart-pole swing up task. We apply the cloud-based evaluation framework described in Section 5.3.3 to show the performance

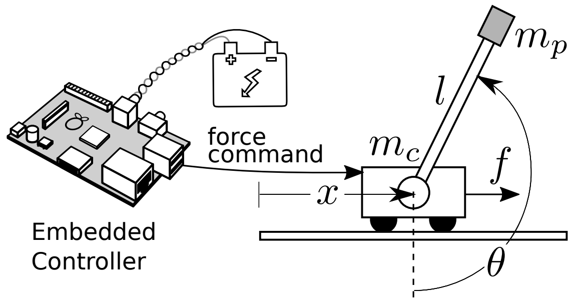


Figure 5.3: Cart-pole case study

improvement using the variable sampling time.

Consider the cart-pole system depicted in Figure 5.3. The processor, powered by a battery, can generate force commands applied to the cart. The control task is to swing up the pole from fall position to upright position and keep the pole upright for the longest time period possible using the limited energy in the battery. We explain the steps to develop the VS-ECS to balance this system using a digital controller.
The first step is to define different elements of the RL framework:

Environment: In RL, the environment is defined as the part of system which can be influenced by the agent. By this definition, the environment is the physical system in the classical cart-pole example because the applied force is the only action available to the agent. In A-ECS the concept of environment had to be extended to include properties of the controller itself because the agent can change the controller parameters dynamically.

Agent: In A-ECS the agent is the embedded controller. More precisely, the “fixed” elements of the controller is the agent and the “varying ” elements are considered part of the environment as explained before.

Actions: A-ECS supports two set of actions: physical system actions, and controller parameter tuning actions. In the cart-pole example, the force command to the cart is the physical action and the sampling time is the controller parameter action. Both actions are continuous variables, but for simplicity they are defined as discrete quantities in the case study. The force can be zero, or maximum force in any of two directions (right and left in Figure 5.3). The sampling time can be chosen from some bounded number of available choices. Therefore, we define the action vector consisting of force command and sampling time as

$$\boldsymbol{a} = (\ f \quad h\ )^{\intercal} \tag{5.3}$$

System State Variables: The system state variables are:

$$s = (\ x \quad \dot{x} \quad \theta \quad \dot{\theta} \quad e\ )^{\intercal} \tag{5.4}$$

where x is the cart position, θ is the pole angle and e is the current battery energy.

Policy π: The policy π is defined as mapping of system state to optimal actions, that is the force applied to cart and the next sampling time as a function of current physical system state and the battery storage.

Reward: In the classical cart-pole example the reward is defined as positive value, (e.g. one) if the pendulum is in the upright position with some tolerance ($\pi - \delta < \theta < \pi + \delta$) and a large negative value if the pole falls. In our example we define the reward as the time period that the controller can keep the pole in upright position. By this definition, the agent tries to use longer sampling times to save processing power to be able to balance the pole for a longer time. We also add a term proportional to the angular distance of pole to the upward position to encourage swinging the pole.

Next, we chose Q-learning and Tile coding approximation function to implement the RL algorithm as described earlier in this chapter, while other state-of-the art approaches can be used as well.

The next step is to simulate the physical system and processing power consumption. In the next subsections we describe the details of simulation models that we implemented in C++ to use them in the cloud-based evaluation process. The final steps apply the simulation code in the cloud computing platform and summarize the results by the analysis scripts. In Section IV the results of proposed VS-ECS applied on the cart-pole case study are given.

5.3.1 Cart-pole dynamics

Nomenclature

x	cart position
θ	pole angle
l	pole length
m_c	cart mass
m_p	pole mass
T	kinematic energy
U	potential energy
k_1	cart viscous friction coefficient
k_2	pole angular viscous friction coefficient
f	applied force to the cart

Now we explain the modeling of the physical system that is implemented in the simulation model used in the evaluation framework in Section 5.3.3. The cart-pole system is modeled using dynamics differential equations. The kinematic and potential of the cart-pole system is derived by:

$$T = \frac{1}{2}m_c\dot{x}^2 + \frac{1}{2}m_p\left((\dot{x} + l\dot{\theta}\cos\theta)^2 + l^2\dot{\theta}^2\sin^2\theta\right) \tag{5.5}$$

$$U = -m_p gl\cos\theta \tag{5.6}$$

The Lagrangian using (5.6) can be written as:

$$L = \frac{1}{2}(m_c + m_p)\dot{x}^2 + m_p l\dot{\theta}\dot{x}\cos\theta + \frac{1}{2}m_p l^2\dot{\theta}^2 + m_p gl\cos\theta \tag{5.7}$$

We can write the differential equations of the pole motion as

$$(m_c + m_p)\ddot{x} + m_p l\ddot{\theta}\cos\theta - m_p l\dot{\theta}^2 \sin\theta = F - k_1\dot{x} \tag{5.8}$$

$$m_p l\ddot{x}\cos\theta + m_p l^2\ddot{\theta} + m_p gl\sin\theta = -k_2\dot{\theta} \tag{5.9}$$

Finally, solving (5.9) for the linear acceleration of cart and angular acceleration of pole we have:

$$\begin{aligned}
\ddot{x} &= \frac{1}{m_c + m_p\sin^2\theta}\times \\
&\left(f - k_1\dot{x} + m_p\sin\theta(l\dot{\theta}^2 + g\cos\theta) + k_2\dot{\theta}\cos\theta\right)\ddot{\theta} && (5.10) \\
\ddot{\theta} &= \frac{1}{l^2(m_c + m_p\sin^2\theta)}\times \\
&\Big(-(m_c + m_p)k_2\dot{\theta}/m_p - lf\cos\theta + lk_1\dot{x}\cos\theta \\
&- (m_c + m_p)gl\sin\theta - m_p l^2\dot{\theta}^2\sin\theta\cos\theta\Big) && (5.11)
\end{aligned}$$

The equations (5.10) and (5.11) are highly nonlinear but we can still RL framework to control the physical system with this nonlinear dynamics.

5.3.2 Processing power modeling

Nomenclature

t_r	processing time
h_i	ith sampling time
p_r	processing power in run mode
p_s	processing power in idle mode
N	total number of sampling times (steps) before the battery energy ends
T	total time before the battery energy ends
E_{proc}	total processing power

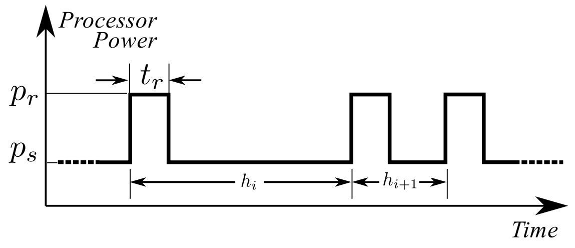


Figure 5.4: Processing power temporal modeling

Now we explain the simulation modeling of processing power, which is directly applied for the results provided in Section 5.3.3. We assume that the processor of the ECS is in sleep mode between each two successive control routine invocations. The mentioned idle time can be used to do other processing tasks, but as we are focused on the power consumption of the control task we can simply assume that the controller is in sleep mode in idle time to save energy. The power consumption scheme with this assumption is shown in Figure 5.4. The total processing energy is modeled by

$$\begin{aligned} E_{proc} &= \sum_{i=1}^{N} \left(p_r t_r + p_s(h_i - t_r)\right) \\ &= N(p_r - p_s)t_r + p_s T \end{aligned} \tag{5.12}$$

while the total time T is defined as sum of all N sampling times:

$$T = \sum_{i=1}^{N} h_i \tag{5.13}$$

For a fixed total time T, that means that longer sampling times, h_i results in lower total steps N, and lower N value in (5.12) results in lower E_{proc} which means higher power efficiency.

5.3.3 Simulation Results

In this subsection we describe the results for two experimental setups to investigate the efficiency of the proposed VS-ECS approach. The first setup is a swing-up and balance task, the second setup addresses the balance-only of the cart-pole example. We also implement and simulate event-triggered controller for the first setup as the baseline method. We will compare the results with our proposed method in the next subsection. For each setup we conducted three experiments with different sampling schemes:

- Fixed sampling time (with value h_1),
- Fixed sampling time (with value h_2), and

Table 5.1: Parameter value settings for the experiment

Parameter	Value
x range	$\pm$ 6 (m)
θ range	$\pm$ 2 π (rad)
θ balance range	$\pi\pm$ 0.1 (rad)
l	20 (cm)
m_c	1 (kg)
m_p	0.1 (kg)
k_1	0 ($\frac{N}{m/s}$)
k_2	0 ($\frac{Nm}{1/s}$)
ϵ	0.001
α	0.7
f_{max}	200 (N)
t_r	200(μs)
p_r	220 (mW)
p_s	16 (μW)
Battery capacity	0.3 (J)
# Tiling grids/dimension	7
# Tilings	40

- Variable sampling time (with values either h_1 or h_2 decided in real-time and in each control step).

We use the simulation models and the Q-learning algorithm explained previously to run the experiments. Table 5.1 lists system parameters that are used in the experiments. All experiments are done for 25 million steps. After every batch of 10,000 steps, the framework evaluates the control policy learned by the RL agent. This is done by running the greedy policy and calculating the return which is defined as the time period in which the agent was able to keep the pole almost in upward position ($\pi - 0.1 \leq \theta \leq \pi + 0.1$) before the battery energy is completely depleted. An additional term in the return function encourages the agent to swing up the pole. The overall return is determined by the following equation:

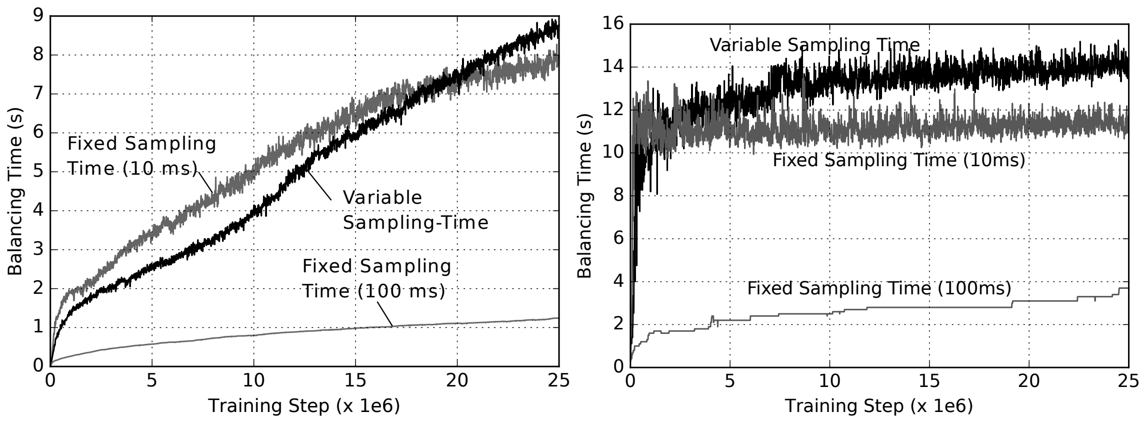
$$r = 10^{-6}(\pi - |\pi - \theta|) + \begin{cases} h_i & |\pi - \theta| \leq 0.1 \\ 0 & \text{otherwise} \end{cases} \tag{5.14}$$

where h_i is selected sampling time at time step i.

For each experiment we study the average balancing time and the identified maximum balancing time. Due to the invariant energy supply, a longer balancing time indicates a lower average power consumption, and therefore is desirable.

Swing-up and Balance Task

For the swing-up and balance experiment, each episode starts from the state where cart-pole system is still with $x = 0$ and $\theta = 0$ and ends whenever the battery energy is fully depleted or one of x or θ passes the allowable range listed in Table 5.1. $h_1 = 10(ms)$ and $h_2 = 100(ms)$ are used for the experiments. Conventional fixed sampling-time approaches have a desirable performance for the



(a) (b)

Figure 5.5: (a) Average (b) Maximum balancing time (Return) achievable by the ECS in swing-up and balance task

h_1 and fail in most cases when using h_2. We expect that the variable sampling time can achieve a higher performance by switching between the two sampling times in real-time.

Figure 5.5(a) shows the learning curves for the average balancing time for the swing-up and balance task. The plots are generated by averaging results of 200 runs on 32 instances launched by the cloud-based evaluation tool. The total simulation time was around 6 hours on Intel Xeon E5-2600 processors. We see that the larger fixed sampling time results in a short total balancing time. The reason is the severe instability due to the large sampling time. The VS-ECS performance is lower at short-term, but starts to outperform the fast fixed sampling time after around 19×10^6 learning steps, since it can utilize the two modes of operations.

At the beginning, the agent should act fast to move to pole to upright position quickly, but after that the system dynamics is slow around the balancing point and the agent should do small corrections with a slower rate that the swing up phase (Figure 5.6). Therefore the probability of selecting longer sampling time is higher in the balancing state.

The benefit of VS-ECS is more obvious when we look at the maximum balancing time, shown in Figure 5.5(b). VS-ECS identifies better settings already after less than 10×10^6 learning steps and

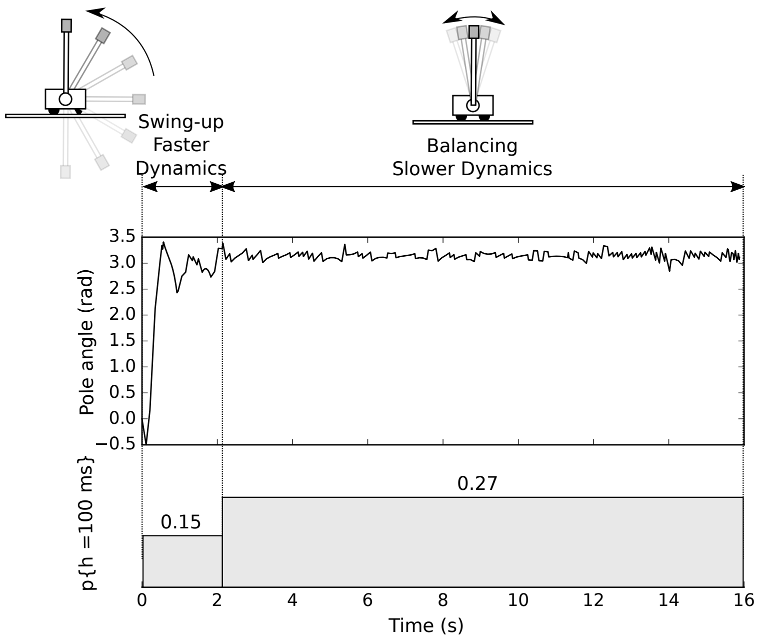


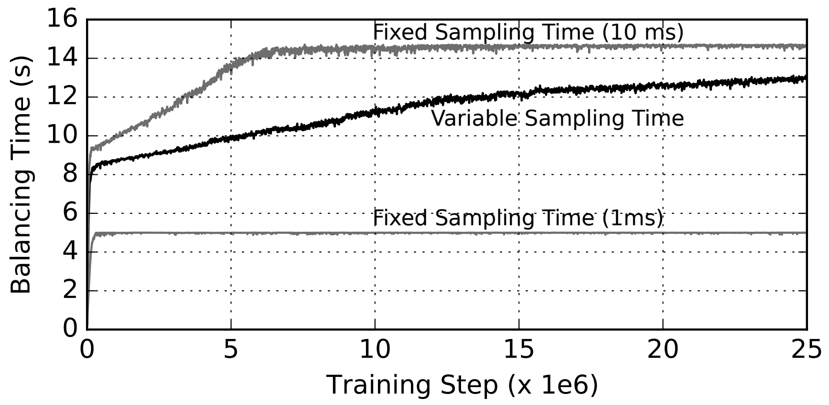
Figure 5.6: Two modes in swing-up and balance task shown by the pole angle time plot. Probability of selecting longer sampling time by A-ECS is shown in the lower time plot.

is able to balance the pole about two seconds longer than with the fast fixed sampling rate.

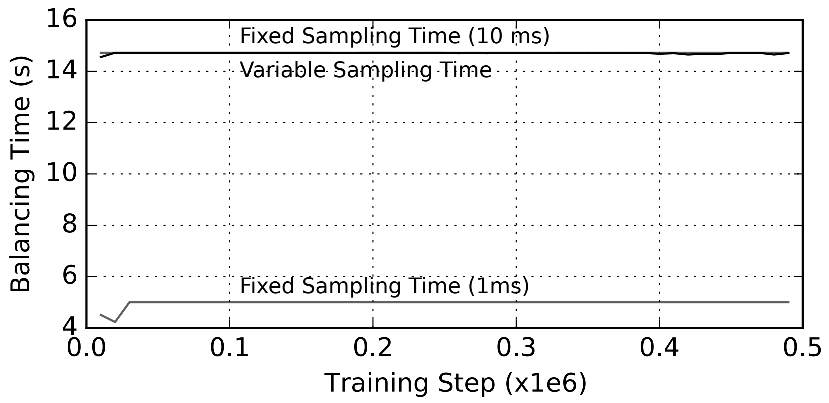
Balance-only Task

The second setup considers the upright balancing time only. The control problem in this case is less complex since the pole is already close to upright positions at the start ($\pi - 0.04$). In contrast to the previous example, both applied fixed sampling times ($h_{slow} = 10(ms)$ and $h_{slow} = 1(ms)$) can be used to control the system. In the new experiment setup, the allowable balance range is tighter ($\pi - 0.05 \leq \theta \leq \pi + 0.05$).

Figure 5.7 shows the learning curves for the average and maximum balancing time. It can be seen that the faster fixed sampling time exhausts the battery earlier, caused by the higher processing power. However, the results also show that in average the slower fixed sampling time outlasts VS-ECS, while the maximum balancing time in both cases are equal. The results confirm that a fixed sampling scheme is more suited in system with unimodal dynamics, since the system does



(a)



(b)

Figure 5.7: (a) Average (b) Maximum balancing time (Return) achievable by the ECS in balance-only task

not need to switch between modes. In these cases, VS-ECS requires learning to converge to the static behavior of the optimum static systems.

5.3.4 Comparison to Event-Triggered controller

In this subsection we compare the performance of proposed RL based VS-ECS controller with an Event-Triggered Controller (ETC) as the state-of-the-art non-uniform sampling solution for the cart-pole example. The performance metric is the balancing time as discussed in the previous subsection. The ETC for cart-pole system designed based on the method described in [64] and

[30]. ETC is realized by implementation of two functions, *event function* and *feedback function*. The event function, $\varepsilon : \chi \times \chi \to \mathbb{R}$, where χ is the state space indicates if a new control calculation is needed ($\varepsilon \leq 0$) or not ($\varepsilon > 0$). The first argument of ε is the memorized state of the system at last control update and the second argument is the current state. The feedback function, $\gamma : \chi \to \mathcal{U}$, is the state feedback control law where $\mathcal{U}$ is the control input space. Shortcoming of ETC is the required evaluation of the event function on a regular basis to detect the control update event where as in VS-ECS the embedded controller can go to sleep between successive control updates.

The controller which is proposed in [30] has two modes: *Swing up* and *Stabilization*. The controller starts at Swing up mode and after it reaches a specific angle (close enough to upright position) switches to stabilization mode. The controller uses energy control in swing up mode and Linear Quadratic Regulator (LQR) in the stabilization mode using a linearized model. The event and feedback functions are derived for both modes in [30]. The ETC controller has a number of tunable parameters. We selected the optimal settings by exhaustive search of the design space in each case.

The physical parameters of the cart-pole systems are the same as for the previous experiments (Table 5.1). To study the improvement of control object metric by using ETC the experiment is also repeated for an invariant sampling time controller with fixed period of 10 ms same as VR-ECS experiments. The results of the ETC simulations are compared with the VS-ECS maximum performance (subsection 5.3.3) in Table 5.2. ETC's extended balancing time is caused by ETCs continuous control signal, while in RL approach, discrete control is used to limit the state space dimension, resulting in more variation of the pendulum angle. However, in the RL based approach adaptive rate results in more balancing time improvement comparing to ETC which is the main contribution of this chapter.

The robustness of VS-ECS and ETC are also compared by an experiment where the system is designed for cart weight of $1(kg)$ but the actual cart weight $0.7(kg)$. The results are also shown in Table 5.2. Here we see that ETC cannot stabilize the system, since ETC relies on an exact system model, while CS-ETC stabilizes the system. It should be noted that the reduced balancing time for

Table 5.2: Comparison of VE-ECS (proposed here) and ETC designed by the concepts proposed in [64].

	Balancing time (s)			
	Accurate Physical Model		Inaccurate Physical Model	
	Single Sampling Time	Adaptive Sampling Time	Single Sampling Time	Adaptive Sampling Time
VS-ECS	11.2	14.1	9.6	13.2
ETC	14.1	14.7	unstable	9.25

VS-ECS is not caused by model errors but by the additional weight of the pole.

5.4 Case Study 2: Mountain Car Problem

5.4.1 Problem definition

The Mountain Car example [67], discussed in this section, evaluates VS-ECS for a system with nonlinear dynamics. In the original Mountain Car example the goal is to control the acceleration of a car inside a valley in order to move it to the top of the mountain (Figure 5.8). However, the maximum acceleration of the car is limited and it can not be driven to the top of mountain in a single pass and the car has to go back and forth a number of times to get enough momentum to reach to the desired destination.

In the original Mountain Car problem, there is no computational overhead limitation and the car has not to stop at the destination and the goal is to reach to the destination on top of mountain in minimum time. Therefore, the RL reward is defined as following:

$$r = \begin{cases} -1 & \text{car has not reached to the destination} \\ 0 & \text{otherwise} \end{cases} \tag{5.15}$$

In this experiment we define two different goals to the original problem to make it a more difficult control task: 1- reach the destination with minimum computational overhead. 2- car should almost be stopped (i.e. its speed should be less than a threshold) at the destination. To realize the minimum computational objective a computational budget is defined that is decreased by one each sample time. Hence, once the car nearly stops at the destination the higher remaining computational budget means less computational overhead and should be rewarded. Using the explained problem definition, the reward defined in (5.15) is modified to the following reward function:

$$r = \begin{cases} 0 & x > 0.5 \\ 0 & b \leq 0 \\ b & |v| < 0.1 \quad and \quad 0.44 < x < 0.45 \\ -10^{-4} & \text{otherwise} \end{cases} \tag{5.16}$$

where x is the car position, v is the car speed and b is the computational budget. The small negative reward in the last case included to encourage faster task completion.

5.4.2 Simulation Results

We run the Q-learning algorithm with tile coding function approximation for three different sampling schemes: 1- fixed 0.1(s), 2- fixed 1(s) and 3-variable 0.1(s) or 1(s) chosen by RL agent. Figure 5.9 shows the results of simulation for these scenarios. The y-axis shows the return which is the remaining computational budget in the case of successful task completion and the y-axis is the training step number. All simulations are performed with starting computational budget of 60. For the fixed 1(s) the agent is unable to command the car to reach and stop at the destination

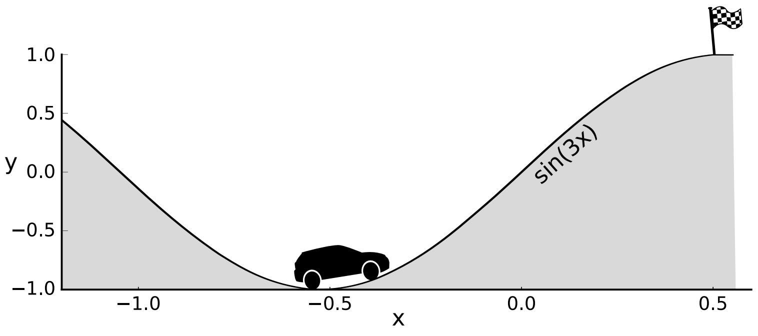


Figure 5.8: Mountain Car example

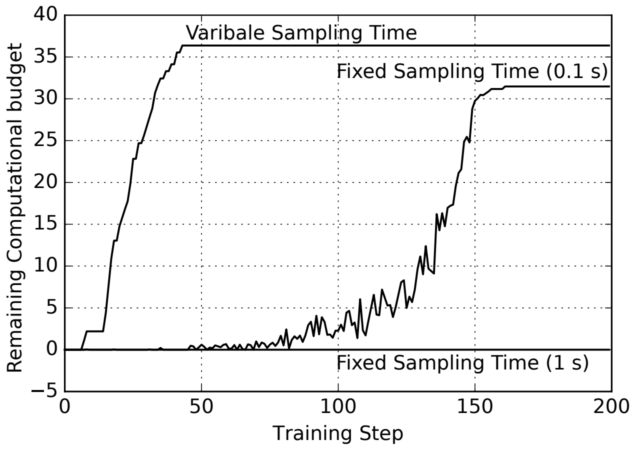


Figure 5.9: Training curves for three different scenarios in Mountain Car example

since fixed time step of one second is too coarse for the precise control needed to accomplish the problem objective. The agent is able to accomplish the task by the finer time precision of 0.1 (s). However, using a variable sampling time ECS in which sampling time is chosen among the fine and coarse sampling times, we can get higher efficiency of around 17%.

5.5 Conclusions

In this chapter we demonstrated the suitability of reinforcement learning to adapt software properties of the embedded control system (ECS) at run-time. The benefit of our adaptable online approach is a reduced average power consumption of the ECS and the overall CPS, which leads to an extended life time of battery powered CPSs. Specifically for the cart-pole example we determined an extension of the life-time by up to 20% compared to a system with optimal fixed settings. While our approach could not improve the power efficiency of a fixed optimal system in every case, all our experiments could achieve at least an equivalent performance. We further could outperform all sub-optimal fixed settings in all investigated scenarios, and improve the tolerance of the system to model uncertainties.

We presented a framework that facilitates the application of our approach to generic CPSs. We also presented an exploration tool that allows a designer to systematically evaluate the impact of different system settings and control modes.

While the work at hand is a very promising first step to use RL for the online software configuration of embedded control systems, the work contains a range of limitations that might be interesting to address in future work. In our experiments we used a dual-modal system. Adding more modes might further improve the system properties. Also applying our framework to consider voltage and frequency settings as well as resource allocations in addition to sampling rates is a promising next step. Finally, combining our approach with existing RL frameworks that focus on control properties [50] could further improve the overall design quality and system performance of CPSs, while reducing the complexity of designing the system.

Chapter 6

Autonomous Intersection Management Using RL [1]

6.1 Introduction

Previous works on autonomous intersection management (AIM) in urban areas have mostly focused on intersection arbitration as a shared resource among a large number of autonomous vehicles. In these works [28, 41], high-level control of the vehicles is implemented such that the vehicles are self-contained agents that only communicate with the intersection management agent to reserve space-time slots in the intersection. This means that low-level vehicle navigation which involves acceleration and speed control is performed by each individual vehicle independent of other vehicles and intersection agents. This approach is appropriate for minor arterial roads where a large number of vehicles utilize the main roads at similar speeds while the adjacent intersections

[1]This chapter is mainly reprinted from:H. Mirzaei and T. Givargis, "Fine-grained acceleration control for autonomous intersection management using deep reinforcement learning," 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), San Francisco, CA, 2017. IEEE, Copyright (2017), with permission from IEEE
Permission is included in Appendix A.

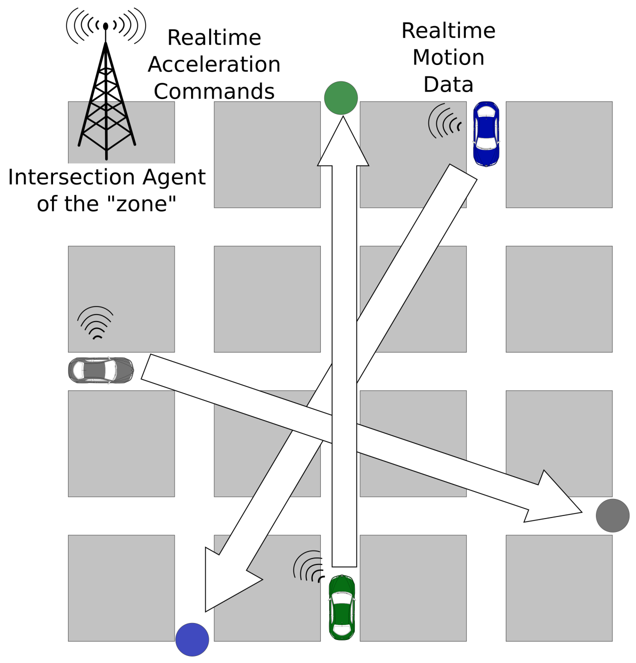

Realtime Acceleration Commands
Realtime Motion Data
Intersection Agent of the "zone"

Figure 6.1: Intersection Management Problem. The goal of the problem is to navigate the vehicles from the sources to destinations in minimum time with no collisions.

are far away.

In scenarios involving local roads, where the majority of the intersections are managed by stop signs, the flow of traffic is more efficiently managed using a fine-grained vehicle control methodology. For example, when two vehicles are crossing the intersection of two different roads at the same time, one vehicle can decelerate slightly to avoid collision with the other one or it can take another path to avoid confronting the other vehicle completely. Therefore, the nature of the AIM problem is a combination of route planning and real-time acceleration control of the vehicles. In this chapter, we propose a novel AIM formulation which is the combination of route planning and fine-grained acceleration control. The main objective of the control task is to minimize travel time of the vehicles while avoiding collisions between them and other obstacles. In this context, since the movement of a vehicle is dependent on the other vehicles in the same vicinity, the motion data of all vehicles is needed in order to solve the AIM problem.

To explain the proposed AIM scheme, let us define a “zone” as a rectangular area consisting of a number of intersections and segments of local roads. An agent for each zone collects the motion data and generates the acceleration commands for all autonomous vehicles within the zone's boundary. All the data collection and control command generation should be done in real-time. This centralized approach cannot be scaled to a whole city, regardless of the algorithm used, due to the large number of vehicles moving in a city which requires enormous computational load and leads to other infeasible requirements such as low-latency communication infrastructure. Fortunately, the spatial independence (i.e., the fact that navigation of the vehicles in one zone is independent of the vehicles in another zone that is far enough away) makes AIM an inherently local problem. Therefore, we can assign an agent for each local zone in a cellular scheme.

The cellular solution nevertheless leads to other difficulties that should be considered for a successful design of the AIM system. One issue is the dynamic nature of the transportation problem. Vehicles can enter or leave a zone controlled by an agent or they might change their planned destinations from time to time. To cope with these issues, the receding horizon control method can be employed where the agent repeatedly recalculates the acceleration command over a moving time horizon to take into account the mentioned changes. Additionally, two vehicles that are moving toward the same point on the boundary of two adjacent zones simultaneously might collide because the presence of each vehicle is not considered by the agent of the adjacent zone. This problem can be solved by adequate overlap between adjacent zones. Furthermore, any planned trip for a vehicle typically crosses multiple zones. Hence, a higher level planning problem should be solved first that determines the entry and exit locations of a vehicle in a zone.

In this chapter we focus on the subproblem of acceleration control of the vehicles moving in a zone to minimize the total travel time. We use a deep reinforcement learning (RL) approach to tackle the fine-grained acceleration control problem since conventional control methods are not applicable because of the non-convex collision avoidance constraints [33]. Furthermore, if we want to incorporate more elements into the problem, such as obstacles or reward/penalty terms for gas usage, passenger comfort, etc., the explicit modeling becomes intractable and an optimal control law derivation will be computationally unattainable.

RL methods can address the above mentioned limitations caused by the explicit modeling requirement and conventional control method limitations. The main advantage of RL is that most of the RL algorithms are "model-free" or at most need a simulation model of the physical system which is easier to develop than an explicit model. Moreover, the agent can learn optimal policies just by interacting with the environment or executing the simulation model. However, conventional RL techniques are only applicable in small-scale problem settings and require careful design of approximation functions. Emerging Deep RL methods [29] that leverage the deep neural networks to automatically extract features seem like promising solutions to shortcomings of the classical RL methods.

6.2 Problem Statement

There is a set of vehicles in a grid street plan area consisting of a certain number of intersections. For simplicity, we assume that all the initial vehicle positions and the desired destinations are located at the intersections. There is a control agent for the entire area. The agent's task is to calculate the acceleration command for the vehicles in real-time (see Fig. 6.1). We assume that there are no still or moving obstacles other than vehicles' or street boundaries.

The input to the agent is the real-time state of the vehicles which consists of their positions and speeds. We are assuming that vehicles are point masses and their angular dynamics are ignored. However, to take the collision avoidance in the problem formulation, we define a safe radius for each vehicle and no objects (vehicles or street boundaries) should be closer than the safe radius to the vehicle.

The objective is to drive all the vehicles to their respective destinations in a way that the total travel time is minimized. Furthermore, no collision should occur between any two vehicles or a vehicle and the street boundaries.

To minimize the total travel time, a positive reward is assigned to the terminal state in which all the vehicles approximately reach the destinations within some tolerance. A discount factor γ strictly less than one is used. Therefore, the agent should try to reach the terminal state as fast as possible to maximize the discounted return. However, by using only this reward, too many random walk trajectories are needed to discover the terminal state. Therefore, a negative reward is defined for each state, proportional to the total distance of the vehicles to their destinations as a hint of how far the terminal state is. This negative reward is not in contradiction with the main goal which is to minimize total travel time.

To avoid collisions, two different approaches can be considered: we can add large negative rewards for the collision states or we can incorporate a collision avoidance mechanism into the environment model. Our experiments show that the first approach makes the agent too conservative about moving the vehicles to minimize the probability of collisions. This might lead to extremely slow learning which makes it computationally infeasible. Furthermore, collisions are inevitable even with large negative rewards which limits the effectiveness of learned policies in practice.

For the above mentioned reasons, the second approach is employed, i.e. the safety mechanism that is used in practice is included in the environment definition. The safety mechanism is activated whenever two vehicles are too close to each other or a vehicle is too close to the street boundary. In these cases, the vehicle built-in collision avoidance system will control the vehicle's acceleration and the acceleration commands from the RL agent are ignored as long as the distance is near the allowed safe radius of the vehicle. In the agent learning process these cases are simulated in a way that the vehicles come to a full stop when they are closer than the safe radius to another vehicle or boundary. By applying this heuristic in the simulation model, the agent should avoid any "near collision" situations explained above because the deceleration and acceleration cycles take a lot of time and will decrease the expected return.

Based on the problem statement explained above, we can describe the RL formulation in the rest of the subsection. The state is defined as the following vector:

$$\boldsymbol{s_t} = \left(x_t^1, y_t^1, v_{xt}^1, v_{yt}^1, \ldots, x_t^n, y_t^n, v_{xt}^n, v_{y\,t}^n\right)^\intercal \tag{6.1}$$

where (x_t^i, y_t^i) and $(v_{xt}^i, v_{y_t}^i)$ are the position and speed of vehicle i at time t. The action vector is defined as:

$$\boldsymbol{a_t} = \left(a_{xt}^1, a_{yt}^1, \ldots, a_{xt}^n, a_{y\,t}^n\right)^\intercal \tag{6.2}$$

where $(a_{xt}^i, a_{y_t}^i)$ is the acceleration command of vehicle i at time t. The reward function is defined as:

$$r(s) = \begin{cases} 1 \quad \text{if} \quad \|(x^i - d_x^i, y^i - d_x^i)^\intercal\| < \eta \ (1 \leq i \leq n) \\ -\alpha \sum_{i=1}^n \|(x^i - d_x^i, y^i - d_x^i)^\intercal\| \quad \text{otherwise} \end{cases} \tag{6.3}$$

where (d_x^i, d_y^i) is the destination coordinates of vehicle i, η is the distance tolerance and α is a positive constant.

Assuming no collision occurs, the state transition equations for the environment are defined as follows:

$$\begin{aligned} x_{t+1}^i &= \text{sat}_{\underline{x},\overline{x}}(x_t^i + h{v_{xt}}^i) \\ y_{t+1}^i &= \text{sat}_{\underline{y},\overline{y}}(y_t^i + h{v_{yt}}^i) \\ {v_{xt+1}}^i &= \text{sat}_{\underline{v_m},\overline{v_m}}({v_{xt}}^i + h{a_{xt}}^i) \\ {v_{yt+1}}^i &= \text{sat}_{\underline{v_m},\overline{v_m}}({v_{yt}}^i + h{a_{yt}}^i) \end{aligned} \tag{6.4}$$

where h is the sampling time, $(\underline{x}, \overline{x}, \underline{y}, \overline{y})$ defines area limits, v_m is the maximum speed and $\text{sat}_{\underline{w},\overline{w}}(.)$

is the saturation function defined as:

$$\text{sat}_{\underline{w},\overline{w}}(x) = \begin{cases} \underline{w} & x \leq \underline{w} \\ \overline{w} & x \geq \overline{w} \\ x & \text{otherwise.} \end{cases} \tag{6.5}$$

To model the collisions, we should check certain conditions and set the speed to zero. A more detailed description of collision modeling is presented in Algorithm 9.

Algorithm 9: State Transition Function

```
   Data: $s_t$                                        ▷ State at time $t$
   $a_t$                                              ▷ Action at time $t$
   Result: $s_{t+1}$                                  ▷ State at time $t+1$
1  $a_{x_t}^i \leftarrow \text{sat}_{\underline{a_m},\overline{a_m}}(a_{x_t}^i)$;
2  $a_{y_t}^i \leftarrow \text{sat}_{\underline{a_m},\overline{a_m}}(a_{y_t}^i)$;
3  $s_{t+1} \leftarrow$ updated state using (6.4);
4  $v_{c1} \leftarrow$ find all the vehicles colliding with street boundaries;
5  speed elements of $v_{c1}$in $s_{t+1} \leftarrow 0$;
6  location elements of $v_{c1}$in $s_{t+1} \leftarrow$ closest point on the street boundary with the margin of $\epsilon$;
7  $v_{c2} \leftarrow$ find all the vehicles colliding with some other vehicle;
8  speed elements of $v_{c2}$in $s_{t+1} \leftarrow 0$;
9  location elements of $v_{c2}$in $s_{t+1} \leftarrow$ pushed back location with the distance of $2\times$ safe radius
   to the collided vehicle;
10 return $s_{t+1}$;
```

6.2.1 Solving the AIM problem using TRPO

The simulation model can be implemented based on the RL formulation described in Section 6.2. To use TRPO, we need a parameterized stochastic policy, $\pi^{\theta}(a_t|s_t)$, in addition to the simulation model. The policy should specify the probability distribution for each element of the action vector defined in (6.2) as a function of the current state s_t.

We have used the sequential deep neural network (DNN) policy representation as described in [79]. The input layer receives the state containing the position and speed of the vehicles (defined in (6.1)). There are a number of hidden layers, each followed by $\tanh$ activation functions [47]. Finally, the output layer generates the mean of a gaussian distribution for each element of the action vector.

To execute the optimal policy learned by TRPO in each sampling time, the agent calculates the forward-pass of DNN using the current state. Next, assuming that all the action elements have the same variance, the agent samples from the action gaussian distributions and applies the sampled actions to the environment as the vehicle acceleration commands.

6.3 Evaluation

6.3.1 Baseline Method

To the best of our knowledge there is no other solution proposed for the fine-grained acceleration AIM problem introduced in this chapter. Therefore, we use conventional optimization methods to study how close the proposed solution is to the optimal solution. Furthermore, we will see that the conventional optimization is able to solve the AIM problem only for very small-sized problems. This confirms that the proposed RL-based solution is a promising alternative to the conventional methods.

Theoretically, the best solution to the problem defined in Section 6.2 can be obtained if we reformulate it as a conventional optimization problem. The following equations and inequalities describe the AIM optimization problem:

$$\boldsymbol{a_t^*} = \arg\max_{\boldsymbol{a_t}} \sum_{t=0}^{T-1} \sum_{i=1}^{n} \|(x_t^i - d_x^i, y_t^i - d_x^i)^\intercal\| \tag{6.6}$$

$$\text{s. t.} \quad \underline{x} \leq x_t^i \leq \overline{x} \quad (1 \leq i \leq n) \tag{6.7}$$

$$\underline{y} \leq y_t^i \leq \overline{y} \quad (1 \leq i \leq n) \tag{6.8}$$

$$\underline{v_m} \leq {v_x}_t^i \leq \overline{v_m} \quad (1 \leq i \leq n) \tag{6.9}$$

$$\underline{v_m} \leq {v_y}_t^i \leq \overline{v_m} \quad (1 \leq i \leq n) \tag{6.10}$$

$$\underline{a_m} \leq {a_x}_t^i \leq \overline{a_m} \quad (1 \leq i \leq n) \tag{6.11}$$

$$\underline{a_m} \leq {a_y}_t^i \leq \overline{a_m} \quad (1 \leq i \leq n) \tag{6.12}$$

$$-\lfloor N/2 \rfloor \leq r_t^i \leq \lfloor N/2 \rfloor \quad (1 \leq i \leq n, r_t^i \in \mathbb{Z}) \tag{6.13}$$

$$-\lfloor M/2 \rfloor \leq c_t^i \leq \lfloor M/2 \rfloor \quad (1 \leq i \leq n, c_t^i \in \mathbb{Z}) \tag{6.14}$$

$$x_0^i = s_x^i,\ y_0^i = s_y^i \quad (1 \leq i \leq n) \tag{6.15}$$

$$x_{T-1}^i = d_x^i,\ y_{T-1}^i = d_y^i \quad (1 \leq i \leq n) \tag{6.16}$$

$${v_x}_0^i = 0,\ {v_y}_0^i = 0 \quad (1 \leq i \leq n) \tag{6.17}$$

$$x_{t+1}^i = x_t^i + {v_x}_t^i.h \quad (1 \leq i \leq n) \tag{6.18}$$

$$y_{t+1}^i = y_t^i + {v_y}_t^i.h \quad (1 \leq i \leq n) \tag{6.19}$$

$${v_x}_{t+1}^i = {v_x}_t^i + {a_x}_t^i.h \quad (1 \leq i \leq n) \tag{6.20}$$

$${v_y}_{t+1}^i = {v_y}_t^i + {a_y}_t^i.h \quad (1 \leq i \leq n) \tag{6.21}$$

$$(x_t^i - x_t^j)^2 + (y_t^i - y_t^j)^2 \geq (2R)^2 \quad (1 \leq i < j \leq n) \tag{6.22}$$

$$|x_t^i - c_t^i.b_w| \leq (\frac{l}{2} - R) \text{ or}$$

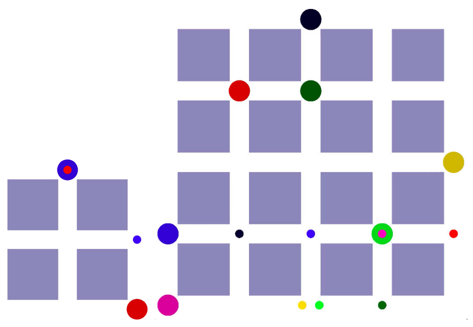


Figure 6.2: Initial setup of each episode. Small circles are the sources and big circles are the destinations. (a) small example (b) large example

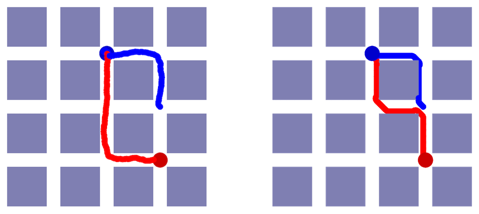


Figure 6.3: Learnt policy by (left)RL agent and (right)the baseline method for the small example

$$|y_t^i - r_t^i.b_h| \leq (\frac{l}{2} - R)\ (1 \leq i \leq n) \tag{6.23}$$

where r_t^i and c_t^i are the row number and column number of vehicles at time t, respectively, assuming the zone is a perfect rectangular grid; N and M are the number of rows and columns, respectively; b_w and b_h are block width and block height; l is the street width; R is the vehicle clearance radius; T is number of sampling times; and (s_x^i, s_y^i) is the source coordinates of vehicle i.

In the above mentioned problem setting, (6.7) to (6.12) are the physical limit constraints. (6.15) to (6.17) describe the initial and final conditions. (6.18) to (6.21) are dynamic constraints. (6.22) is the vehicle-to-vehicle collision avoidance constraint and finally (6.23) is the vehicle-to-boundaries collision avoidance constraint.

The basic problem with the above formulation is that constraint (6.22) leads to a non-convex function and convex optimization algorithms cannot solve this problem. Therefore, a Mixed-Integer

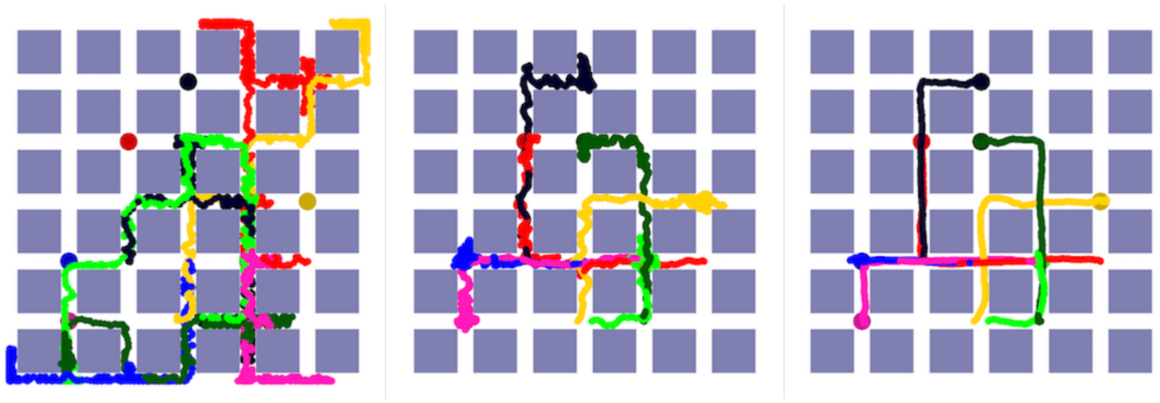


Figure 6.4: Learnt policy by the AIM agent at different iterations of the training. left: beginning of the training, middle: just after the fast learning phase in Fig. 6.6. right: end of the training.

Nonlinear Programming (MINLP) algorithm should be used to solve this problem. Our experiments show that even a small-sized problem with two vehicles and 2×2 grid cannot be solved with an MINLP algorithm, i.e. AOA[43], in a reasonable time. To overcome this issue, we should reformulate the optimization problem using 1-norm and introduce new integer variables for the distance between vehicles using the ideas proposed in [78].

To achieve the best convergence and execution time by using a Mixed-integer Quadratic Programming (MIQP), the cost function and all constraints should be linear or quadratic. Furthermore, the "or" logic in (6.23) should be implemented using integer variables. The full MIQP problem can be written as the following equations and inequalities:

$$\boldsymbol{a_t^*} = \arg\max_{\boldsymbol{a_t}} \sum_{t=0}^{T-1} \sum_{i=1}^{n} (x_t^i - d_x^i)^2 + (y_t^i - d_x^i)^2 \tag{6.24}$$

$$\text{s. t.} \quad (6.7) \text{ to } (6.21)$$

$$b_{xt}^{\ i}, b_{yt}^{\ i} \in \{0,1\} \quad (1 \leq i \leq n) \tag{6.25}$$

$$b_{xt}^{\ i} + b_{yt}^{\ i} \geq 1 \quad (1 \leq i \leq n) \tag{6.26}$$

$$c_{xt}^{\ i,j}, c_{yt}^{\ i,j}, d_{xt}^{\ i,j}, d_{yt}^{\ i,j} \in \{0,1\} \quad (1 \leq i < j \leq n) \tag{6.27}$$

$$c_{xt}^{\ i,j} + c_{yt}^{\ i,j} + d_{xt}^{\ i,j} + d_{yt}^{\ i,j} \geq 1 \quad (1 \leq i < j \leq n) \tag{6.28}$$

$$x_t^i - x_t^j \geq 2Rc_{xt}^{i,j} - \mathcal{M}(1 - c_{xt}^{i,j}) \quad (1 \leq i < j \leq n) \tag{6.29}$$

$$x_t^i - x_t^j \leq -2Rd_{xt}^{i,j} + \mathcal{M}(1 - d_{xt}^{i,j}) \quad (1 \leq i < j \leq n) \tag{6.30}$$

$$y_t^i - y_t^j \geq 2Rc_{yt}^{i,j} - \mathcal{M}(1 - c_{yt}^{i,j}) \quad (1 \leq i < j \leq n) \tag{6.31}$$

$$y_t^i - y_t^j \leq -2Rd_{yt}^{i,j} + \mathcal{M}(1 - d_{yt}^{i,j}) \quad (1 \leq i < j \leq n) \tag{6.32}$$

$$x_t^i - c_t^i b_w \leq (\frac{l}{2} - R)b_{xt}^i + \mathcal{M}(1 - b_{xt}^i) \quad (1 \leq i \leq n) \tag{6.33}$$

$$x_t^i - c_t^i b_w \geq -(\frac{l}{2} - R)b_{xt}^i - \mathcal{M}(1 - b_{xt}^i) \quad (1 \leq i \leq n) \tag{6.34}$$

$$y_t^i - c_t^i b_w \leq (\frac{l}{2} - R)b_{yt}^i + \mathcal{M}(1 - b_{yt}^i) \quad (1 \leq i \leq n) \tag{6.35}$$

$$y_t^i - c_t^i b_w \geq -(\frac{l}{2} - R)b_{yt}^i - \mathcal{M}(1 - b_{yt}^i) \quad (1 \leq i \leq n) \tag{6.36}$$

where $\mathcal{M}$ is a large positive number.

(6.29) to (6.32) represent the vehicle-to-vehicle collision avoidance constraint using 1-norm:

$$\|(x_t^i, y_t^i)^\intercal - (x_t^j, y_t^j)^\intercal\|_1 \geq 2R \tag{6.37}$$

for any two distinct vehicles i and j. This constraint is equivalent to the following:

$$|x_t^i - x_t^j| \geq 2R \text{ or } |y_t^i - y_t^j| \geq 2R \quad \forall t, (1 \leq i < j \leq n) \tag{6.38}$$

The absolute value function displayed in (6.38) should be replaced by logical "or" of two linear conditions to avoid nonlinearity. Therefore, we obtain the following constraint which is represented by (6.29) to (6.32):

$$\begin{aligned} &x_t^i - x_t^j \geq 2R \text{ or } x_t^i - x_t^j \leq -2R \text{ or} \\ &y_t^i - y_t^j \geq 2R \text{ or } y_t^i - y_t^j \leq -2R \quad \forall t, (1 \leq i < j \leq n) \end{aligned} \tag{6.39}$$

(6.27) implements the "or" logic required in (6.39).

(6.33) to (6.36) describe the vehicle-to-boundaries collision avoidance constraint:

$$
\begin{aligned}
&|x_t^i - c_t^i b_w| \leq (\frac{l}{2} - R) b_{x_t}^{\,i} \text{ or} \\
&|y_t^i - r_t^i b_w| \leq (\frac{l}{2} - R) b_{y_t}^{\,i} \quad \forall t, (1 \leq i \leq n)
\end{aligned} \tag{6.40}
$$

which is equivalent to:

$$
\begin{aligned}
&(x_t^i - c_t^i b_w \leq (\frac{l}{2} - R) b_{x_t}^{\,i} \text{ and } x_t^i - c_t^i b_w \geq -(\frac{l}{2} - R) b_{x_t}^{\,i}) \text{ or} \\
&(y_t^i - r_t^i b_w \leq (\frac{l}{2} - R) b_{y_t}^{\,i} \text{ and } y_t^i - r_t^i b_w \geq -(\frac{l}{2} - R) b_{y_t}^{\,i}) \\
&\qquad \forall t, (1 \leq i \leq n)
\end{aligned} \tag{6.41}
$$

The "or" logic in this constraint is realized in (6.26).

We will show in the next subsection that the explained conventional optimization formulation is not feasible except for very small-sized problems. Another limitation that makes the conventional method impractical is that this formulation works only for a perfect rectangular grid. However, the proposed RL method in this chapter can be extended to arbitrary street layouts.

6.3.2 Simulation results

The implementation of the TRPO in rllab library [29] is used to simulate the RL formulation of the AIM problem described in Section 6.2. For this purpose, the AIM state transition and reward calculation are implemented as an OpenAI Gym [14] environment.

The neural network used to approximate the policy is an Multilayer Perceptron (MLP) which consists of three hidden layers. Each hidden layer has 100 nodes (Fig. 6.5). Table 6.1 lists the parameters for the simulation. To speed up simulation, normalized units are used for the physical properties of the environment instead of real-world quantities.

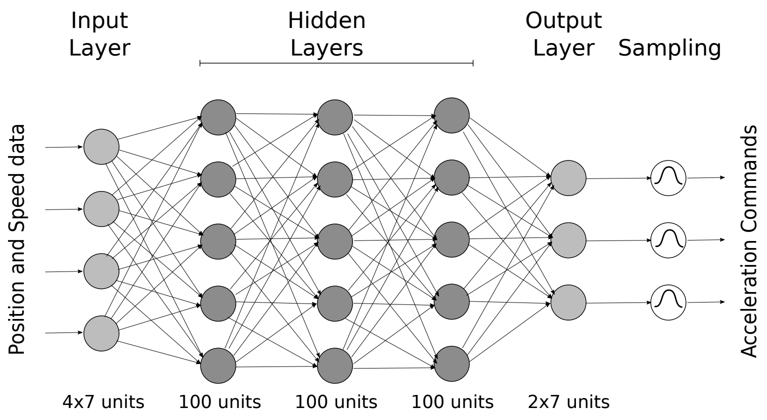


Figure 6.5: Neural network used in the simulations.

Fig. 6.2 shows the small and large grid plans used for the simulation. The small and large circles represent the source and destination locations, respectively. The vehicles are placed at the intersections randomly at the beginning of each episode. The destinations are also chosen randomly.

Table 6.1: Parameter value settings for the experiment

Parameter	Value
discount factor(γ)	0.999
distance reward penalty factor(α)	0.1
distance tolerance(η)	0.05
maximum speed(v_m)	0.8
maximum acceleration(a_m)	30
sampling time(h)	10 (ms)
maximum episode length	200
vehicle safe radius	0.02

Table 6.2: Total travel time obtained by baseline method and proposed method

	Baseline Method	Proposed Method
Small example	1.79 (s)	2.43 (s)
Large Example	no solution	11.2 (s)

When the simulator is reset, the same set of source and destination locations are used.

The small grid can be solved both by the baseline method and by the proposed RL method. However, the large grid can only be solved by the RL method because the MIQP algorithm could not find a feasible solution (which is not optimal necessarily) and was stopped after around 68 hours and using 21 GB of system memory. On the other hand, the RL method can solve the problem using 560 MB of system memory and 101 MB of GPU memory.

Table 6.2 and Fig. 6.3 show the comparison of proposed RL and baseline method results. In Table 6.2 the total travel time is provided for both methods and Fig. 6.3 shows the vehicles' trajectories by running the navigation policy obtained by both solutions for the small examples.

The learning curve of the RL agent which is the expected return vs the training epoch number is shown in Fig. 6.6 for the large grid example. This figure shows that the learning rate is higher at the beginning which corresponds to the stage where in the agent is learning the very basics of driving and avoiding collisions, but improving the policy towards the optimal policy takes considerably more time. The increase in learning occurs after two epochs when the agent discovers the policy that successfully drives all the vehicles to the destination and the positive terminal reward is gained. Moreover, the trajectories of vehicles are depicted in Fig. 6.4 at three stages of the learning process, i.e. at the early stage, at epoch 2 where the learning curve slope rapidly decreases, and the end of the training.

The total number of "near collision" incidents discussed in Section 6.2 is shown in Fig. 6.7. Fig. 6.8 shows the total travel time as a function of training iteration.

6.4 Conclusion

In this chapter, we have shown that Deep RL can be a promising solution for the problem of intelligent intersection management in local road settings where the number of vehicles is limited

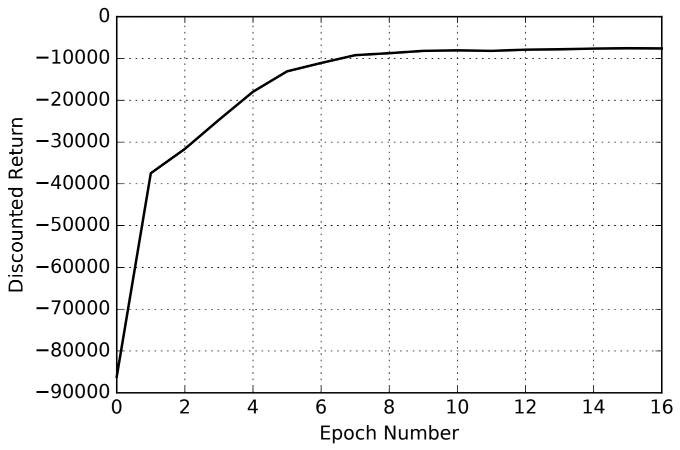


Figure 6.6: Learning curve of the AIM agent for large grid example. The discounted return is always a negative value because the return is the accumulated negative distance rewards and there is only one positive reward in the terminal state in which all the vehicles are at the destinations.

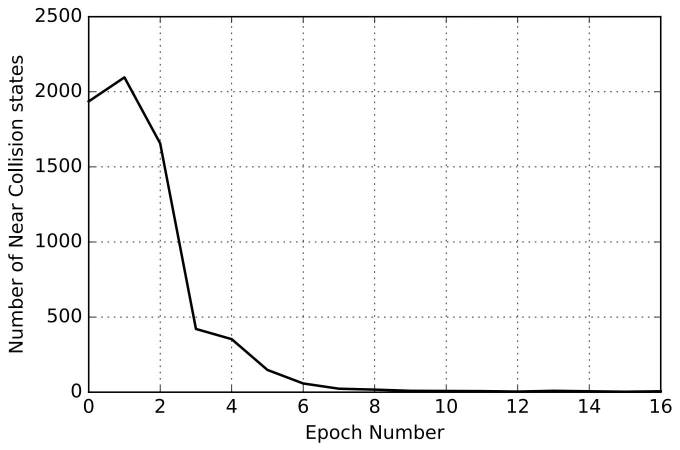


Figure 6.7: Number of near collision incidents vs training iteration number.

and fine-grained acceleration control and motion planning can lead to a more efficient navigation of the autonomous vehicles. We proposed an RL environment definition in which collisions are avoided using a safety mechanism. Using this method instead of large penalties for collision in the reward function, the agent can learn the optimal policy faster and the learned policy can be used in practice where the safety mechanism is actually implemented. The experiments show that the conventional optimization methods are not able to solve the problem with the sizes that are solvable by the proposed method.

Similar to the learning process of human beings, the main benefit of the RL approach is that an ex-

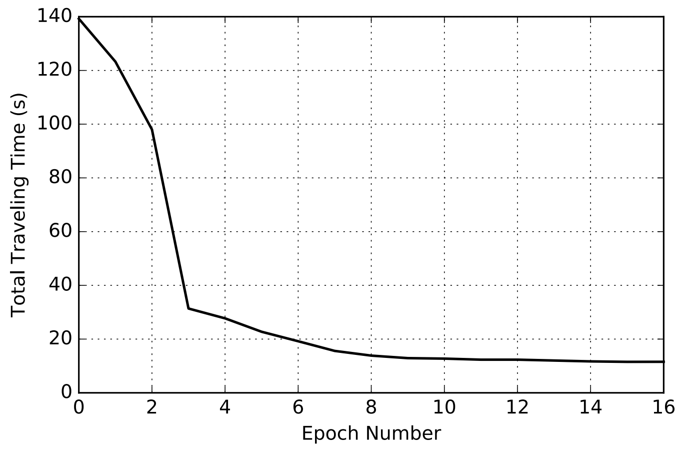


Figure 6.8: Total travel time of all the vehicles vs training iteration number.

plicit mathematical modeling of the system is not required and, more importantly, the challenging task of control design for a complex system is eliminated. However, since the automotive systems demand a high safety requirement, training of the RL agent using a simulation model is inevitable in most cases. However, developing a simulation model for a system is considerably simpler task compared to explicit modeling especially for systems with uncertainty.

While the work at hand is a promising first step towards using RL in autonomous intersection management, a number of potential improvements can be mentioned that might be interesting to address in future work. First, the possibility of developing pre-trained neural networks similar to the works in other mainstream deep learning domains that reduce learning time can be a studied in future. Furthermore, a more advanced rewards system that includes gas usage penalties is another track towards developing a practical intelligent intersection management algorithm.

Chapter 7

Link-based Micro-tolling Parameter Tuning Using RL [1]

7.1 Introduction

Advancements in connected and automated vehicle technology present many opportunities for highly optimized traffic management mechanisms [7]. One such mechanism, *micro-tolling*, has been the focus of a line of recently presented studies [20, 81, 82]. In the micro-tolling paradigm, tolls can be charged on many or all network links, and changed frequently in response to real-time observations of traffic conditions. Toll values and traffic conditions can then be communicated to vehicles which might change routes in response, either autonomously, or by updating directions given to the human driver. A centralized system manager is assumed to set toll values with the objective of optimizing the traffic flow. Many methods for computing such tolls were presented over the last century most of which made very specific assumptions regarding the underlying traffic

[1]This chapter is mainly reprinted from: Buini, Hamid Mirzaei, Guni Sharon, and Stephen Boyles. "Enhanced Delta-tolling: Traffic Optimization via Policy Gradient Reinforcement Learning" to appear in Proceedings of Workshop on Reinforcement learning for Transportation, Intelligent Transportation Systems (ITSC), 2018

model. For instance, assuming that demand is known or fixed [60], assuming that links' capacity is known or fixed, assuming that the user's value of time (VOT) is homogeneous [91], assuming traffic follows specific latency functions [74], or assuming traffic patterns emerge instantaneously [11].

A recent line of work [81, 82] suggested a new tolling scheme denoted Δ-*tolling*. Unlike previous tolling schemes, Δ-*tolling* makes no assumptions regarding the demand, links' capacity, users' VOT, and specific traffic formation models. Δ-*tolling* sets a toll for each link equal to the difference (denoted Δ) between its current travel time and free flow travel time multiplied by a proportionality parameter β. The rate of change in toll values between successive time steps is controlled by another parameter R. Despite being extremely simple to calculate, Δ-*tolling* was shown to yield optimal system performance under the stylized assumptions of a macroscopic traffic model using the Bureau of Public Roads (BPR) type latency functions [83]. Moreover, Δ-*tolling* presented significant improvement in total travel time and social welfare across markedly different traffic models and assumptions. In fact, the simple working principle of Δ-*tolling* is what allows it to act as a model-free mechanism. Whereas the original Δ-*tolling* algorithm required a single β and R parameter for the entire network, the main contribution in this chapter is a generalization of Δ-*tolling* to accommodate separate parameter settings for each link in the network. While conceptually straightforward, we demonstrate that doing so enables significant performance improvements in realistic traffic networks.

The increased representational power of Enhanced Δ-*tolling* compared to Δ-*tolling* does come at the cost of necessitating that many more parameters be tuned. A secondary contribution is a demonstration that policy gradient reinforcement learning methods can be leveraged to set tune these parameters effectively. Our detailed empirical study in Section 7.4 validates our claim that Enhanced Δ-*tolling* has the potential to improve upon the already impressive results of Δ-*tolling* when it comes to incentivizing self-interested agents to coordinate towards socially optimal traffic flows.

7.2 Problem definition and terminology

We consider a scenario where a set of agents must be routed across a traffic network given as a directed graph, $G(V, E)$. Each agent a is affiliated with a source node, $s_a \in V$, a target node, $t_a \in V$, a departure time, d_a, and a VOT, c_a (the agent's monetary value for a delay of one unit of time).

Agents are assumed to be self-interested and, hence, follow the least cost path leading from s_a to t_a. The cost of a path, p, for an agent, a, is a function of the path's latency, l_p, and tolls along it, τ_p. Formally, $cost(p, a) = l_p \cdot c_a + \tau_p$. The value of time, c_a, is assumed to be constant per agent. Although this assumption might not hold in real-world, it follows common practice in the transportation literature [27, 76, 82].

Since traffic is dynamically evolving, travel times and toll values might change over time, agents are assumed to continually re-optimize their chosen route. As a result, an agent might change its planned route at every node along its path. Each link in the network, $e \in E$, is affiliated with a dynamically changing toll value τ_e where for any path, p, $\tau_p = \sum_{e \in p} \tau_e$. Moreover, each link is affiliated with a latency l_e representing the travel time on link e. Similar to τ_e, l_e is dynamically changing as a function of the traffic state.

The objective of the system manager is to assign tolls such that if each agent maximizes its own self interest, the system behavior will maximize social welfare. Denoting the latency suffered by agent a as l_a, social welfare is defined as $\sum_a l_a \cdot c_a$.[2] The system manager addresses the micro-tolling assignment problem which is defined as follows.

Given: L^i - the vector of links' latencies at time step i.
Output: τ^{i+1} - the vector of tolls applied to each link at the next time step.
Objective: Optimize social welfare.

[2]The tolls are not included in the calculation of social welfare, because we assume that toll revenues are transfer payments which remain internal to society.

Assumption: Agents are self interested i.e., they travel the least cost path $(\arg\min_p\{cost(p,a)\})$ leading to their assigned destination (t_a).

7.3 Enhanced Delta-tolling

We now present the Enhanced Δ-*tolling* mechanism for solving the micro-tolling assignment problem. Enhanced Δ-*tolling* extends the Δ-*tolling* mechanism that is presented in Section 2.4.1. Δ-*tolling* uses two global variables that are used to set tolls on every link in the network. Since different links possess different attributes e.g., capacity, length, speed limit, etc. optimizing the β and R parameters per link can potentially yield greater benefits (higher social welfare, lower total travel time). However, doing so would require optimizing a set of $2|E|$ parameters instead of only two. Optimizing such a high dimensional function cannot be done efficiently in a brute force way.

Enhanced Δ-*tolling* extends Δ-*tolling* by first, considering unique β and R parameters per link and second, incorporating policy gradient RL for optimizing these parameters.

In order to apply policy gradient RL (specifically FD-PGRL, as described in Section 2.4.2), the traffic assignment policy that maps the current state of the traffic to the appropriate actions, which are assigning tolls to each link of the network, should be parameterized. Since the Δ-*tolling* scheme, inherently implemented a policy that takes into account the real-time state of the traffic by assigning tolls proportional to the current links delay, we only use RL policy gradient method to optimize the performance metric at the end of each traffic cycle. Therefore, we define the cost to be the total travel time at the end of each day and consider the following three parametrization of

Δ-*tolling*:

$$\begin{aligned}\pi_R &= [\beta, R_1, \dots, R_n] \\ \pi_\beta &= [R, \beta_1, \dots, \beta_n] \\ \pi_{R,\beta} &= [R_1, \dots, R_n, \beta_1, \dots, \beta_n] \end{aligned} \tag{7.1}$$

The experimental results presented by Sharon et al. [82] suggest that there is some correlation between the optimally performing β and R values. However, no conclusions were presented regarding how they correlate and their individual impact on the convergence rate in a parameter tuning procedure.

As the relation between the β and R parameters remains unclear, we consider three variants of Enhanced Δ-*tolling*based on the parameterized policies listed in (7.1):

EΔ-*tolling*$_\beta$: this variant uses a global R parameter and link specific β parameters ($|E| + 1$ parameters in total). It should perform well under the assumption that there is a correlation between the best performing β and R values and when FD-PGRL estimates the gradient over link specific β parameters more accurately than it does for link specific R parameters.

EΔ-*tolling*$_R$: this variant uses a global β parameter and link specific R parameters ($|E| + 1$ parameters in total). It should perform well under the assumption that there is a correlation between the best performing β and R values and when FD-PGRL estimates the gradient over link specific R parameters more accurately than it does for link specific β parameters.

EΔ-*tolling*$_{\beta,R}$: this variant uses link specific β and R parameters ($2|E|$ parameters in total). It should perform best if there is no correlation between the best performing β and R values and if sufficient computation time is given (converging on $2|E|$ parameters is usually slower than on $|E| + 1$).

7.4 Empirical study

Our experimental evaluation focuses on real-life road networks. Traffic is evaluated using the cell transmission model (CTM) [23, 24] which is a discrete, explicit solution method for the hydrodynamic theory of traffic flow proposed in [58] and [75].

CTM is frequently used in dynamic traffic assignment. The time step used in this model is typically short, on the order of a few seconds. When used with Enhanced Δ*-tolling*, this allows for a truly adaptive toll which can be updated based on observed traffic conditions.

7.4.1 Scenario specification

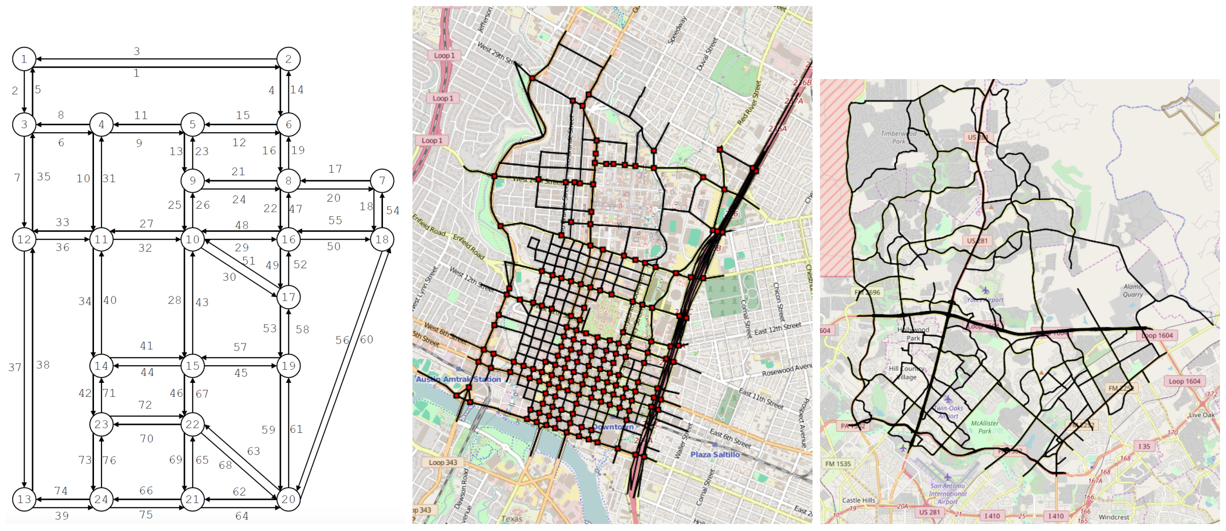
Demand model: demand is given as a trip table, where every entry is affiliated with a single agent (a) and specifies: a source node (s_a), a target node (t_a), and a departure time step (i_a).

Agent model: let l_p^i be the sum of latency along path p during time step i and let τ_p^i be the sum of tolls along p during time step i. When agent a reaches a diverge node n at time step i all paths (P_{nt}) leading from n to destination t_a are considered. Agent a is assigned the minimal cost path i.e., $\arg\min_{p \in P_{nt}} \{\tau_p^i + l_p^i \cdot c_a\}$.

7.4.2 Experiments and results

For running CTM we used the DTA simulator [21] implemented in Java. Whenever a vehicle is loaded onto the network, it is assigned a VOT randomly drawn from a Dagum distribution with parameters $\hat{a} = 22020.6$, $\hat{b} = 2.7926$, and $\hat{c} = 0.2977$, reflecting the distribution of personal income in the United States [35, 61].[3]

[3]The simulation settings were chosen to be identical to those presented in [82].



(a) Sioux Falls (b) Austin (c) San Antonio

Figure 7.1: Maps of traffic networks used in the experiments

The step size in FD-RPGS , η, is 0.4. The policy perturbation parameter, ϵ (see Line 2.2 in Algorithm 2) is set to 0.01 and the number of policy runs at each step, M, is 60 for all the experiments. These values presented best performance overall. Our empirical study focuses on three traffic scenarios:

Sioux Falls: [53] — this scenario is common in the transportation literature [54], and consists of 76 directed links, 24 nodes (intersections) and 28,835 trips spanning 3 hours.

Downtown Austin: [55] — this network consists of 1,247 directed links, 546 nodes and 62,836 trips spanning 2 hours during the morning peak.

Uptown San Antonio: this network consists of 1,259 directed links, 742 nodes and 223,479 trips spanning 3 hour during the morning peak.

The networks affiliated with each scenario are depicted in Figure 7.1. All of these traffic scenarios are available online at: `https://goo.gl/SyvV5m`

	Sioux Falls	Austin	San Antonio
Latency (hr)	11,859	21,590	26,362
cost ($)	353,169	637,086	780,739

Table 7.1: Average total latency and total generalized cost when applying no tolls.

System performance

Our first set of results aims to evaluate the performance of the different variants of Enhanced Δ-*tolling*, by comparing them with each other and basic Δ-*tolling*. Figure 7.2 presents normalized values of total latency summed over all trips (top figure) and social welfare that is the summation of costs, i.e., latency times VOT, over all agents (bottom figure). The values are normalized according to the system's performance when no tolls are applied. Table 7.1 presents the total latency and social welfare performance when applying no-tolls (representing the value of 1.0 in Figure 7.2).

The results present a clear picture in which Δ-*tolling* improves on applying no tolls in both total latency and social welfare. EΔ-*tolling*$_{\beta}$ further improve the system's performance and both EΔ-*tolling*$_{R}$ and EΔ-*tolling*$_{\beta,R}$ achieve the best performance.

The fact that EΔ-*tolling*$_{R}$ results in system performance which is similar to EΔ-*tolling*$_{\beta,R}$ suggests that there is a correlation between the best performing β and R values. The slight superiority of of EΔ-*tolling*$_{R}$ comparing to EΔ-*tolling*$_{\beta,R}$ is due to faster convergence which will be discussed later in this section. The fact that EΔ-*tolling*$_{\beta}$ performs worse than EΔ-*tolling*$_{R}$ suggests that policy FD-PGRL estimates the gradient over link specific R parameters more accurately than it does for link specific β parameters.

Convergence rate

applying EΔ-*tolling* to real-life traffic raises two concerns:

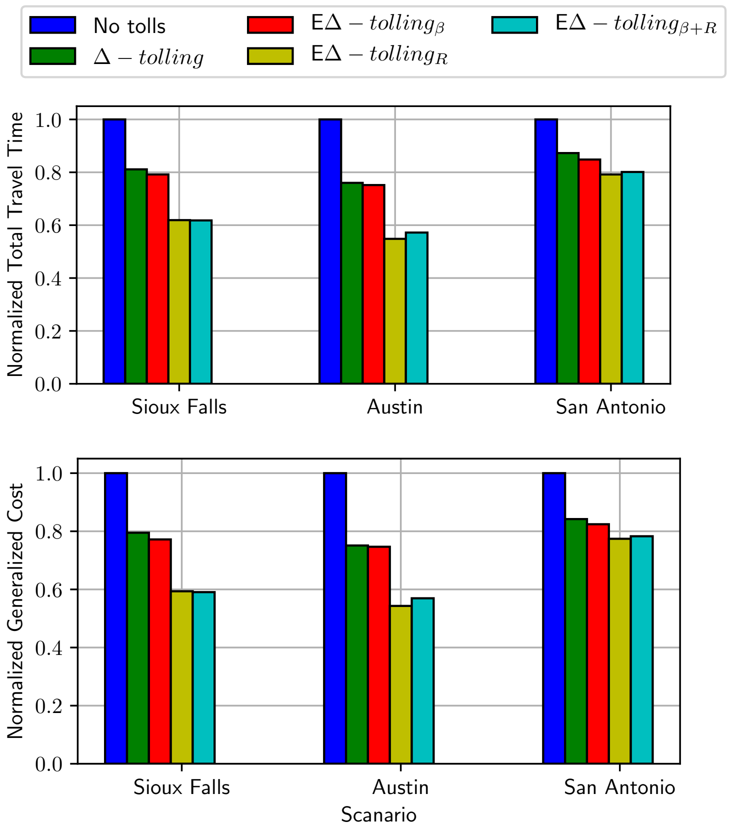


Figure 7.2: Total Travel Time and Total Generalized Cost for different tolling schemes and scenarios.

1. Convergence rate - the system should converge to a good solution with as few learning iterations as possible.
2. Worst case performance - during the learning process EΔ-*tolling* should perform at least as well as Δ-*tolling*.

Figure 7.3 presents the system performance w.r.t total latency (y-axis) versus learning iteration step (x-axis) for each of our three scenarios and every EΔ-*tolling* variant. The error regions are obtained using 10 different runs of the algorithm for each example and EΔ-*tolling* variant and they show the standard error of the average performance in each iteration. Results for basic Δ-*tolling* are also included for comparison. The results are consistent with each other, showing that EΔ-*tolling*$_R$ performs best overall w.r.t convergence rate.

Scheme	S. Falls	Austin	S. Antonio	Total
Δ-*tolling*	962,000	1,640,900	2,300,700	4,903,600
EΔ_β	943,076	1,619,928	2,257,830	4,820,834
EΔ_R	779,990	**1,360,861**	**2,144,502**	**4,285,353**
E$\Delta_{\beta+R}$	**777,469**	1,415,094	2,162,006	4,354,569

Table 7.2: Area under the convergence curves from Figure 7.3.

Table 7.2 presents the area under the curve for each scenario and EΔ-*tolling* variant. These results give a quantitative comparison of the convergence rates. We learn that EΔ-*tolling*$_R$ has the best overall performance with a total AUC of 4,285,353. Nonetheless, EΔ-*tolling*$_{\beta,R}$ performs better on the Sioux Falls scenario. All the experiments are initialized with $\beta = 4$ and $R = 10^{-4}$ for all the links. A set of experiments (not presented) with different starting parameter values show that the performance is sensitive to the initial settings. However, the mentioned default starting values ($\beta = 4$ and $R = 10^{-4}$) perform relatively well across all scenarios and EΔ-*tolling* variants.

7.5 Discussion and Future Work

The promising experimental results reported in Section 7.4 suggest that EΔ-*tolling* can have practical applications where traffic optimization is performed constantly and in real-time through manipulations to the R and or β parameters. Nonetheless, implementation of EΔ-*tolling* raises several practical issues that must first be addressed.

Limitations: EΔ-*tolling* is limited in its convergence rate. General traffic patterns might change frequently, preventing EΔ-*tolling* from advancing in a promising direction. Practitioners must evaluate the convergence rate of EΔ-*tolling* versus the rate in which traffic patterns change in order to determine the applicability of EΔ-*tolling* in a specific network.

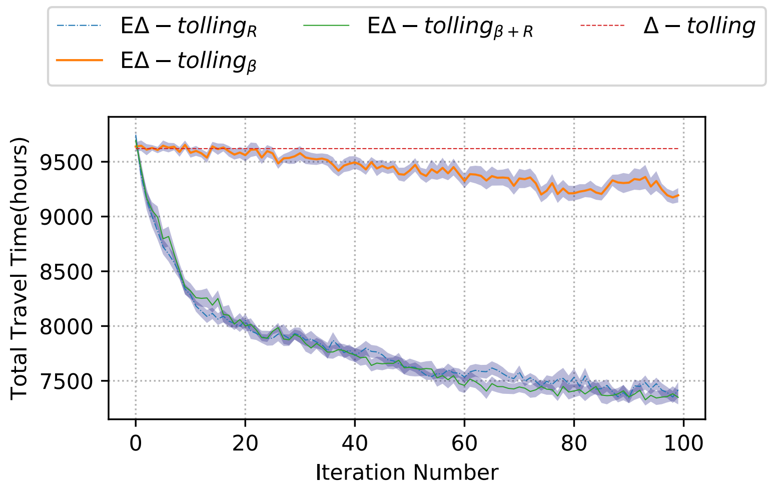
Assumptions: EΔ-*tolling*assumes that all agents traversing the network are self-interested and responsive to tolls in real time. Real world scenarios might violate these assumptions and the trends observed in our results cannot be assumed in such cases.

Practical aspects of EΔ-*tolling* present many promising directions for future work. Since the convergence rate of EΔ-*tolling* plays an important role in determining its applicability, one promising direction for future work is developing heuristics and utilizing advanced RL methods to guide the gradient exploration towards promising directions in order to facilitate faster learning.

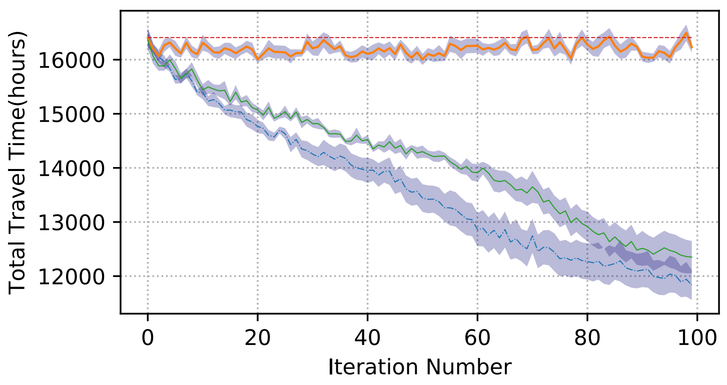
Examining the effects of partial compliance to tolls is another promising direction. Building on recent study that examines the effects of partial compliance on similar micro-tolling schemes [80], studying the practical impacts of partial compliance on EΔ-*tolling* is a promising direction to pursue.

7.6 Conclusion

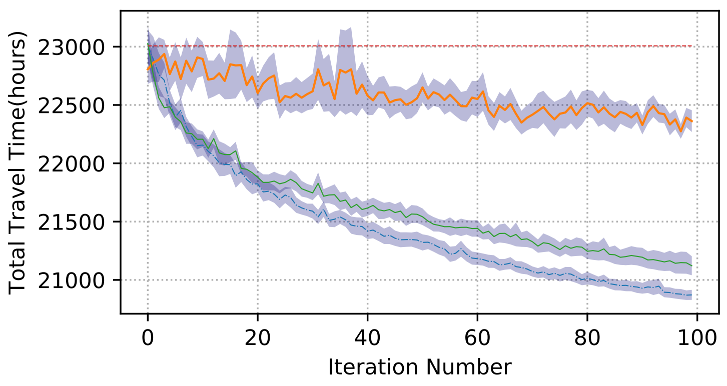
In this chapter, Enhanced Δ-*tolling*, a micro-tolling assignment scheme that builds on the previously suggested Δ-*tolling* scheme, is introduced. The previously suggested Δ-*tolling* scheme makes use of two global parameters, β and R, to tune the system for optimized performance (minimal total latency or maximal social welfare). Enhanced Δ-*tolling* generalizes Δ-*tolling* in two complementary ways. First, recognizing that different links in the network have different attributes (length, capacity, speed limit) Enhanced Δ-*tolling* considers individual β and R parameters per link. Second, given the resulting large parameter set (twice the number of links), Enhanced Δ-*tolling* suggests a policy gradient RL approach for tuning these parameters. Experimental results suggest that tuning the R parameter while keeping a global β parameter performs best overall (w.r.t total latency, social welfare, worst case performance, and convergence rates).



(a) Sioux Falls



(b) Austin



(c) San Antonio

Figure 7.3: System performance w.r.t total latency (y-axis) versus learning iteration step (x-axis) for different scenarios and EΔ-*tolling* variants

Chapter 8

RL Physical Environment Benchmark [1]

8.1 Introduction

Recent advancements in using Artificial Intelligence (AI) to solve continuous-control tasks have shown promise as a replacement for conventional control theory to tackle the challenges in emerging complex Cyber-Physical Systems, such as self-driving control, smart urban transportation and industrial robots. An example of AI approaches is Reinforcement Learning (RL). RL algorithms are mostly model-free, meaning that the explicit modeling of the physical system is not required. Also, RL-based agents can work under uncertainty and adapt to the changing environment or objectives. These unique characteristics of RL make it a good candidate to solve the control problem of complex physical systems. However, the RL solutions for continuous control are in their infancy, since there are limitations when applying them in real-world applications. Some examples are unpredictability of agent actions, lack of formal proofs of closed-loop system stability and not being able to transfer learning from one task to other tasks with slight modifications. This calls for

[1]This chapter is mainly reprinted from: H. Mirzaei, M. Fathollahi and T. Givargis, ”OPEB: Open physical environment benchmark for artificial intelligence,” 2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI), Modena, 2017. IEEE, Copyright (2017), with permission from IEEE.

extensive research to address these limitations and design RL and other AI algorithms that can be used in real-world applications.

While there are a number of widely-used benchmarks in different computing domains, for example MiBench [40] for embedded processing and ImageNet [25] for computer vision, the available AI benchmarks are very limited. This makes conducting research in AI difficult and expensive. Moreover, since there are not many available standard benchmarks, it is hard to evaluate and compare newly proposed AI algorithms. One of the reasons for the lack of AI benchmarks is the interactive nature of dynamical systems. In other words, while it is possible for many other domains to record and label datasets and make them publicly available, AI benchmark developers should provide an interactive "environment" which the AI agent must be able to interact with by applying actions and gathering the new system state (or observation) along with reward signals. This makes AI benchmark development a challenging task. Nevertheless, significant progress has been made recently towards building simulation/emulation based AI benchmarks such as OpenAI Gym and OpenAI Universe [13].

Although the recently developed AI benchmarks enable the researchers to apply their algorithms on a vast variety of different artificial environments, such as PC games or physical systems simulations, real-world physical environments such as industrial robots and self-driving cars are only available to a limited number of groups in big institutes due to the high costs of manufacturing and maintenance of those environments. The lack of physical benchmarks slows down the research progress in developing AI algorithms that can address challenges that usually exist in the real-world such as sensor noise and delay, processing limitations, communicational bandwidth, etc., and can be used in emerging Internet-of-things (IoT) and Cyber-Physical systems.

In this chapter, we propose the Open Physical Environment Benchmark (OPEB) framework to integrate different physical environments. Similar to OpenAI Gym, in our approach a unified interface of the environments is proposed that enables research groups to integrate their physical environment designs to OPEB regardless of the details involved in the hardware/software design and implementation. To achieve the main goals of universality and affordability, we propose leveraging 3D printing technology to build the customized mechanical parts required in the environments and using low-cost generic hardware components such as bolts, ball bearings, etc. We also use popular and affordable embedded processing platforms, such as the Raspberry Pi [89], which is a promising processing solution for IoT and Industry 4.0.

Furthermore, the users are not only able to replicate physical environments using OPEB, but they can also share the implemented environment on the cloud enabling other users to evaluate their algorithms on the actual physical environment. This feature results in higher availability of physical benchmarks and facilitates collaborative research to design robust AI algorithms that can be applied on different realizations of an environment with slight variations in the physical properties of the hardware components. Since OPEB is based on low-cost fabrication solutions, it can be used for educational purposes for IoT, control, AI and other related courses.

The remainder of this chapter is organized as follows. In Section 2.5, we review the background and some related works in AI benchmarks. In Section 8.2, the elements of a physical environment are introduced and it is explained how the required artifacts are provided in OPEB to replicate a physical environment. In Section 8.3, an example implementation of an OPEB, i.e., the classical mountain-car problem, is described, and the results of the experiments that are performed on the physical system using an RL-based method is presented in Section 8.4. Finally, conclusions are presented in Section 8.5.

8.2 Open Physical Environment Benchmark (OPEB)

In this section, we describe our OPEB framework. First, the elements of a physical environment (PE) are introduced and the requirements for each element are discussed. Next, we will explain how the required components to replicate the PE are encapsulated in OPEB and also how the actual implementation can be shared to other users on the cloud.

8.2.1 Physical Environment Elements

The PE consists of the following elements:

- Mechanical parts and structures
- Electromechanical components
- Electrical components
- Embedded processing unit
- Embedded software

To achieve the goal of affordability and universality of PE implementation, the physical parts should include either generic mechanical hardware such as bolts, ball-bearings, etc., or the parts that can be easily printed using a 3D printer. The electromechanical parts such as actuators, dc motors or transducers should be generic parts that can be easily found all over the world. For example, low cost hobby electromechanical parts can be used to build a PE. To drive and interface the electromechanical parts, some electrical parts such as motor drives should be included in the PE. Additionally, to measure the physical quantities, some sensors are required. Examples of such sensors are digital camera, thermometer and proximity sensor.

The embedded processing unit is needed to perform basic required tasks to run the environment such as timing, reading the sensors' outputs and the required signal processing, producing the environment observation, applying the action calculated by the AI algorithm, sending the monitoring data over the network to the monitoring node locally or over the cloud and running the AI algorithm. These tasks are implemented by the embedded software developed for the PE. All of the software components are provided by the OPEB except the AI algorithm which is developed by the PE user.

Emerging single-board embedded computing platforms can be used as the embedded processing unit in PE. Some examples of these solutions are Raspberry Pi [89], C.H.I.P. computer [1] and Arduino [9] platforms. Using a dedicated embedded processor instead of a general purpose computer reduces the cost of deployment of multiple instances of the PE on the cloud and simplifies interfacing the electrical and electromechanical elements because most of these platforms have on-board I/O capabilities.

8.2.2 OPEB Components

In Fig. 8.1, the different components of OPEB for each environment are shown. To realize an environment consisting of the elements listed in the previous subsection, the following components are provided in OPEB for that specific environment:

- Parts that should be 3D printed in STL [39] format.
- List of materials of the generic mechanical hardware.
- Diagrams and instructions required for mechanical structure assembly.
- List of electrical and electromechanical components.
- List of embedded processing units and peripherals.
- Wiring diagram of the electrical components.
- PE control and monitoring Embedded software.
- Web application for the cloud-based sharing of the PE.

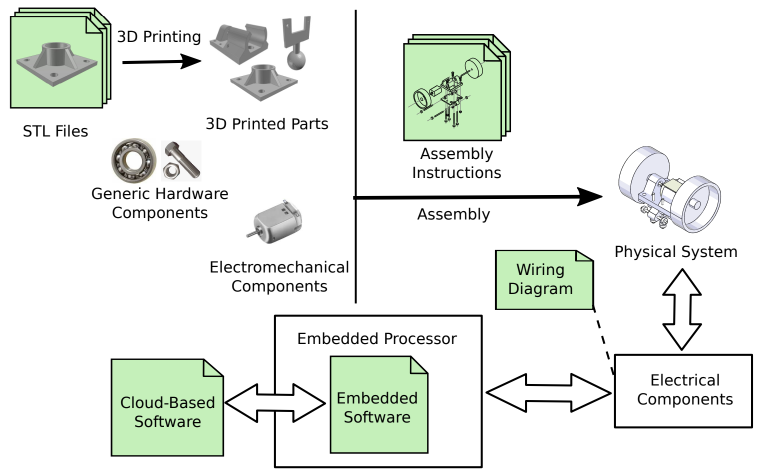


Figure 8.1: OPEB framework components for each environment. Green blocks are provided in OPEB. All other components listed and specified in OPEB.

The customized mechanical parts required by a PE are included in OPEB as 3D models in STL format that can be easily fabricated using a 3D printer. The specifications of other parts that are not printable or can be selected from off-the-shelf products are provided in OPEB. However, these parts are generic mechanical hardwares that are supplied by many manufacturers around the world.

Besides the information provided to obtain or fabricate the components, OPEB includes the complete instructions and diagrams to assemble the mechanical structures of the PE. The main goal of OPEB is that the environments can be reproduced with minimum discrepancy across different implementations. To achieve this goal, the user should be able to build the whole environment using the provided components in the OPEB without ambiguity. On the other hand, the instruction assembly should be of low complexity and easy to follow to be usable by users with different levels of expertise. For this purpose, a step-by-step assembly instruction approach proposed in [2] is employed for the mechanical and electromechanical parts.

Electrical and electromechanical parts, including actuators, sensors, processing units and drivers are usually selected from off-the-shelf products. The list of needed components and their specification are listed in OPEB for each environment. Also, unambiguous wiring diagrams are provided for electrical interconnections.

After building the hardware components, the embedded software should be deployed on the embedded processing unit. The embedded software is included in OPEB and can be deployed using installation manuals. To enable the OPEB users to evaluate their algorithms using different PEs, a standard API is defined similar to OpenAI Gym environments. More specifically, the AI agent can interact with the PE using functions that apply actions and returns the environment observations and reward signal. Furthermore, the environment can be reset to the initial state using the PE API.

Finally, the back-end and front-end software components are provided that enable the OPEB users to deploy their implemented PE over the cloud. Using this web-based application, other users can use the PE to upload and run their AI algorithms on the physical system and see the evaluation reports such as accumulated score over time and record the videos of the PE that runs their algorithm.

8.3 Example implementation: Mountain-Car Example

In this section, we discuss the process of developing an example OPEB environment, i.e., the Mountain-Car example, to demonstrate the methods mentioned in Section 8.2.

In the Mountain Car example, which is first introduced in [68], the goal is to control the acceleration of a car inside a valley in order to move it to the top of the mountain (Fig. 8.2). However, the maximum acceleration of the car is limited and it can not be driven to the top of mountain in a single pass and the car has to go back and forth a number of times to get enough momentum to reach to the desired destination. An AI solution based on Q-learning and tile coding approximation is presented in [86] for this example with a fast convergence in a couple of hundred episodes. However, several simplifying assumptions are made in the original mountain car example including simplified dynamics equations, exact measurements without noise and nonlineariy, no sensor or processing delays and car motion with no friction and no slipping. The last assumption makes

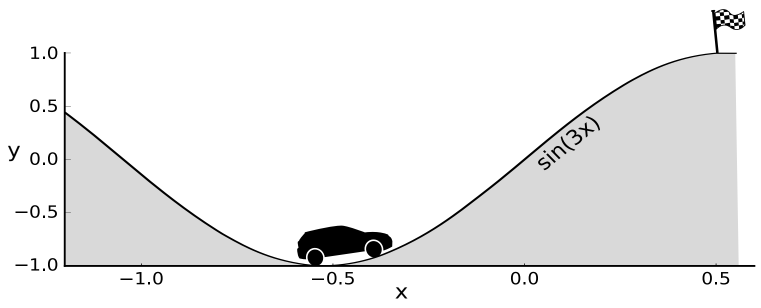


Figure 8.2: Mountain Car example

the learning process a fairly easy task since the kinetic energy delivered by the car's motor is preserved in the system. Consequently, the car can endlessly swing in the valley and the AI agent can make gradual progress towards the goal by increasing the swing range bit-by-bit using successive actions. In a real-world situation, none of these assumptions hold and the agent has to learn a successful policy in a limited time since the car is going to stop after a few swings. The mentioned limitations justify the importance of physical benchmarks that can evaluate the AI algorithms which are useful in real-world applications, for example industrial robotics or self-driving vehicles.

8.3.1 Mechanical Structures

The MC-OPEB consists of two mechanical structures: Car and Mountain rail. The car, which is shown Fig. 8.3, consists of only two large wheels because a car with two pairs of rear and front wheels might entangle around the positions of the path that have low radius of curvature. Also, using only two wheels results in less overall car weight which enables us to use a low power motor and simplifies the design or selection of electrical parts such as motor drive and power supply. Moreover, to prevent the motor from spinning and to constrain the car to move inside the mountain rail, 8 pieces of small ball-bearings are embedded in the car structure using short metal bars.

Each side of the mountain rail, which is shown in Fig. 8.4, is divided to two smaller parts to make them printable using 3D printers with small beds. Additionally, the whole rail surface is not printed

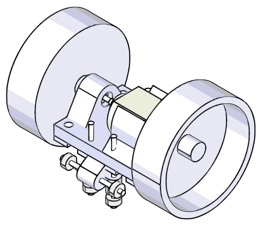


Figure 8.3: Car Assembly in the MC-OPEB.

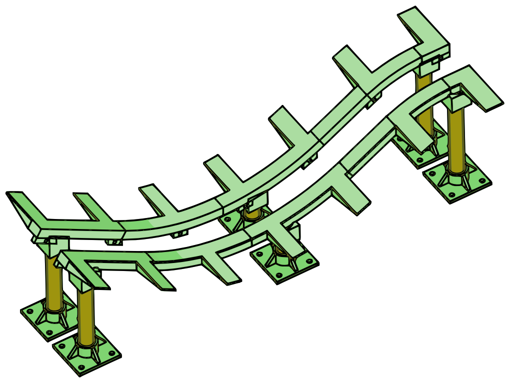


Figure 8.4: Mountain Rail Assembly in the MC-OPEB.

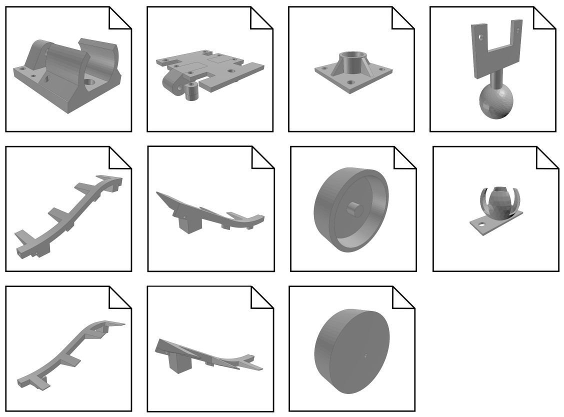


Figure 8.5: STL files included for MC-OPEB for all required 3D printed parts.

to preserve filament. A flexible cardboard should be placed on the support bars attached to the rail structure. The complete STL set of the 3D printed objects are shown in Fig. 8.5 and the set of required hardware is listed in Table 8.1.

An example of assembly instruction documents is provided in Fig. 8.6 which shows the exploded-view diagram of car assembly. The assembly instruction includes the step-by-step action diagrams as explained in [2].

8.3.2 Electromechanical Parts

The only electromechanical part needed for MC-OPEB is the widely-used and low-cost 1.5-3 (V) hobby motor. To reduce the friction and simplify the mechanical design this motor is directly coupled to one of the large wheels on the car. Also, no transducers, such as potentiometer or a shaft encoder is coupled to the motor to reduce the weight of the car and overall cost of MC-OPEB.

8.3.3 Electrical Parts

The required electrical parts are: motor driver, two 5V power supplies for the Raspberry Pi board and driving the motor, and Raspberry pi camera. We have used low-cost HG7881 motor drive with

Table 8.1: List of Materials of required generic hardware parts

Item	Quantity
3mmx10mm bolt and nut	29
32mmx2mm steel bar	7
2x6x2.5mm ball bearing	8
10mmx100mm wooden bar	1

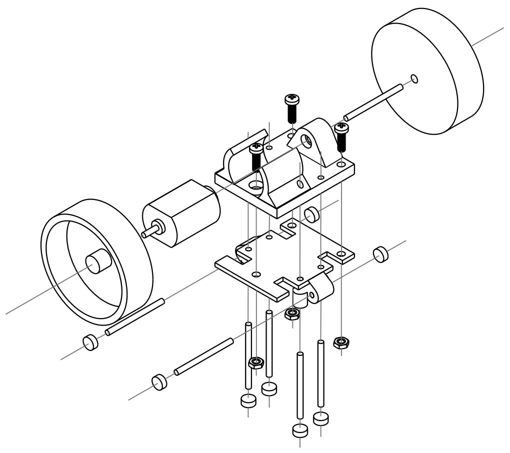


Figure 8.6: Exploded-view of car assembly as an example of assembly instruction diagrams in MC-OPEB.

PWM inputs. Since the Raspberry Pi has two on-board pwm outputs we can directly connect it to the motor drive without any additional interfacing circuit.

The Raspberry Pi camera is used to measure the motion quantities of the car, i.e., position and speed. The captured image of the car also can be used to evaluate emerging deep reinforcement learning algorithms that can control a physical system only by raw visual data.

8.3.4 Embedded Processing Unit

We have used “Raspberry Pi Zero W” platform which is a powerful and affordable processing unit for different embedded applications.

8.3.5 Embedded Software

The Embedded software used in MC-OPEN is a C++ program that is executed on the Raspbian Jessie OS. The embedded software is responsible for implementing the 0.01(s) control timing, capturing and processing the camera image, running the AI routine supplied by the environment user, applying the motor voltage command using PWM outputs, sending monitoring data consisting of instantaneous speed, position and other status variables, running the learned policy and recording the performance video upon user’s request.

The camera image is post processed to calculate the position and speed of the car which are the observations of the MC-OPEB. First, the HSV pixel values are filtered by some fixed thresholds to extract the pixels of the yellow marker attached to the car. Next, the spatial moments of filtered pixels are calculated and used to obtain the single (x, y) coordinate of the car. To reduce the noise and estimate the car’s speed, a linear Kalman filter is implemented in the embedded software.

Algorithm 10: Hand-engineered policy for the mountain-car environment

```
Data: $x, v$                ▷ Instantaneous car position(x) and speed(v)
Result: $a$                 ▷ action(a): acceleration direction
1 if $|v| < 50$ then
2     Choose $a \leftarrow$ left or $a \leftarrow$ right randomly with same probability.;
3 else
4     if $v > 0$ then
5         $a \leftarrow$ left;
6     else
7         $a \leftarrow$ right;
```

8.3.6 Web Application

The web application is an optional component that can be run on a secondary general purpose computer. Using the web application, the MC-OPEB user can see the monitoring data online and share the implemented physical environment on the cloud. The cloud user can upload a c++ routine that implements any custom AI application and evaluate the algorithm performance using the web application. The cloud user can also pause the learning and run the learned policy and see the recorded view of the actual AI algorithm performance.

Fig. 8.7 shows a picture of the actual MC-OPEB. In the next section, we show the results of running a reference algorithm and an RL-based algorithm on the built environment.

8.4 Results

In this section, we present the results of the experiments performed on the MC-OPEB to show the effectiveness of a low-cost PE to perform real-world experiments using AI methods. The objective

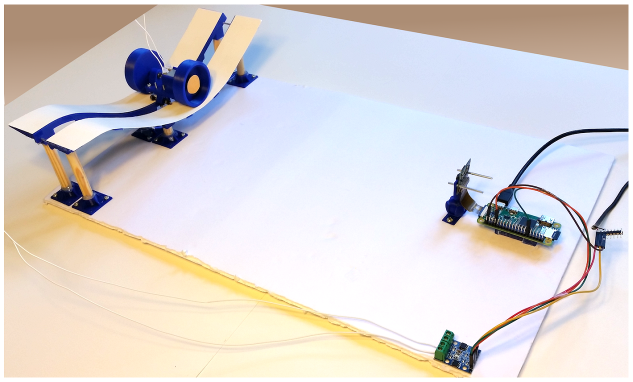


Figure 8.7: Actual Implementation of MC-OPEB.

is to move the car to a certain height on the left side of the rail which corresponds to 80 pixel displacement of the car to the left in the captured image. The reward is defined is as -1 for all the sampling times that the car has not reached the destination. Each episode starts from the car being at the bottom of the valley and ends when it reaches the desired height on the left side. Therefore, the total reward which is the RL "return" is proportional to the negated total episode time. The action is the car's acceleration direction assuming that the car moves with the maximum acceleration and only changes the direction of the acceleration.

8.4.1 Reference Solution

To ensure the possibility of moving the car from the lowest point in the valley to some certain height by any algorithm, a hand-engineered solution is proposed in Algorithm 10. The performance of the AI-based solution can be compared with the reference solution to evaluate the AI algorithm. Fig. 8.8 shows the result of the reference solution.

8.4.2 AI-based solution

The Q-learning algorithm with tile-coding function approximation is used to show that the proposed MC-OPEB can be used to evaluate AI algorithms on a physical environment in real-time.

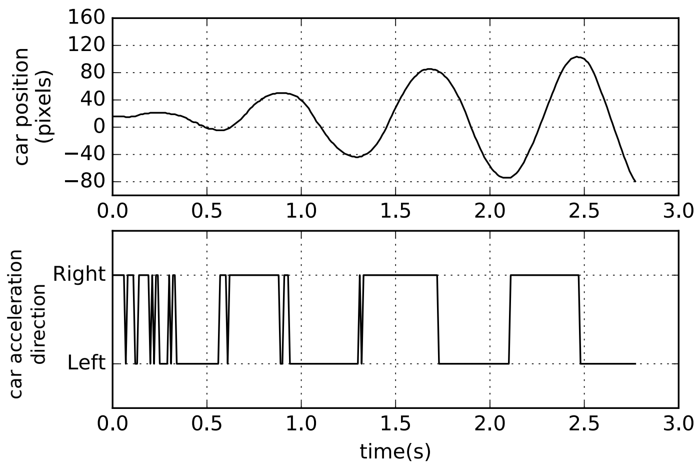


Figure 8.8: The upper plot represents the car position vs. time. The lower plot represents the car acceleration command computed by the hand-engineered algorithm. After a few swings, the designed algorithm is able to move the car to desired height on the left side which corresponds to -80 pixel coordinate of the car in the captured image.

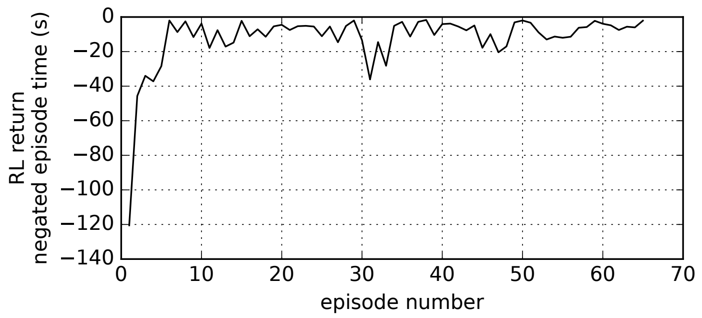


Figure 8.9: Learning curve of the RL agent. x-axis is the episode number and y-axis shows the RL return translated to the total time of each episode. Less absolute value of return means less episode time.

Fig. 8.9 shows the learning curve of the AI agent where the accumulated return vs the episode number is shown. Fig. 8.10 shows the learned policy at episode 37 which is the best performance obtained using the AI algorithm. The results show that the RL algorithm is able to achieve the performance of hand-engineered reference solution. The less number of swings made by the RL agent might be due to slight variations in the physical system and does not necessarily mean the superiority of the RL algorithm.

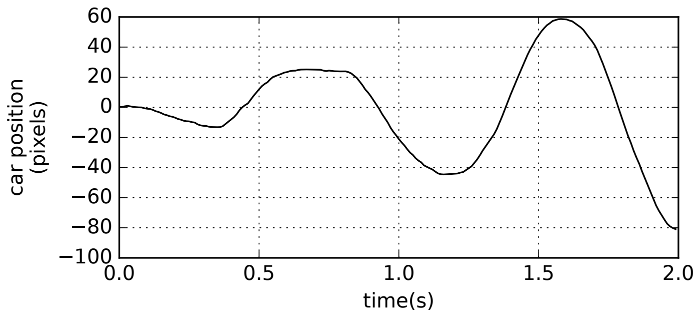


Figure 8.10: Car position vs. time plot obtained by RL algorithm after 37 episodes which its best performance in the experiment.

8.5 Conclusion

In this chapter, a novel physical environment benchmark is presented for AI algorithms. The environments can be implemented using low-cost parts and fabrication methods such as 3D printing. The proposed benchmarks enable researchers to easily replicate physical benchmarks to evaluate their AI algorithms and also share their implemented physical environments on the cloud with other users. Such collaborative benchmarking accelerates development of AI algorithms which can address challenges from real-world physical systems by engaging many researchers that can replicate the physical environments or access them on the cloud. We also presented an example implementation of the proposed physical environment framework. The results show the effectiveness of the proposed methods to develop a simple and low-cost physical benchmark.

Some possible future directions are adding more physical benchmarks, addressing the resource limitations of Raspberry PI for more computationally expensive algorithms and easy deployment of the whole framework on cloud solutions such as Amazon AWS.

Bibliography

[1] C.h.i.p. website. `http://getchip.com/`.

[2] M. Agrawala, D. Phan, J. Heiser, J. Haymaker, J. Klingner, P. Hanrahan, and B. Tversky. Designing effective step-by-step assembly instructions. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 828–837. ACM, 2003.

[3] K. Ahnert and M. Mulansky. Odeint-solving ordinary differential equations in c++. *arXiv preprint arXiv:1110.3397*, 2011.

[4] P. Albertos and A. Crespo. Real-time control of non-uniformly sampled systems. *Control Engineering Practice*, 7(4):445–458, 1999.

[5] P. Albertos and J. Salt. Non-uniform sampled-data control of mimo systems. *Annual Reviews in Control*, 35(1):65–76, 2011.

[6] A. Aminifar, P. Eles, Z. Peng, and A. Cervin. Control-quality driven design of cyber-physical systems with robustness guarantees. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 1093–1098. IEEE, 2013.

[7] M. Amir and T. Givargis. Hybrid state machine model for fast model predictive control: Application to path tracking. In *Proceedings of the 36th International Conference on Computer-Aided Design*, ICCAD '17, pages 185–192. IEEE Press, 2017.

[8] A. Balluchi, P. Murrieri, and A. L. Sangiovanni-Vincentelli. Controller synthesis on non-uniform and uncertain discrete–time domains. In *Hybrid Systems: Computation and Control*, pages 118–133. Springer, 2005.

[9] M. Banzi and M. Shiloh. *Getting Started with Arduino: The Open Source Electronics Prototyping Platform*. Maker Media, Inc., 2014.

[10] A. L. Bazzan. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Autonomous Agents and Multi-Agent Systems*, 18(3):342, 2009.

[11] M. J. Beckmann, C. B. McGuire, and C. B. Winston. *Studies in the Economics of Transportation*. Yale University Press, New Haven, CT, 1956.

[12] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[15] D. Broman. *Meta-Languages and Semantics for Equation-Based Modeling and Simulation*. PhD thesis, Linköping University Electronic Press, 2010.

[16] H. M. Buini, S. Peter, and T. Givargis. Including variability of physical models into the design automation of cyber-physical systems. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6, 2015.

[17] A. Canedo, E. Schwarzenbach, and M. A. Al Faruque. Context-sensitive synthesis of executable functional models of cyber-physical systems. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pages 99–108. ACM, 2013.

[18] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Arzen. How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime. *IEEE control systems*, 23(3):16–30, 2003.

[19] A. Cervin, M. Velasco, P. Martí, and A. Camacho. Optimal online sampling period assignment: Theory and experiments. *Control Systems Technology, IEEE Transactions on*, 19(4):902–910, 2011.

[20] H. Chen, B. An, G. Sharon, J. P. Hanna, P. Stone, C. Miao, and Y. C. Soh. Dyetc: Dynamic electronic toll collection for traffic congestion alleviation. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, February 2018.

[21] Y.-C. Chiu, J. Bottom, M. Mahut, A. Paz, R. Balakrishna, T. Waller, and J. Hicks. Dynamic traffic assignment: A primer. *Transportation Research E-Circular*, (E-C153), 2011.

[22] V. Costa, M. Duarte, T. Rodrigues, S. M. Oliveira, and A. L. Christensen. Design and development of an inexpensive aquatic swarm robotics system. In *OCEANS 2016-Shanghai*, pages 1–7. IEEE, 2016.

[23] C. F. Daganzo. The cell transmission model: a dynamic representation of highway traffic consistent with the hydrodynamic theory. *Transportation Research Part B*, 28(4):269–287, 1994.

[24] C. F. Daganzo. The cell transmission model, part II: network traffic. *Transportation Research Part B*, 29(2):79–93, 1995.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[26] P. Derler, E. A. Lee, S. Tripakis, and M. Törngren. Cyber-physical system design contracts. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pages 109–118. ACM, 2013.

[27] R. B. Dial. Minimal-revenue congestion pricing part I: A fast algorithm for the single-origin case. *Transportation Research Part B*, 33:189–202, 1999.

[28] K. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*, 31:591–656, 2008.

[29] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. *arXiv preprint arXiv:1604.06778*, 2016.

[30] S. Durand, J. F. G. Castellanos, N. Marchand, and W. F. G. Sanchez. Event-based control of the inverted pendulum: Swing up and stabilization. *Journal of Control Engineering and Applied Informatics*, 15(3):96–104, 2013.

[31] S. El Bsat, H. Bou-Ammar, and M. E. Taylor. Scalable multitask policy gradient reinforcement learning. In *AAAI*, pages 1847–1853, 2017.

[32] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad. Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): methodology and large-scale application on downtown toronto. *Intelligent Transportation Systems, IEEE Transactions on*, 14(3):1140–1150, 2013.

[33] C. Frese and J. Beyerer. A comparison of motion planning algorithms for cooperative collision avoidance of multiple cognitive automobiles. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 1156–1162. IEEE, 2011.

[34] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nystrom, L. Saldamli, D. Broman, and A. Sandholm. Openmodelica-a free open-source environment for system modeling, simulation, and teaching. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pages 1588–1595. IEEE, 2006.

[35] L. M. Gardner, H. Bar-Gera, and S. D. Boyles. Development and comparison of choice models and tolling schemes for high-occupancy/toll (hot) facilities. *Transportation Research Part B: Methodological*, 55:142–153, 2013.

[36] A. Geramifard, C. Dann, R. H. Klein, W. Dabney, and J. P. How. Rlpy: a value-function-based reinforcement learning framework for education and research. *Journal of Machine Learning Research*, 16:1573–1578, 2015.

[37] J. Ginsberg. *Engineering dynamics*, volume 10. Cambridge University Press, 2008.

[38] D. Goswami, R. Schneider, and S. Chakraborty. Co-design of cyber-physical systems via controllers with flexible delay constraints. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pages 225–230. IEEE Press, 2011.

[39] T. Grimm. *User's guide to rapid prototyping*. Society of Manufacturing Engineers, 2004.

[40] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14. IEEE, 2001.

[41] M. Hausknecht, T.-C. Au, and P. Stone. Autonomous intersection management: Multi-intersection optimization. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4581–4586. IEEE, 2011.

[42] D. Henriksson and A. Cervin. Optimal on-line sampling period assignment for real-time control tasks based on plant state information. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 4469–4474. IEEE, 2005.

[43] M. Hunting. The aimms outer approximation algorithm for minlp. *Paragon Decision Technology, Haarlem*, 2011.

[44] J. C. Jensen, D. H. Chang, and E. A. Lee. A model-based design methodology for cyber-physical systems. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1666–1671. IEEE, 2011.

[45] D.-C. Juan, S. Garg, J. Park, and D. Marculescu. Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2013 International Conference on*, pages 1–10. IEEE, 2013.

[46] E. C. Kara, M. Berges, B. Krogh, and S. Kar. Using smart devices for system-level management and control in the smart grid: A reinforcement learning framework. In *Smart Grid Communications (SmartGridComm), 2012 IEEE Third International Conference on*, pages 85–90. IEEE, 2012.

[47] B. Karlik and A. V. Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[48] K. Katsuki. Machine learning system and motor control system having function of automatically adjusting parameter, Mar. 30 2017. US Patent 20,170,090,434.

[49] S. Khan, R. M. Goodall, and R. Dixon. Non-uniform sampling strategies for digital control. *International Journal of Systems Science*, 44(12):2234–2254, 2013.

[50] S. G. Khan, G. Herrmann, F. L. Lewis, T. Pipe, and C. Melhuish. Reinforcement learning and optimal adaptive control: An overview and implementation examples. *Annual Reviews in Control*, 36(1):42–59, 2012.

[51] D. E. Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2012.

[52] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation, ICRA 2004, April 26 - May 1, 2004, New Orleans, LA, USA*, pages 2619–2624, 2004.

[53] L. J. LeBlanc, E. K. Morlok, and W. P. Pierskalla. An efficient approach to solving the road network equilibrium traffic assignment problem. *Transportation Research*, 9(5):309–318, 1975.

[54] M. W. Levin and S. D. Boyles. Intersection auctions and reservation-based control in dynamic traffic assignment. In *Transportation Research Board 94th Annual Meeting*, number 15-2149, 2015.

[55] M. W. Levin, M. Pool, T. Owens, N. R. Juri, and S. T. Waller. Improving the convergence of simulation-based dynamic traffic assignment methodologies. *Networks and Spatial Economics*, 15(3):655–676, 2015.

[56] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[57] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *arXiv preprint arXiv:1603.02199*, 2016.

[58] M. J. Lighthill and G. B. Whitham. On kinematic waves. ii. a theory of traffic flow on long crowded roads. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 229, pages 317–345. The Royal Society, 1955.

[59] 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.

[60] S. Lu. Sensitivity of static traffic user equilibria with perturbations in arc cost function and travel demand. *Transportation Science*, 42(1):105–123, 2008.

[61] P. Lukasiewicza, K. Karpioa, and A. Orlowskia. The models of personal incomes in USA. In *Proceedings of the 5th Symposium on Physics in Economics and Social Sciences*, Warsaw, Poland, 2012.

[62] M. Maasoumy, Q. Zhu, C. Li, F. Meggers, and A. Sangiovanni-Vincentelli. Co-design of control algorithm and embedded platform for building hvac systems. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pages 61–70. ACM, 2013.

[63] A. A. Malikopoulos, C. G. Cassandras, and Y. J. Zhang. A decentralized optimal control framework for connected and automated vehicles at urban intersections. *arXiv preprint arXiv:1602.03786*, 2016.

[64] N. Marchand, S. Durand, and J. F. G. Castellanos. A general formula for the stabilization of event-based controlled systems. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 8199–8204. IEEE, 2011.

[65] J. Mattner, S. Lange, and M. Riedmiller. Learn to swing up and balance a real pole based on raw visual input data. In *International Conference on Neural Information Processing*, pages 126–133. Springer, 2012.

[66] M. N. Mladenović and M. M. Abbas. Self-organizing control framework for driverless vehicles. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 2076–2081. IEEE, 2013.

[67] A. W. Moore. Efficient memory-based learning for robot control. Technical report, 1990.

[68] A. W. Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, Computer Laboratory, 1990.

[69] N. Mühleis, M. Glaß, L. Zhang, and J. Teich. A co-simulation approach for control performance analysis during design space exploration of cyber-physical systems. *ACM SIGBED Review*, 8(2):23–26, 2011.

[70] N. Navarro-Guerrero, C. Weber, P. Schroeter, and S. Wermter. Real-world reinforcement learning for autonomous humanoid robot docking. *Robotics and Autonomous Systems*, 60(11):1400–1407, 2012.

[71] H. Neema, Z. Lattmann, P. Meijer, J. Klingler, S. Neema, T. Bapty, J. Sztipanovits, and G. Karsai. Design space exploration and manipulation for cyber physical systems. In *Workshop on Design Space Exploration of Cyber-Physical Systems (IDEAL)*, 2014.

[72] M. Pecka, K. Zimmermann, M. Reinstein, and T. Svoboda. Controlling robot morphology from incomplete measurements. *IEEE Transactions on Industrial Electronics*, 64(2):1773–1782, 2017.

[73] J. Peters and S. Schaal. Policy gradient methods for robotics. In *IROS*, pages 2219–2225. IEEE, 2006.

[74] A. C. Pigou. *The Economics of Welfare*. Palgrave Macmillan, 1920.

[75] P. Richards. Shock waves on the highway. *Operations Research*, 4(1):42–51, 1956.

[76] T. A. Roughgarden. *Selfish Routing*. PhD thesis, Cornell University, 2002.

[77] A. Sala. Computer control under time-varying sampling period: An lmi gridding approach. *Automatica*, 41(12):2077–2082, 2005.

[78] T. Schouwenaars, B. De Moor, E. Feron, and J. How. Mixed integer programming for multi-vehicle path planning. In *Control Conference (ECC), 2001 European*, pages 2603–2608. IEEE, 2001.

[79] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. *CoRR, abs/1502.05477*, 2015.

[80] G. Sharon, M. Albert, S. B. Tarun Rambha, and P. Stone. Traffic optimization for a mixture of self-interested and compliant agents. In *32th AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018.

[81] G. Sharon, J. P. Hanna, T. Rambha, M. W. Levin, M. Albert, S. D. Boyles, and P. Stone. Real-time adaptive tolling scheme for optimized social welfare in traffic networks. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2017)*, May 2017.

[82] G. Sharon, M. W. Levin, J. P. Hanna, T. Rambha, S. D. Boyles, and P. Stone. Network-wide adaptive tolling for connected and automated vehicles. *Transportation Research Part C*, 84:142–157, September 2017.

[83] Y. Sheffi. Urban transportation network. *Equilibrium analysis with mathematical programming methods, Prentice Hall*, 1985.

[84] R. Sheh, H. Komsuoglu, and A. Jacoff. The open academic robot kit: Lowering the barrier of entry for research into response robotics. In *Safety, Security, and Rescue Robotics (SSRR), 2014 IEEE International Symposium on*, pages 1–6. IEEE, 2014.

[85] D. Simon, D. Robert, and O. Sename. Robust control/scheduling co-design: application to robot control. In *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, pages 118–127, 2005.

[86] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[87] B. Tanner and A. White. Rl-glue: Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10(Sep):2133–2136, 2009.

[88] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 4, pages 101–104. IEEE, 2000.

[89] E. Upton and G. Halfacree. *Raspberry Pi user guide*. John Wiley & Sons, 2014.

[90] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[91] H. Yang, Q. Meng, and D.-H. Lee. Trial-and-error implementation of marginal-cost pricing on networks in the absence of demand functions. *Transportation Research Part B: Methodological*, 38(6):477–493, 2004.

[92] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney. Task scheduling for control oriented requirements for cyber-physical systems. In *Real-Time Systems Symposium, 2008*, pages 47–56. IEEE, 2008.

[93] Y. Zhu, E. Westbrook, J. Inoue, A. Chapoutot, C. Salama, M. Peralta, T. Martin, W. Taha, M. O'Malley, R. Cartwright, et al. Mathematical equations as executable models of mechanical systems. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, pages 1–11. ACM, 2010.

[94] I. H. Zohdy, R. K. Kamalanathsharma, and H. Rakha. Intersection management for autonomous vehicles using icacc. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 1109–1114. IEEE, 2012.