**Title**
Reasoning for Representations for Learning-based Control



**Author**
Xu, Zhuo

**Publication Date**
2021

Peer reviewed|Thesis/dissertation

# Reasoning for Representations for Learning-based Control

by

Zhuo Xu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering- Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Masayoshi Tomizuka, Chair
Professor Pieter Abbeel
Professor Mark W. Mueller

Summer 2021

Reasoning for Representations for Learning-based Control

Abstract

Reasoning for Representations for Learning-based Control

by

Zhuo Xu

in Engineering- Mechanical Engineering

University of California, Berkeley

Professor Masayoshi Tomizuka, Chair

Data-driven machine learning approaches offer great intelligence capabilities and can help solve many challenging control problems. However, the agnostic nature of the internal representations in the learning-based control (LbC) policies makes them difficult to apply on a broad basis. Since the LbC policies are generally optimized in an end-to-end manner based on specialized domain settings, they can fail, for unknown reasons, in domains with setting variations. Moreover, a lack of understanding of the underlying logics in LbC policies also limits the transferability of the learned knowledge to different tasks.

This dissertation presents a series of works on reasoning for representations for LbC policies, including decomposition of LbC policies, and design of interpretable and transferable representations. The thread of this dissertation is based on the roles of the representations for LbC. Three major areas are covered in this dissertation. Three major representation reasoning areas are covered in this dissertation:: (I) inside the LbC policies, (II) at the interface of the LbC policies, and (III) outside the LbC policies.

First, within the parameterized LbC policies, the reasoning for learning representations is developed to capture subtle features in complex scenarios. A sophisticated neural network structure is applied online to infer the task context representations in a contact-aware way; this is followed by a comprehensive investigation on design and selection of representations. The presented representations include the Gaussian mixture model, the graph neural network, and a novel history-encoding representation; the applications range from robotic manipulation to autonomous driving to human intention inference.

In regard to the challenge of knowledge transfer, we propose a policy decomposition approach to learn attribute-wise modules separately. We design two representation frameworks at the interface of the LbC policies for the decomposition and combination of attribute-wise modules. The proposed architecture, the cascade attribute networks (CANs), and parallel attribute networks (PANs) can transfer learned knowledge between tasks and efficiently

produce sophisticated LbC policies by fusing learned attribute-wise modules.

Variation between the training and deployment domains is another major reason there are LbC policy failures. In the last part of the dissertation, we focus our investigation on representation reasoning for autonomous-driving policy transfer against vehicle dynamics variation and external dynamics disturbances. We leverage the interpretable kinematic-level representations to bridge the transferring domains. We propose two external adaptation modules—model agnostic meta learning (MAML) and disturbance-observer-based (DOB) robust controllers— to achieve one-shot or zero-shot adaptation of the kinematic-level representations to the deployment domain.

To my family

# Contents

# List of Figures

# List of Tables

## Acknowledgments

The past five years at Berkeley have been a precious and memorable journey in my life. I would never have come this far without the tremendous help from so many awesome people.

First and foremost, I would like to express my deepest and earnest gratitude to my Ph.D. advisor, Prof. Masayoshi Tomizuka, who has greatly shaped my life and career with his tremendous mentorship and support. His profound knowledge, insightful visions, enthusiasm and dedication in research are going to influence me till forever. Prof. Tomizuka respects all my immature ideas, inspires me to explore the unknowns, and is always responsive to any of my questions or troubles. Without his guidance and patient help, this dissertation would not have been possible. I sincerely wish I could be such a great person and extraordinary mentor like him in the future.

I am very grateful to the faculty who helped me during my Ph.D. and gave me invaluable advice on my research. I am deeply thankful to my dissertation committee members, Prof. Pieter Abbeel and Prof. Mark Mueller, and qualifying examination committee members, Chair and Prof. Roberto Horowitz, Prof. Mark Mueller, Prof. Jonathan Shewchuk, and Prof. Javad Lavaei, and other Berkeley faculty Prof. Sergey Levine, Prof. Kameshwar Poolla. Thank you so much for all the insightful advice and valuable discussions! I am also grateful to Prof. Karen Liu from Stanford, who was the faculty advisor during my AI residency at Google X, Professor Micheal Yu Wang and Professor Ming Liu from HKUST, who were the faculty advisors of the UCB-HKUST collaboration project.

Special thanks go to Berkeley's William C. Webster Graduate Fellowship, and sponsors for the works in this dissertation: the Berkeley Deep Drive Consortium, Lens Technology and HAI Robotics from China, and the Innovation and Technology Fund, Hong Kong. I would like to thank my internship mentors and colleagues, Dr. Yunfei Bai, Daniel Ho, Dr. Wenhao Yu, Dr. Alexander Herzog, Dr. Wenlong Lu, Dr. Chuyuan Fu from Google, as well as Dr. Yizhou Wang from NVIDIA, who is also my lab alumni. I am grateful to Mr. Yu Cheng, and Peter Zhe Chen from 5Y Capital. Their consistent help provided me with rich experience and insight in the industry.

It is a great honor of mine to be a member of the Mechanical System Control (MSC) Laboratory, where I gained precious friendship and uncountable research inspirations. I received tremendous help from former MSC members: Prof. Jianyu Chen, Dr. Zining Wang, Dr. Liting Sun, Prof. Changliu Liu, Prof. Cong Wang, who have provided so many precious suggestions to me in life and career. I sincerely thank my fellow MSC students Chen Tang, Jiachen Li, Huidong Gao for all inspiring discussions, postdoc Dr. Wei Zhan for the guidance and advice, and Jianhao Jiao from HKUST for the discussions and help. I was also inspired by many current and past colleagues in the MSC lab and thank you all: Dr. Yeping Hu, Yujiao Cheng, Shiyu Jin, Dr. Wenjie Chen, Prof. Minghui Zheng, Dr. Shiying Zhou, Dr. Hsien-Chung Lin, Dr. Te Tang, Dr. Yu Zhao, Dr. Xiaowen Yu, Dr. Chen-Yu Chan, Dr. Yongxiang Fan, Dr. Yu-Chu Huang, Dr. Shuyang Li, Dr. Cheng Peng, Dr. Daisuke Kaneishi, Dr. Kiwoo Shin, Dr. Long Xin, Hengbo Ma, Jessica Leu, Xinghao Zhu, Changhao Wang, Lingfeng Sun, Yiyang Zhou, Ge Zhang, Zheng Wu, Jinning Li, Xiang Zhang, Wu-Te

Yang, Saman Fahandezhsaadi, Catherine Faulkner, Akio Kodaira, Chenran Li, Ran Tian, and Chenfeng Xu. I also thank the interns I had the fortune to advise: Haonan Chang, Hongyu Zhou, Rui Zhou, and Hanxiao Chen. I wish all the best in your future endeavors!

In addition, I would like to thank all my friends at Berkeley for the wonderful time that we spent together, in particular, Prof. Yang Gao, Dr. Biye Jiang, Zhengliang Su, James Li, Qianyi Xie, Qing Tian, Huazhe Xu, David Ren, Shouping Chen, and happy wedding to An Ju and Hongyu Zhang! I would also like to thank my friends at the Association of Chinese Entrepreneurs (ACE), Dr. Bodi Yuan, Dr. Teng Lei, Quinn Sure, Coco Chen, Nikki Shen, Leona Lan, and everyone.

Last but not least, my deepest love goes to my parents, Y. Xu and C. Yu, who have given me endless love, protection, and support. I am fortunate to be your son, I hope you are proud of me. Many thanks to my lovely and always joyful Magie, for your unconditional love, support and encouragement throughout my PhD. I wish you all the best pursuing your Ph.D. at Harvard Business School. I thank my uncles, aunts, and cousins who have taken care of me. Thank Charlie and Chuhan, with you in the Bay Area, I feel the warmth at home, and my younger cousin Hongtao, I hope you successfully obtain your doctorate from PUMC, Tsinghua University. Finally, my best wishes go to new members of my family, Hanlin, Wanyuan, Juncai, and Manci.

# Chapter 1

# Introduction

## 1.1 Challenges in Learning-based Control (LbC)

In recent years, advances in artificial intelligence technology and increasingly accessible and affordable computation and data resources have brought groundbreaking progress to the field of robotics. From playing StarCraft [110] to mastering the game of Go [99] and from intelligent robotic manipulation [52] to driving autonomous cars [58], these once impossible dreams are coming true. This is thanks to the intelligence capability that is obtained through deep learning and reinforcement learning algorithms and embedded in learning-based control (LbC) policies. It is a common hope of humankind that LbC policies can take on more intelligent capabilities and replace human effort in more complicated and challenging tasks.

However, there are still lots of challenges limiting the intelligent and general-purpose LbC future we envision. For example, sophisticated deep neural networks are powerful nonlinear function approximators, but as LbC policies, they suffer from the problem of internal meaning agnosticism. Given that LbC policies are usually optimized using gradient-descent-based methods in an end-to-end manner in specialized training settings, their application in robotics and control faces the following challenges:

1. Adapt across domains: Learned LbC policies are specialized to the training domain setting. Therefore, when deployed to a target domain different from the training domain, or when affected by unexpected disturbances, LbC policies often fail to achieve the expected performance.

2. Transfer knowledge between tasks: Since the neural networks use a large number of neurons and sophisticated nonlinear activation functions to achieve general function approximation, it is hard to understand the information carried by a LbC policy. Therefore, it is also challenging to transfer the knowledge learned in one task to a similar task.

3. Obtain generalizable skills: Due to the agnostics of the internal meanings of the neural networks, the LbC policies are unable to capture the properties of the tasks and

offer generalizable skills for various task settings. They tend to produce conservative behavior derived from the statistical learning process.

## 1.2 Representation Reasoning for LbC

In the past few years, the research community has devoted a significant amount of effort to adapt an LbC policy to different task settings, variation of dynamics, and unknown disturbances. A common practice is to train policies on randomized environments with possible variations to gain robustness. Rajeswaran et al. [86] and Peng et al. [81] studied reinforcement learning (RL) policies in randomized simulated environments and directly applied them to different domains. Chebotar et al. leveraged target domain experiences to adapt the source domain setting [22]. Other works aim to derive a model of the task context for domain-specific planning. Rakelly et al. take a meta learning direction to learn patterns within the task to help plan [87]. Yu et al. studied a universal policy and used an online system identification module to understand the dynamics parameters [126]. [42] used a neural network for a dynamics model and adapted its local linear model online for model-based control. In [28], a deep inverse dynamics model was used for policy transfer. Beyond pure learning-based methods, planning-control frameworks can also help overcome the domain gap. Harrison et al. used model predictive control to track a reference trajectory derived from a learning policy [47].

Based on the philosophy of prior work, this dissertation proposes that the reasoning of the representations for the LbC policies is at the core of solving the proposed challenges of their application. By decomposing the end-to-end LbC policies into separate interpretable modules and designing proper representations, we can design the LbC policies with favorable behaviors, such as domain adaptation, knowledge transfer, and robustness to general settings. Such representations can have various formulations and reside in different positions in the control loop, based on domain requirements (as shown in Fig. 1.1). They can be an extra neural layer within an LbC policy or be a concrete feature that bridges the learning-based and non-learning based modules. The selection, design, and reasoning of the representations in LbC policies are the most challenging and, thus, most intriguing parts of this problem.

## 1.3 Dissertation Outline

In this dissertation, we present a series of works on the reasoning of representations for LbC policies. The dissertation follows the thread of the function-reasoned representation in the control policy and is divided into three parts: reasoning for representations (I) inside the control policy, (II) at the input and output of the policy, and (III) outside the policy.

In Part I, we present works on reasoning for representations inside the LbC policies. The representations we discuss in this part are feature layers in parametrized models, which serve as information extractors or encoders. We carefully select and design proper represen-

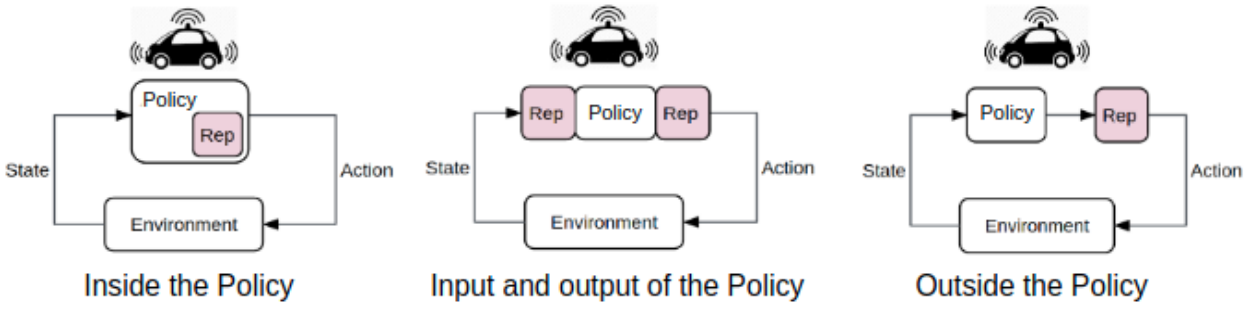


Figure 1.1: The three kinds of representations this thesis investigates are: the representation inside the control policy, the representation at the input and output of the control policy, and the representation outside the control policy.

tations so as to obtain a better understanding of the task scenario and improve the learning efficiency. We examine a typical challenge of nonplanar pushing manipulation in Chapter 2, where the robot aims to push the object while avoiding knocking it over. In order to achieve generalizable pushing of different objects with various dynamics properties, we design a contact-aware online context inference module inside the LbC policy to understand the dynamics representation. In Chapter 3, we present a series of works on various representations selection and design, including Gaussian mixture model, graph neural networks, and a novel history encoding representation. Their applications range from autonomous driving to human intention prediction.

In Part II, we present works on decomposing LbC policies to modules in terms of task attributes. The decomposition is designed such that attribute-wise knowledge can be learned separately for better transfer. We reason for LbC policy interface representations, which serve as the input and output of the attribute modules. We present two different designs of representations and attribute combination frameworks. In Chapter 4, we present the cascade attribute network (CAN), where an attribute module's output is fed as part of the input of the next attribute module and the last attribute module output is the overall LbC policy output. In Chapter 5, we present the parallel attribute network (PAN), where different attribute modules produce outputs in parallel, and their outputs are combined using an online optimization process. Experiments show that PANs achieve good performance on complicated tasks with multiple attributes.

In Part III, we present efforts to design domain-invariant representations outside the LbC policies. The problem setting is focused on domain transfer of autonomous-driving LbC policy against vehicle dynamics variation and unexpected external disturbances. We propose that the kinematic-level representations, unlike the dynamics-level representations, are domain invariant and, thus, selected for bridging different domains. In Chapter 6, we present an LbC policy transfer framework, with the inverse dynamics model serving as the domain-specific representation. We apply meta learning for one-shot adaptation. In Chapter 7, the kinematic-level representations are directly tracked using a disturbance-observer-based

(DOB) robust controller that rejects the domain variations in a zero-shot manner. The DOB controller has been proven successful, with the LbC policies learned on a simulated vehicle successfully transferred to a real vehicle, a Lincoln MKZ.

# Part I

# Reasoning for Representations Inside the LbC Policies

# Chapter 2

# Contact-aware Online Context Inference for Generalizable Non-Planar Pushing Policy Learning

General contact-rich manipulation problems are long-standing challenges in robotics due to the difficulty of understanding complicated contact physics. Deep reinforcement learning (RL) has shown great potential in solving robot manipulation tasks. However, existing RL policies have limited adaptability to environments with diverse dynamics properties, which is pivotal in solving many contact-rich manipulation tasks. In this chapter, we propose Contact-aware Online COntext Inference (COCOI), a deep RL method that encodes a context embedding of dynamics properties online using contact-rich interactions [121]. We sample sensor data using a novel contact-aware strategy and formulate an interpretable dynamics transition module. We study this method based on a novel and challenging non-planar pushing task, where the robot uses a monocular camera image and wrist force torque sensor reading to push an object to a goal location while keeping it upright. We run extensive experiments to demonstrate the capability of COCOI in a wide range of settings and dynamics properties in simulation, and also in a sim-to-real transfer scenario on a real robot (Webpage: `https://context-inference.github.io/`).

## 2.1 Introduction

Contact rich manipulation problems are ubiquitous in the physical world. In millions of years of evolution, humans have developed the remarkable capability to understand environment physics, so as to achieve general contact rich manipulation skills. Combining visual and tactile perception with end-effectors like fingers and palms, humans effortlessly manipulate objects with various shapes and dynamics properties in complex environments. Robots, on the other hand, lack this capability – due to the difficulty of understanding high dimensional perception and complicated contact physics. Recent development in deep rein-

forcement learning (RL) has shown great potential towards solving manipulation problems [63, 52, 59] by leveraging two key advantages. First, the representative capability of a deep neural network structure can capture complicated dynamics models. Second, control policy optimization explores vast contact interactions. However, contact-rich manipulation tasks are generally dynamics-dependent; since the RL policies are trained in a specific dynamics setting, they specialize within the training scenario and are vulnerable to variations of dynamics. Learning a policy that is robust to dynamics variations is pivotal for deployment to scenarios with diverse object dynamics properties.

In this chapter, we design a deep RL method that takes multi-modal perception input and uses deep representative structure to capture contact-rich dynamics properties. Our method, which we refer to as Contact-aware Online COntext Inference (COCOI), improves planning by inferring the system dynamics from prior camera frames and force readings. Furthermore, we propose to sample the sensor data using a contact-aware strategy, and formulate the data into a dynamics transition structure, which serves as the input of the COCOI module. With this design, the contact dynamics information can be effectively encoded into a context representation with interpretability. This allows the RL policy to plan with dynamics-awareness and improves in robustness against domain variations.

We apply COCOI to a novel pushing task where dynamics property reasoning plays a vital role: the robot needs to push an object to a target location while avoiding knocking it over (Fig. 2.1). Despite being commonly seen in everyday life, these tasks have the following challenges:

1. Visual perception: unlike in planar pushing, where concrete features can be retrieved from a top down view, in non-planar pushing, key information can not be easily extracted from the third angle perspective image.

2. Contact-rich dynamics: the task dynamics properties are not directly observable from raw sensor information. Furthermore, in our non-planar pushing task, dynamics property reasoning is vital to avoid knocking the object over.

3. Generalizable across domain variations: the policy needs to be effective for objects with different appearances, shapes, masses, and friction properties.

Prior work in pushing mostly focus on objects inherently stable when pushed on a flat surface[102]. This essentially reduces the task to a 2D planar problem. Concretely, earlier studies on pushing by Mason are based on analytical approaches and consider quasi-static planar pushing [70]. Later, researchers have introduced data driven methods to model pushing physics. Zhou et al. develop a dynamics model for planar friction and design force control method for planar sliding [128]. Yu et al. [125] and Bauza et al. [11] create a planar pushing dataset for data-driven modeling. More recent works involve using deep learning to learn object properties [65, 2, 115], but they only focus on planar pushing problems. Stuber et al. [101] and Byravan et al. [20] learn motion models for pushing simple blocks. As a result, these prior approaches on planar pushing cannot handle our proposed class of "non-planar

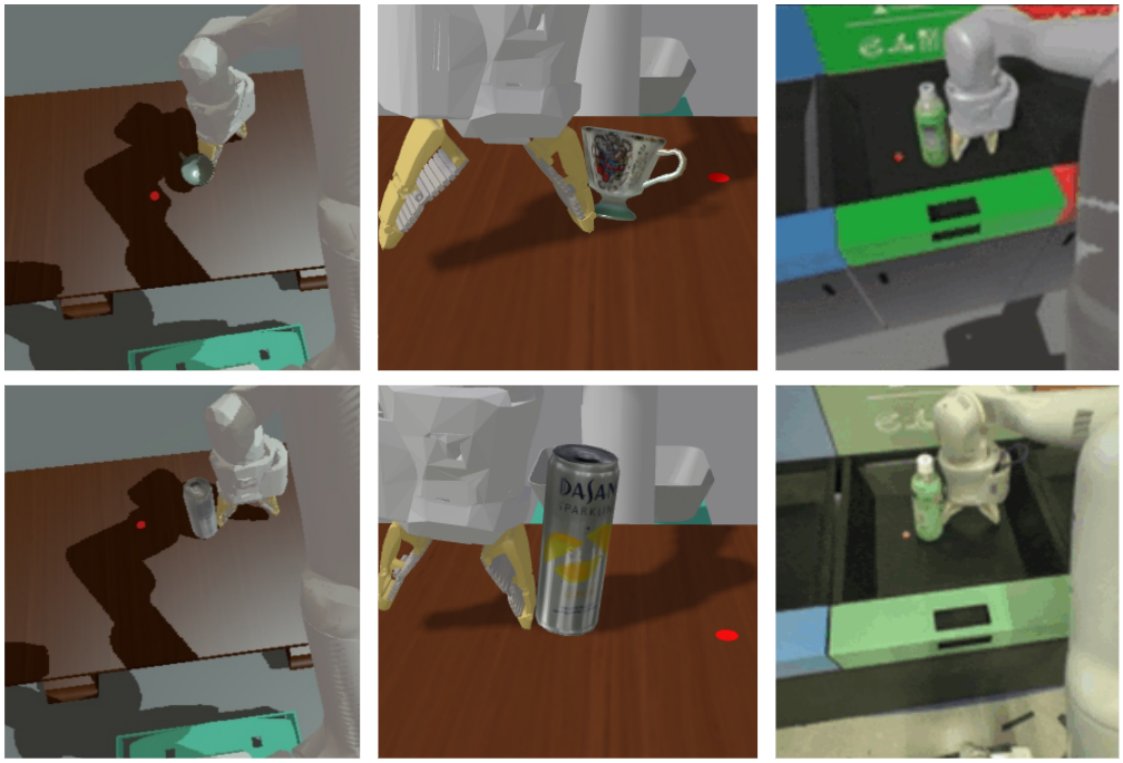

Figure 2.1: Our method, COCOI, achieves dynamic-aware, non-planar pushing of an upright 3D object. The method is robust against domain variations, including various objects and environments, in both simulation and the real world. The first and second columns show the push-on-table simulation setting in the robot's perspective and the third party perspective, respectively. The third column shows the simulated and real world push-in-bin settings in the robot perspective.

pushing tasks" where real-world 3D objects can move with the full six degrees of freedom during pushing.

On non-planar pushing, the majority of the prior works are still in the exploration stage. Ridge et al. propose an object image fragment matching method for 3D objects [91], albeit for a limited library of objects. Zhu et al. use a simulator as a predictive model [129]. Kopicki et al. learn a 3D dynamics model in the PhysX simulator [56]. These works rely on carefully selected objects and precise detection and localization. We make use of the monocular camera image input and the force sensor reading information, and aim to derive a generalized non-planar pushing task for diverse objects. As for usage of force sensor information to help improve manipulation performance, Lee and Zhu et al. use the force torque sensor reading to extend the observation space of a contact rich task [60]. Wang et al. utilize the GelSight force sensor to perceive the force contact information [112]. Our work is the first to use force sensor data to achieve state of the art performance in the domain of

generalizable non-planar pushing.

Our contribution is three-fold. First, we design a learning framework with efficient dynamics inference by leveraging dynamics history information formulated in a dynamics transition structure, and a novel contact-aware sampling strategy. It is also validated that the representation obtained by this framework captures key information of the contact dynamics properties. Second, we demonstrate the effectiveness of the proposed method on a challenging contact-rich non-planar pushing task with a diverse set of objects in simulation. Third, we demonstrate that our trained policies can be successfully transferred to a real-world bottle-pushing task.

## 2.2 Problem Statement

In this section, we formally define the proposed non-planar pushing task and formulate the problem using a Partially Observable Markov Decision Process (POMDP). We focus on the class of pushing tasks where maintaining the upright pose of the object is critical. For example, when pushing a glass of water on the table, the glass should not tilt and spill. In our work, we assume that an object is randomly placed on a flat surface with an upright initial pose. The task objective is to use the robot end effector to push the object to a random target position on the surface, while maintaining its upright orientation. The object can have irregular shape and mass distribution, and the robot may push at any point on the object, making the contact dynamics physics complicated.

We use state $\boldsymbol{s} \in S$ to represent the full task state. In a POMDP, the state is not directly available and can only be inferred using observations. Concretely, we use the image captured using the robot's monocular camera as the high dimensional observation (Fig. 2.1, first and third columns), in which the target location is rendered using a red dot. We also include a low dimensional proprioception observation of the gripper height and open/close status. We use state and observation interchangeably in the following sections. The action $\boldsymbol{a} \in A$ contains the designated position and orientation of the gripper, the gripper open/close command, and a termination boolean. The system transition function $T : S \times A \to S$ follows the physical model, and we use a sparse binary reward function $R : S \times A \to \mathbb{R}$ with value 1 when the distance between the object and the target is smaller than a threshold and the object is upright. The task objective is to maximize the expectation of the discounted cumulative reward, or the return

$$R = \mathop{\mathbb{E}}_{\{(\boldsymbol{s_t}, \boldsymbol{a_t})\}} \left[ \sum_{t=0}^{t=\infty} \gamma^t r_t \right] \tag{2.1}$$

where $\gamma$ is the discount coefficient, and $r_t = R(\boldsymbol{s_t}, \boldsymbol{a_t})$ is the reward obtained at time $t$.

## 2.3 Learning Basic Pushing Controller

We use Q-learning to train object pushing control policy. The definition of the Q function is

$$Q(\boldsymbol{s}, \boldsymbol{a}) = \mathop{\mathbb{E}}_{\{(\boldsymbol{s_t}, \boldsymbol{a_t})\}} \left[ \sum_{\boldsymbol{s}, \boldsymbol{a}}^{t=\infty} \gamma^t r_t \right], \tag{2.2}$$

the expected return starting from state $\boldsymbol{s}$ and taking $\boldsymbol{a}$. The Q function satisfies the Bellman function:

$$Q(\boldsymbol{s_t}, \boldsymbol{a_t}) = \mathbb{E}_{\boldsymbol{s_{t+1}}}[R(\boldsymbol{s_t}, \boldsymbol{a_t}) + \gamma \cdot Q(\boldsymbol{s_{t+1}}, \pi(\boldsymbol{s_{t+1}}))] \tag{2.3}$$

where $\pi(\boldsymbol{s_{t+1}})$ represents the action selected by the optimal policy at the current iteration and state $\boldsymbol{s_{t+1}}$. The optimal policy is defined by:

$$\pi(\boldsymbol{s_t}) = \arg\max_{\boldsymbol{a} \in A} Q(\boldsymbol{s_t}, \boldsymbol{a}) \tag{2.4}$$

One Q-learning iteration uses the collected data tuples $(\boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{s_{t+1}})$ in two steps:

1. Estimate the optimal policy output

$$\pi(\boldsymbol{s_{t+1}}) = \arg\max_{\boldsymbol{a} \in A} Q(\boldsymbol{s_{t+1}}, \boldsymbol{a}) \tag{2.5}$$

2. Minimize the Bellman error

$$\min_{Q} \|Q(\boldsymbol{s_t}, \boldsymbol{a_t}) - [R(\boldsymbol{s_t}, \boldsymbol{a_t}) + \gamma \cdot Q(\boldsymbol{s_{t+1}}, \pi(\boldsymbol{s_{t+1}}))]\| \tag{2.6}$$

For the object pushing task, we represent the Q function with a two stream deep neural network $Q_\theta(\boldsymbol{s_t}, \boldsymbol{a_t})$ parameterized by $\theta$, as shown in Fig. 2.2, similar to [52]. The high dimensional stream feeds stacked current and initial images into a convolutional neural network (CNN) encoder. The low dimensional stream feeds stacked low dimensional state (gripper height and open/close status) and action (designated gripper pose, the gripper open/close command, and a termination boolean) into a fully connected network (FCN) encoder. The outputs of the two streams are combined together with broadcasting, and the outcome is fed into another CNN-followed-by-FCN structure to predict the Q function value for the pushing task. The detailed network structure is elaborated in the next section.

For Q function network training, we adopt QT-Opt, a distributional variant of the Q learning framework for continuous state-action tasks [52, 15], which is known to be able to handle large scale exploration with stable performance. Concretely, we use distributed workers to collect data tuples $(\boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{s_{t+1}})$, and we store them into a replay buffer. Each optimization iteration samples a batch of data tuples. For equation (2.5), the optimal policy output is estimated using an online sampling-based cross entropy method (CEM) based on the current Q function. For equation (2.6), the Q function network weights are updated using gradient descent with the following loss function:

$$l(\theta) = \|Q_\theta(\boldsymbol{s_t}, \boldsymbol{a_t}) - [R(\boldsymbol{s_t}, \boldsymbol{a_t}) + \gamma \cdot Q_\theta(\boldsymbol{s_{t+1}}, \pi(\boldsymbol{s_{t+1}}))]\| \tag{2.7}$$

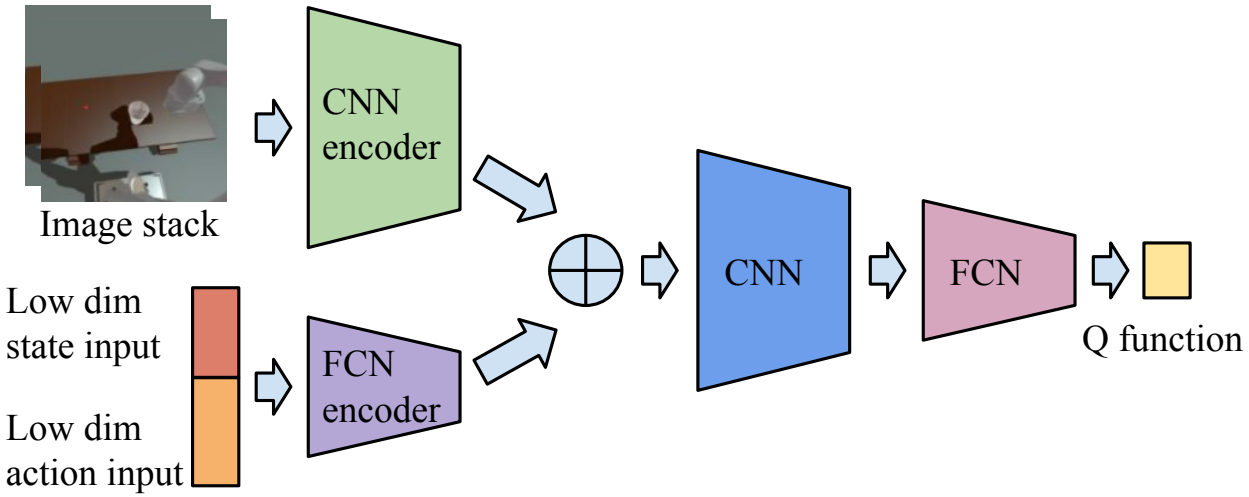


Figure 2.2: The feed forward neural network Q function for object pushing. The stacked current and initial images are fed into a convolutional neural network (CNN) encoder, and the low dimensional input is fed into a fully connected network (FCN) encoder. The output of these two streams are added together and then fed into another CNN-followed-by-FCN structure, the Q value prediction head.

## 2.4 Contact-aware Online Context Inference

### 2.4.1 Online Context Inference

The architecture in Fig. 2.2 has been demonstrated on challenging tasks like grasping [52]. However, given it only has access to a single sensory input, it is not able to infer the dynamics properties of the object, which is necessary for our non-planar pushing task. In this section, we describe online COntext Inference (COI), a module that takes history observation samples and encodes them into a dynamics context representation – thus equipping the control policy with the ability to infer dynamics of the object.

As shown in Figure 2.3, COI consists of a set of additional streams in the policy network that encode history sensor observations into a dynamics context representation. Each stream of COI takes a pair of consecutive sensory inputs separated in time by 0.5s (the sensor update interval in our robot system). We denote each sensory input pair as a tuple $H_\tau = (I_\tau, I_{\tau+1}, f_\tau)$, where $I$ and $f$ refer to the camera image and force reading respectively, and $\tau$ represents the time at which the sensor inputs are retrieved. This formulation follows the structure of a dynamics transition of a pushing operation, such that the contact dynamics properties can be inferred from the input information. The encoded sensory input for each stream is then averaged to obtain the final dynamics context representation of COI, which is concatenated with the state-action representation to estimate the Q value.

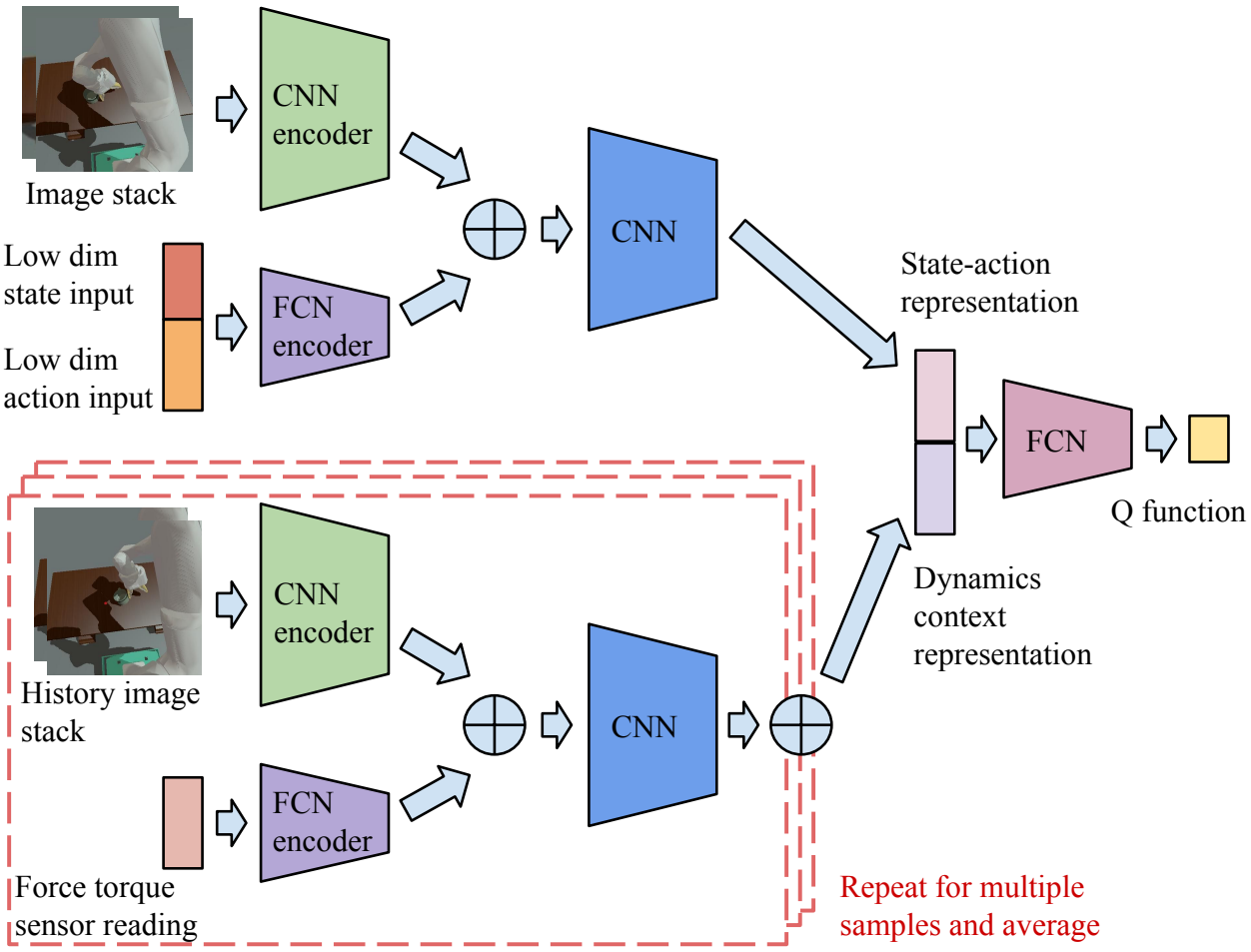


Figure 2.3: The proposed COntext Inference (COI) module, which works in parallel with the state-action stream. The COI takes as input a dynamics-transition-structured history sample consisting of the stacked pre-impact and post-impact images and the force reading, to infer the dynamics representations. The representations of different samples are averaged and concatenated to derive the state-action representation, which is then fed into a final Q value prediction network. Networks with same color share architectures, but not necessarily network weights.

### 2.4.2 The Contact-aware Sampling Strategy

Given an architecture that can process multiple history sensor observation pairs, key questions are:

1. How many history samples should we include?
2. At what time should we retrieve the sensor observations?

With a higher number of history samples, the policy has more information to infer a potentially less noisy object dynamics representation, at the cost of higher computation time and memory. In our experiments, we found three history samples to be a reasonable number to achieve good inference performance with manageable computation cost.

The timings at which sensor observations are sampled is vital for the performance of COI. An arbitrarily sampled sensor observation pair may contain limited information about the object dynamics (e.g. when gripper is far away from the object) and contribute little to the learning performance. To ensure that each history sample contains useful information, we propose a contact-aware sampling strategy which actively checks the readings of a force torque sensor that is mounted at the robot gripper, and only collects a sample when the contact force magnitude is considerably large ($> 1$ N), as shown in Fig. 2.4. This strategy guarantees the samples to be representative, in that the gripper and the object are in contact. We call this sampling strategy COntact-aware-COI, or COCOI.

In our experiments, we validate the performance of COCOI by comparing it to a naïve strategy: vanilla COI, or VCOI, where history samples are retrieved with a uniform sampling interval, as shown in Figure 2.4.

## 2.5 GAN for Visual Gap Bridging

In order to adapt the RL policy trained in simulation to the real world, we also need to overcome the discrepancy between the rendered, simulated image and the image captured by a real-camera. We adopt RetinaGAN, a generative adversarial network (GAN) approach to generate synthetic images that look realistic with object-detection consistency [50][1]. Qualitative performance is shown in Fig. 2.5. We train the RL policy with simulation data only and directly deploy it on the real robot.

## 2.6 Experiments

### 2.6.1 Setup

We train the control policies in PyBullet simulation [29]. We first define a flat surface randomly placed in front of the robot. The surface can be either a desk surface or a designated

[1] `https://retinagan.github.io`

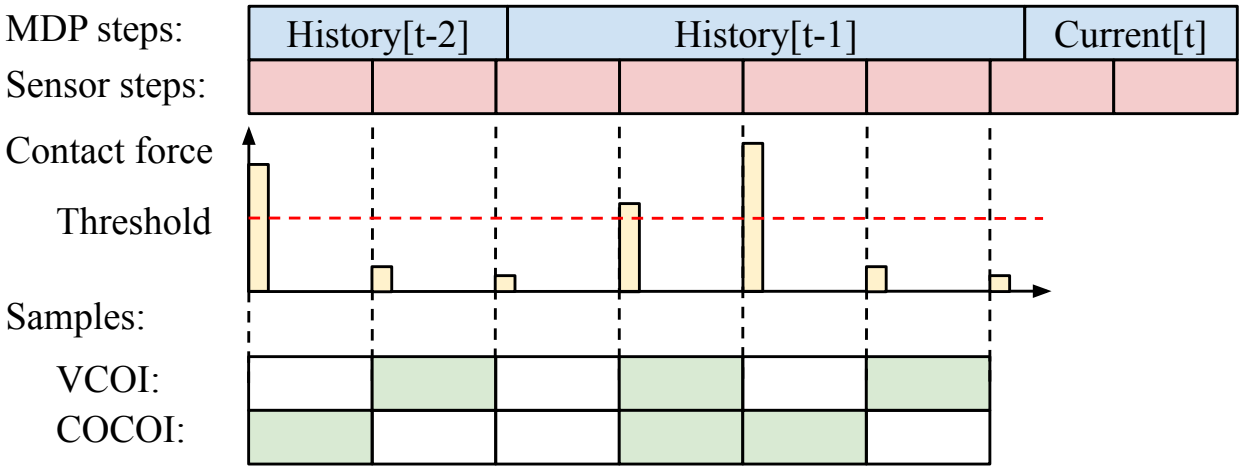


Figure 2.4: Illustrations of the sampling strategies. For VCOI, the samples are retrieved with a uniform sampling interval. COCOI takes a contact-aware sampling method which actively checks the contact force, and only retrieve samples when the force magnitude is larger than 1 Newton.

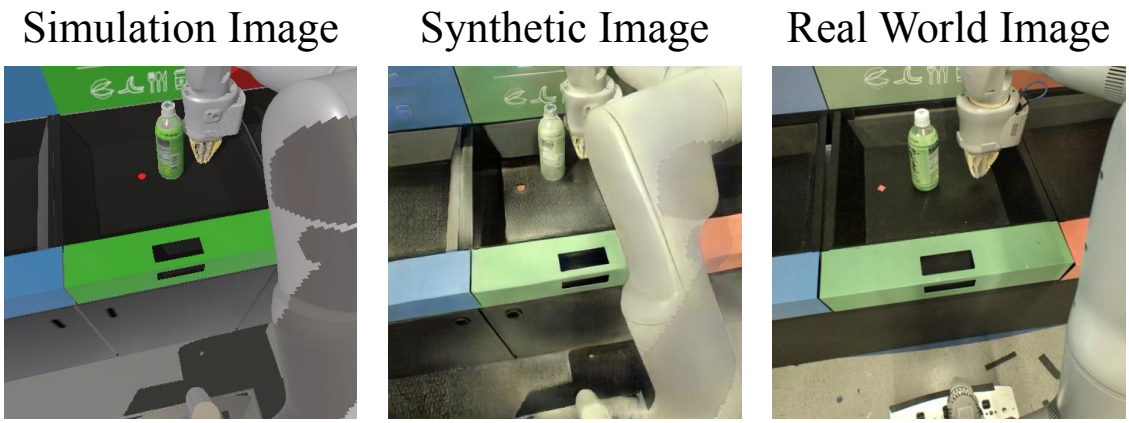


Figure 2.5: Images of the pushing-in-station setting in simulation, with RetinaGAN visual adaptation, and in the real world.

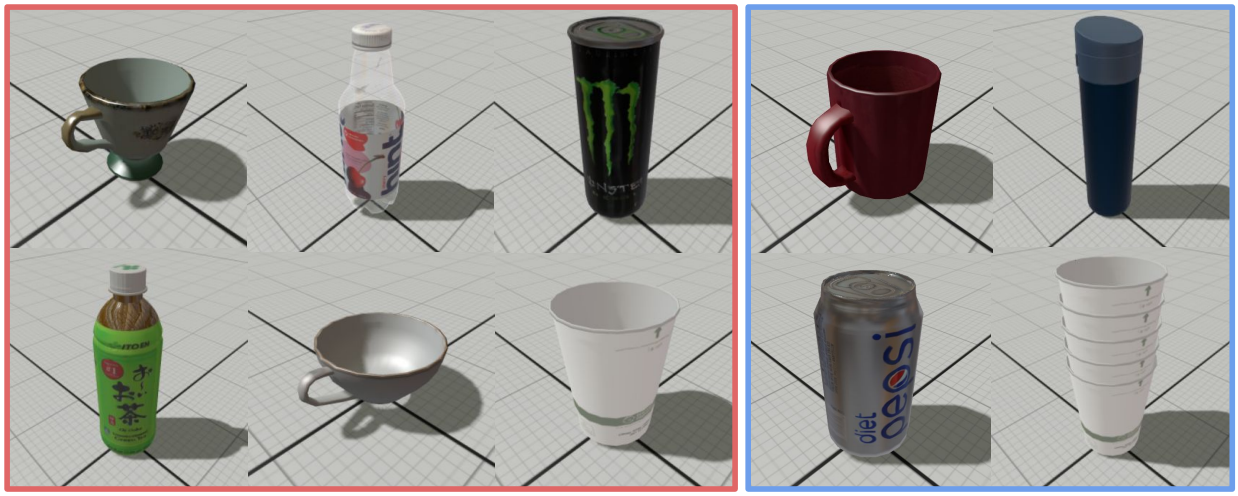


Figure 2.6: A subset of the 75 different 3D objects used for training and testing.

flat area inside a trash bin, as shown in Fig. 2.1. We use a 3D model set containing 75 different objects, such as cups, bottles, cans, mugs, etc. (Fig. 2.6). We divide the objects into a training set containing 64 objects and an unseen testing set of 11 objects including a stack of cups. Note that the upper cups in the stack have the degrees of freedom to tilt when pushed, which makes the pushing task more challenging. In PyBullet, the contact physics between the robot gripper and the object is modelled using a point contact model with an elliptic friction cone and a tunable friction. We train our policies with a randomized friction coefficient in the range [0.5, 1.0] and evaluated the policies with a larger range of [0.0, 1.5].

At the beginning of each episode, the object and the target position are randomly placed within a rectangular area. We set the object upright on the surface, and the target position is rendered using a red dot. The robot gripper is initialized at a randomized position beside the object. The push policy controls the robot gripper to push the object to the red dot. An episode is considered successful if the object is pushed to within 5 cm of the target, and the object tilting angle remains smaller than 0.1 rad. We also apply a small penalty at each timestep to encourage faster execution. The discount factor $\gamma$ of the POMDP is 0.9. During evaluation of our method and the baseline methods, we run the model for 1000 episodes. We report the success rate of different methods for comparison, where we define an episode to be successful when the object is pushed to a location within 5cm from the target in an upright position.

### 2.6.2 Policy Models

The deep neural networks for the Q functions are constructed as in Fig. 2.2 and Fig. 2.3, and the blocks with the same names share the same network architecture. The detailed architecture is shown in Table I. In the table, the symbols follow those in [52]. Concretely, Conv(64, 6, 2) means a convolutional layer with 64 channels, filter size 6 and stride 2, FC(64) means a fully connected layer with hidden layer size 64, and MaxPool(2) means a max pooling layer with filter size 2. When the (8, 8, 64) shape image stream representation and the 64 dimensional low dimensional stream representation are added together, the low dimensional stream representation is first reshaped to (1, 1, 64), and a broadcasting operation is applied.

We train and compare four models:

1. The baseline model: the basic feed forward Q function network as described in Section 2.3 and shown in Fig. 2.2. This model is the most straightforward and shows the capability of the baseline RL method.

2. The VCOI model: the VCOI method as shown in Figure 2.3. The history observations are sampled using uniform sampling.

3. The oracle: the architecture of this model is the same as the baseline, but we expand the low dimensional state input with 2 key, unobservable dynamics parameters: the object mass and friction coefficient. We directly extract the ground truth parameter values from the simulator to obtain an oracle model.

4. The COCOI: the proposed COntact-aware Online COntext inference model as shown in Figure 2.3. The active contact-aware sampling strategy is applied.

### 2.6.3 Policy Learning

We adopt the QT-Opt framework [52] to train the policies. In the training setting, the objects and goal locations are randomly sampled in a 0.5m $\times$ 0.3m area, the object mass is randomized from 0.05 kg to 0.5 kg, and the friction coefficient is randomized from 0.5 to 1.0. We train the models using stochastic gradient descent, using a learning rate of 0.0001 and momentum of 0.9. We train the models with a batch size of 2048 for 80k steps.

In the early stage of training, we use a rule-based scripted policy, which moves the gripper along the line that connects the object and the target, to generate successful episodes and improve the exploration efficiency. This rule-based method only achieves less than 5% success rate, illustrating the difficulty of the non-planar pushing task. During training, we observe that the policy first obtains the capability to solve easier scenarios where the object-goal distance is short and then gradually learns to push objects that are initialized far from the goal. From our experience, the consistency of the converged success rate was similar to previous work [15].

### 2.6.4 Performance under Domain Variations

We evaluate and compare the performance of the four models using the push-on-table task in simulation with a large variety of settings. We report the task success rate in Table 2.2 and Table 2.3.

First, we vary the initial range of the object and the target, and the results are shown in Table 2.2. COCOI consistently outperforms VCOI and baseline that are described in Section 2.6.2, and COCOI achieves a similar performance to the oracle. Specifically, COCOI shows an average relative improvement of 50% and 20% success rate compared to the baseline and VCOI, respectively.

Table 2.1: Architecture of each module in the Q networks

| **Block name** | **Architecture [52]** |
|---|---|
| Images input | Tensor with shape (472, 472, 6) |
| Low dim state input | 2 dim vector |
| Low dim action input | 7 dim vector |
| Force torque sensor reading | 3 dim vector |
| CNN encoder | Conv(64, 6, 2)<br>MaxPool(3)<br>Repeat x6: Conv(64, 5, 1)<br>MaxPool(3) |
| FCN encoder | FC(256)<br>FC(64)<br>Reshape(1, 1, 64) |
| CNN | Conv(64, 3, 1)<br>MaxPool(2)<br>Repeat x3: Conv(64, 3, 1) |
| state-action representation | (8, 8, 64) dim matrix |
| dynamics context representation | (8, 8, 64) dim matrix |
| FCN | FC(64)<br>FC(64)<br>Sigmoid |
| Q function | 1 dim vector |

Second, we fix the initial object placement to 0.4m $\times$ 0.3m and vary dynamics properties. We change the friction coefficient and the object mass to be outside the training range. We

Table 2.2: Comparison of success rate for models evaluated with different initial placement settings.

| **Initial range** | **0.3m x 0.6m** | **0.3m x 0.5m** | **0.3m x 0.4m** |
|---|---|---|---|
| Baseline | 26.5% | 34.1% | 51.1% |
| COI | 36.0% | 43.2% | 63.4% |
| Oracle | **43.8%** | 53.9% | **73.8%** |
| COCOI | 43.0% | **59.3%** | 73.2% |

| **Initial range** | **0.3m x 0.3m** | **0.25m x 0.5m** | **0.25m x 0.5m** |
|---|---|---|---|
| Baseline | 59.2% | 40.0% | 42.5% |
| COI | 72.2% | 49.0% | 50.8% |
| Oracle | **78.2%** | 60.5% | 62.4% |
| COCOI | 75.7% | **64.7%** | **62.6%** |

Table 2.3: Comparison of success rate for models evaluated with different dynamics properties settings.

| **Model** | **default setting** | **0.0-0.5 friction** | **1.0-1.5 friction** |
|---|---|---|---|
| Baseline | 51.1% | 53.8% | 32.1% |
| COI | 63.4% | 59.6% | 44.2% |
| Oracle | **73.8%** | 39.6% | 38.2% |
| COCOI | 73.2% | **72.6%** | **47.2%** |

| **Model** | **0.5-1.0kg mass** | **Unseen objects** | **Cup stack** |
|---|---|---|---|
| Baseline | 32.6% | 42.5% | 26.2% |
| COI | 44.7% | 48.3% | 38.8% |
| Oracle | 41.4% | **60.9%** | 36.5% |
| COCOI | **49.9%** | 58.9% | **43.7%** |

also test performance of the models on unseen objects and a stack of cups whose inertia can change during pushing. Evaluation results with these setting variations are reported in Table 2.3. Again, COCOI performs significantly better than VCOI and the baseline model across all the settings. COCOI reaches even higher success rate than the oracle in many settings, which indicates that COCOI is capable to capture more than just the oracle information (object mass and friction coefficient) - there are other factors such as the object shape and contact point that affect the dynamics properties. The performance of the oracle model is especially poor in some cases where the real friction parameter is not in the range of the training set. This could be due to the policy overfitting to the input dynamics parameters.

Overall, COCOI outperforming VCOI demonstrates the benefit of our contact-aware sampling strategy to effectively leverage force torque sensor data history for inferring dynamics.

We also observed some smart behaviors evolved from RL policy when looking at the motions from the evaluations. For example, the robot can break contact from object when the object leans, and it can open the gripper to use the finger to make subtle impacts.

### 2.6.5 Interpretation of Context Representations

To inspect the dynamics context learned by COCOI, we visualize the inferred representations for three settings with different dynamics parameters. For each setting, we run our controller for 20-30 episodes to fill a buffer of 256 dynamics context representations. We then visualize these representations using a combination of principle component analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) [69] (Fig. 2.7). The visualization shows clear separation between settings, which indicates COCOI learns to infer the dynamics properties. Also, representations within one episode are grouped closer to each other than to other episodes, suggesting that learned representations are consistent and structured. Moreover, we observe the order of the clusters is consistent with the dynamics properties: the clusters with the largest mass and friction and the smallest mass and friction are farthest apart, while the cluster with intermediate parameters is in the center.

### 2.6.6 Real World Deployment

To test real world deployment, we design a push-in-bin task in both the simulator and in the real world, as shown in Fig. 2.1. We adopt the method described in Section 2.5 and train a RetinaGAN model to adapt the simulation images to synthetic images with realistic appearance. We train the pushing policy with COCOI based on synthetic images and run 10 real world pushing episodes. We achieve 90% success, demonstrating the capability of our 3D pushing policy to overcome both the visual and dynamics domain gap. This push-in-bin task is simpler than the randomized simulation push-on-table scenario, as the initial distance between the object and the target is relatively closer. Therefore fewer actions are required, which leads to more frequent success. Fig. 2.8 shows example sequences in simulation and the real world.

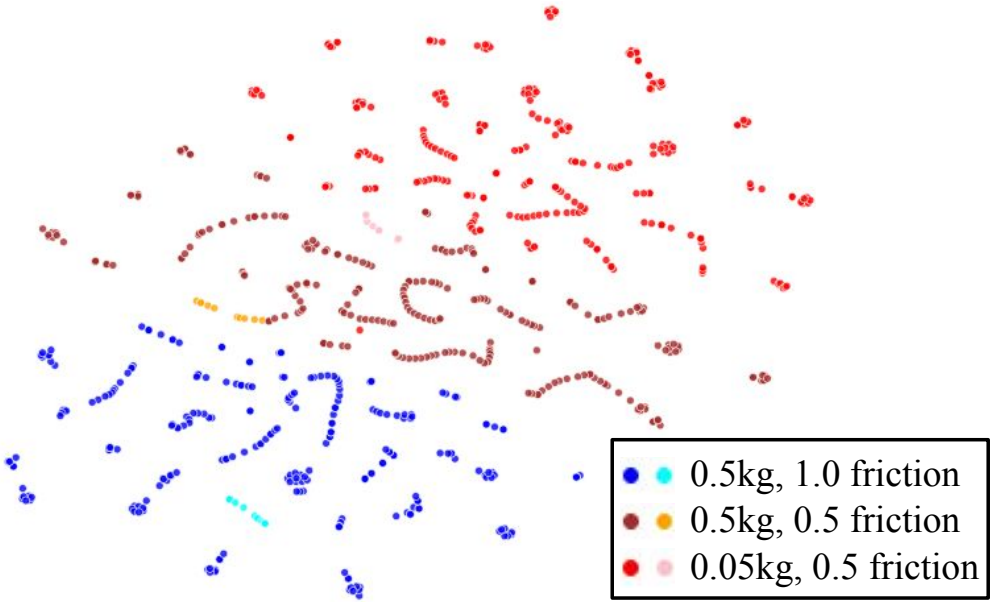


Figure 2.7: t-SNE visualization of the inferred context representation for three different dynamics parameters settings. For each setting, the context representations from a randomly chosen episode is highlighted with a brighter color. The clusters show clear separation and are distributed with an order consistent with the friction magnitude.

## 2.7 Chapter Summary

In this chapter, we propose COCOI, a deep RL method that uses history robot-object interaction samples to infer contact dynamics context, and we show it outperforms baseline in contact-rich manipulation tasks with domain variations. We design and study COCOI on a non-planar pushing task commonly seen in everyday life. Extensive experiments demonstrate the capability of COCOI in a wide range of settings, dynamics properties, and sim-to-real transfer scenarios.

There are many promising future work directions to pursue based on our approach. For example, we study the non-planar pushing task with a single object on the surface. It would be interesting to train manipulation controllers that can perform pushing with multiple objects or in a cluttered environment. In addition, extending our approach to push non-rigid objects such as a piece of cloth is another important direction that can further expand the capability of our controller.

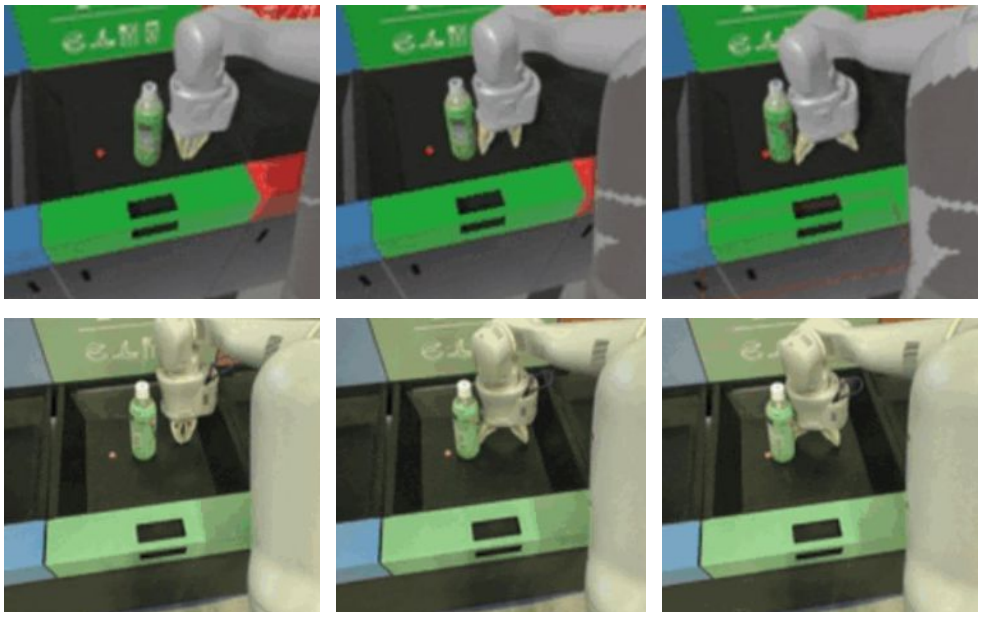

Figure 2.8: Visualization of sim-to-real pushing policy transfer.

# Chapter 3

# Representation Selection and Design in Model-based RL, Model-free RL, and Supervised Learning

In this chapter, we present a series of works that investigates the design of learning representation for different learning approaches and different task settings [23, 117, 119]. The learning approaches we investigate include model-based reinforcement learning (RL), model-free RL, and supervised learning. We study model-based and model-free RL in the setting of autonomous driving in the CARLA simulator. We first present dynamics model selection for model-based RL. In this work, we develop a model-based RL algorithm of guided policy search (GPS) for urban driving tasks. The algorithm iteratively learns a parameterized dynamic model to approximate the complex and interactive driving task and optimizes the driving policy under the nonlinear approximate dynamic model. We present the use of a global Gaussian mixture model (GMM) with local linear adaptation, can help model the complex driving dynamics. We then study the selection of a state feature extractor in model-free RL for a complicated and interactive autonomous driving task. We present the superior performance of a graph representation and graph neural network (GNN) feature extractor over the state vector and sensor data baselines. Finally, in a human behavior prediction example, we present a history-encoding representation that is both interpretable and effective for prediction of human intention. Overall, we show that a proper learning representation design is needed for the success of various learning approaches across different application domains.

## 3.1 Representation Reasoning in RL-based Autonomous Driving

Autonomous driving in an urban situation is a complicated challenge that requires extensive recognition of the surrounding environment and good decision-making skills. The traditional

rule-based methods often fail to accomplish satisfactory driving performance in highly complicated traffic scenarios due to biased human-designed heuristics. But massive data and computation power have advanced learning methods in various fields of computer vision, data science, robotics, and autonomous driving. In the field of autonomous driving, the most popular and studied learning-based methods are imitation learning (IL), which essentially uses neural networks to do behavior cloning of the input-output combinations of the human expert drivers. One of the earliest attempts was the ALVINN [83] and [74], and more recent works include [17], [114]. Waymo has also applied IL to learn an urban driving policy from extensive human driving data [7]. Over the years, researchers have made progress in using larger datasets and more sophisticated neural network structures. However, since the policies are still trained in a supervised manner, they are inevitably limited by the dataset. Chen et al sought to obtain a more stable policy by first learning a future trajectory and then using a low-level controller to track the trajectory [24].

However, the IL-based methods are usually limited to the expert demonstration data and a lack of generality capability. To summarize, IL methods suffer from the following shortcomings:

1. It requires collecting a large amount of driving data, which is costly and time consuming.

2. The training dataset is biased compared to real-world driving, since expert drivers generally do not provide data for dangerous situations.

3. IL essentially clones human driver behavior and cannot exceed human performance.

Compared with the rule-based and IL methods, RL methods are more suitable for training autonomous driving policies because they can have the agent explore and interact with the traffic environment without human guidance. Therefore, the policies trained can learn cases that are not in the IL training set and even potentially exceed human drivers' performance. Within the class of RL methods, there are two major categories:

1. **Model-free RL**, in which the reconstruction of a system dynamics model is not required. Model-free RL has been demonstrated capable of solving many challenging tasks, such as continuous control based on state space [66, 71], driving game with low-dimensional feature vector or image pixels as inputs [94]. For the application for driving, [113] learns the steer command in a simulator with discrete action space. Wayve's researchers use more sophisticated network and representation learning to use RL to drive an autonomous vehicle in urban situations [53]. Chen et al.'s work on model-free RL provided a benchmark comparison of different model-free RL algorithms in the CARLA simulator [25].

2. **Model-based RL methods**, in which optimal control is applied based on the inference of an approximation of the system dynamics function. Model-based RL lies in the intersection of model-based planning, control, and RL, which iteratively learns the

model of the complex environment and optimizes the control policy base on the understanding of the environment dynamics. Model-based RL has been proved capable of solving a number of complicated manipulation tasks [63] and aerial vehicle [127] in both simulators and the real world.

In general, model-free RL methods suffer from low sample efficiency and a lack of interpretability. This is because the RL policy is trained in an end-to-end manner, and the structural information of the complicated driving scenarios cannot be modeled in a compact and representative way. Still, in model-based RL, an accurate approximation of the complicated system dynamics is required for accurate planning and control. The complicated nature of autonomous driving makes it challenging to model using a data-driven method. Therefore, selecting a suitable representation and adopting a suitable representation learning strategy inside the policy network (or inside the dynamics model) is of significance for the success of the policy learning using model-free RL (or model-based RL).

In the first half of this chapter (from Section 3.2 to Section 3.4), we investigate the representation selection and representation learning strategies for solving an autonomous driving task based on model-based and model-free RL. For model-based RL, we propose to adopt a GMM to serve as a general global model and adapt a local linear Gaussian model to precisely capture the complicated autonomous driving system dynamics. For model-free RL, we design the graph neural network representation encoding so as to better extract the multi-entity relationship in urban autonomous driving.

## 3.2 Autonomous Driving Problem Setup

We model the autonomous driving task using a Markov Decision Process (MDP) whose state at timestep $t$ is defined using $s_t$. The agent interacts with the environment by taking actions $a_t$, derived using its control policy $\pi_\theta(a_t|s_t)$, and the control policy is parameterized by $\theta$. In autonomous driving, the MDP state can include lane tracking status, speed of the autonomous vehicle, and the obstacle vehicles. The MDP action represents the control commands for the autonomous vehicle, corresponding to throttle, brake, and steering angle. Accepting the action of the agent, the environment would evolve in time according to the system dynamics $p(s_{t+1}|s_t, a_t)$. The system dynamics can be very complicated given the dense traffic interaction during urban driving. Furthermore, the obstacles' movements are usually not controllable by the control commands of the autonomous vehicle.

With the system governed by the dynamics function and the control policy, the trajectory distribution can be described as a distribution of

$$\pi_\theta(\tau) = p(x_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{3.1}$$

where $\tau = \{s_0, a_0, s_1, a_1, ..., s_T, a_T\}$. The goal of the autonomous driving task is described by a cost function, $l(s_t, a_t)$, which can be penalties for the lateral and reference speed deviation of the autonomous vehicle or for collisions with the obstacle. The overall objective would then be to minimize the expectation $E_{\pi_\theta(\tau)}[\Sigma l(s_t, a_t)]$, which we will abbreviate as $E_{\pi_\theta}[l(\tau))]$. It is noted that the problem formulation for RL is the same as that of the optimal control, only the system dynamics $p(s_{t+1}|s_t, a_t)$ is not known and often very complex, such that one cannot precisely model using mathematical equations, but can use only parametrized models to approximate it. This brings challenges to the learning of a driving policy; but the setting can enable the RL policy to control the vehicle under complicated vehicle dynamics and dense traffic that is hard to model.

### 3.2.1 CARLA Driving Simulation Settings

We conduct our experimental validation using the CARLA urban driving simulator [35], which is a high-fidelity open-source simulator by Intel.

**For the model-based RL experiments,** due to the limitation of the policy optimization operation, we conduct experiments under three different scenarios: straight lane, 90° turning, and circular roundabout. For each scenario, we investigate the performance of the proposed method and a series of baseline algorithms with and without obstacle vehicles. The CARLA simulator and the simulated scenarios are shown in Fig. 3.1. In order to test the performance of the RL methods serving as motion planner, we directly extracted the vehicles and map states from the CARLA simulator, which is input to the model-based RL policy. Because the RL policy takes in fixed dimensional states, we designed two kinds of states for the scenarios with and without obstacles. For the tasks without obstacles, the state inputs include the lateral deviation and yaw error of the autonomous vehicle with respect to the roadblock, and the velocity of the autonomous vehicle. For the cases with obstacles, in order to maintain fixed state dimension, we investigate only the influence of the front vehicle, and we augment the state of the relative position and velocity between the autonomous vehicle and the front vehicle to the previously defined state to obtain the state for these cases.

**For the model-free RL experiments,** since the policy is learned in an end-to-end manner, we are able to learn a more general policy that drives the ego vehicle around within the simulated town. There are 100 surrounding vehicles that are controlled using the intelligent driver models (IDM) randomly generated inside the simulator, and the ego vehicle could encounter some of them during its navigation. We can extract various kinds of state observations from the simulator, such as the front view camera image, the simulated lidar observation, and the state vectors for the surrounding vehicles. We can also design and render a block representation of the local traffic environment for better visualization. Fig. 3.2 shows the block representation and the simulated sensor observations of the driving setting.

Figure 3.1: The CARLA urban driving simulator. Upper Left: The ego view of the autonomous vehicle. Upper Right: The roundabout experiment setting. Lower Left: The 90° turning experiment setting. Lower Right: The straight driving experiment setting.

## 3.3 Model-based RL: Dynamics Representation and Policy Optimization

The model-based RL method we adopt is an RL method that combines the strengths of both RL and optimal control. The algorithm follows the formulation as shown in Fig. 3.3. The model-based RL algorithm iteratively updates the belief of the system dynamics model and optimizes the control policy under the current dynamics approximation function. The two steps are called the system dynamics learning step and the policy optimization step. Our algorithm shares a similar framework of the guided policy search (GPS) [62]; while we do not learn an imitation neural network of the optimization based policy, we still refer to our algorithm as a variant of GPS. For the system dynamics learning step, the agent first explores the environment and collects trajectory samples of $\tau = \{s_0, a_0, s_1, a_1, ..., s_T, a_T\}$. The collected data tuples $(s_t, a_t, s_{t+1})$ are then used to learn the system dynamics function

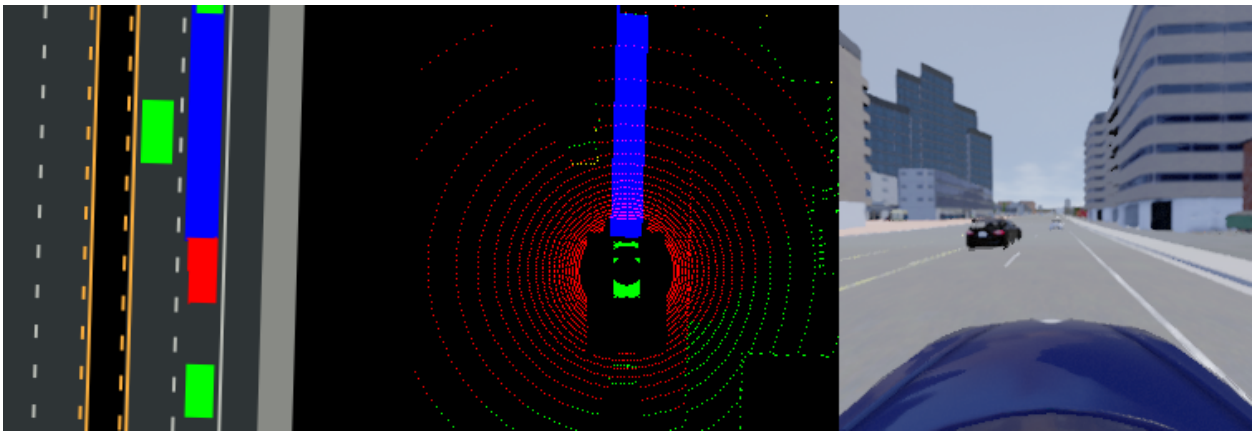

Figure 3.2: Block representation and the simulated sensor observations of the CARLA driving setting. Left: The block representation showing the ego vehicle (red), surrounding vehicles (green) and the road map. Middle: The simulated lidar observation. Right: The front view camera image.

$p(s_{t+1}|s_t, a_t)$. This is a supervised learning process, optimizing the objective of:

$$\max_{p(s_{t+1}|s_t,a_t)} p(\tau) \tag{3.2}$$

The policy optimization step learns the optimal policy $\pi_\theta(a_t|s_t)$ based on the learned dynamics $p(s_{t+1}|s_t, a_t)$ in order to obtain an optimal return $E_{\pi_\theta}[l(\tau))]$. There is a large amount of freedom in the representation formulation and the representation learning strategy selections. In our autonomous driving application, we select the system dynamics representation to be a GMM, which adopts the idea of local models in order to get high sample efficiency. Concretely, a time-varying linear-Gaussian model is applied to approximate the local behavior of the system dynamics. Therefore, the learned system dynamics model shall follow the following formulation:

$$p(s_{t+1}|s_t, a_t) = \mathcal{N}(A_t s_t + B_t a_t + f_t, F_t) \tag{3.3}$$

Similarly, the policy model is also represented using a linear Gaussian model, denoted as:

$$\pi_\theta(a_t|s_t) = \mathcal{N}(K_t s_t + k_t, C_t) \tag{3.4}$$

The local linear models can be learned very efficiently with a small number of samples, but they can only describe the local properties of the nonlinear functions. Therefore, after

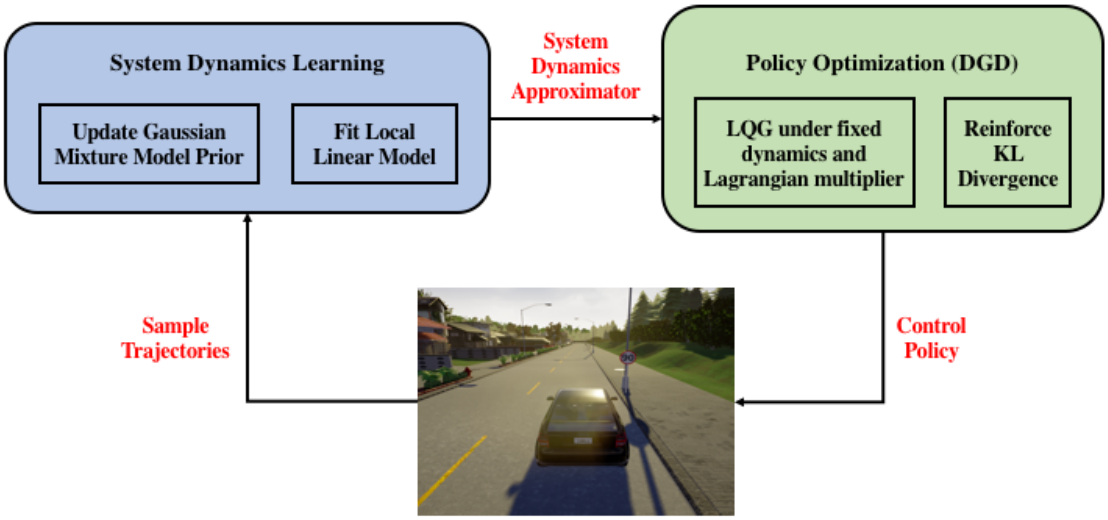


Figure 3.3: The framework of our proposed model-based RL method. For each iteration, we first collect trajectories in the simulator using the current policy, then the trajectory samples are used to update the system dynamics approximator, which is further used as the model within the model-based policy optimization.

the policy optimization step, the new trajectory produced by the new policy, denoted $\tau$, shall not differ significantly from the old trajectory samples we used to learn dynamics $p(s_{t+1}|s_t, a_t)$, so as to guarantee the modeling representation accuracy of the local linear model. We denote the old trajectories using $\hat{\tau}$. We adopt the Kullback-Leibler divergence to describe the change of the trajectory distributions, then the policy optimization process can be modeled using the optimization problem below.

$$\min_{\theta} E_{\pi_\theta}[l(\tau))] \tag{3.5}$$

$$\text{s.t. } D_{KL}(p(\tau)\|p(\hat{\tau})) < \epsilon \tag{3.6}$$

In the next two subsections, we elaborate on the representation selection and usage in the system dynamics learning step and the policy optimization step.

### 3.3.1 System Dynamics Learning

The goal of system dynamics learning is not only to learn a precise local linear function, but also to learn it efficiently. Therefore, we adopt a global model as the prior, which evolves

throughout the whole model-based RL lifetime, and we fit the local linear dynamics to it at each iteration. The global prior itself does not need to be a good approximation; it only needs to capture the major property of the system dynamics to increase the regression sample efficiency. This approach shares a similar idea with model agnostic meta learning (MAML) [40], where the prior model is not an accurate approximation of the dynamics; but after a fast adaptation process, the updated model could capture more precise dynamics properties.

In the case of autonomous driving, there are a series of different driving patterns. Within each driving pattern, the dynamics models follow a similar property. Therefore, we adopt the GMM to serve as the nonlinear prior model, with each mixture element serving as prior for one driving pattern. Under this setting, each tuple sample $(s_t, a_t, s_{t+1})$ is first assigned to a pattern and then used to update the mixture element. This process is a typical Expectation Maximization (EM) process used to train a GMM. Finally, at each iteration, we fit the current episode of data tuples $\{(s_t, a_t, s_{t+1})\}$ to the GMM, incorporating a normal-inverse-Wishart prior. We derive the local linear dynamics $p(s_{t+1}|s_t, a_t)$ by conditioning the Gaussian on $(s_t, a_t)$.

### 3.3.2 Policy Optimization

In order to solve for the policy optimization, we incorporate a popular gradient-based method for constrained optimization: the dual gradient descent (DGD), which is summarized in algorithms 1. The main idea of the DGD is as shown below. We first write out the Lagrangian:

$$L(\theta, \lambda) = \sum_{t=0}^{T-1} E_{\pi_\theta}[l(s_t, a_t) + \lambda(D_{KL}(p(\tau)\|p(\hat{\tau})) - \epsilon)] \tag{3.7}$$

,where

$$D_{KL}(p(\tau)\|p(\hat{\tau})) = E_{\pi_\theta}[log(p(\tau)) - log(p(\hat{\tau}))] \tag{3.8}$$

The DGD trick is to first minimize the Lagrangian function under a fixed Lagrangian multiplier $\lambda$ and then increase the $\lambda$ penalty if the constraint is violated so that more emphasis is placed on the constraint term in the Lagrangian function in the next iteration. We can reformulate the minimization of the Lagrangian function to be some trajectory optimization problem with regard to some augmented cost function $c(s_t, a_t) = l(s_t, a_t)/\lambda - log(p_{\hat{\tau}}(a_t, s_t))$, where $l(s_t, a_t)$ is the original MDP step cost. The optimization is, therefore, updated to:

$$\min_{\pi_\theta} \sum_{t=0}^{T-1} c(s_t, a_t) \tag{3.9}$$

Since we can directly compute the cost function $c(s_t, a_t)$ and its derivatives, we can then solve the trajectory optimization problem using linear quadratic Gaussian (LQG) [5]. After

the Lagrangian is optimized under a fixed $\lambda$, in the second step of DGD, $\lambda$ is updated using the function below with step size $\alpha$:

$$\lambda \leftarrow \lambda + \alpha(D_{KL}(p(\tau)\|p(\hat{\tau})) - \epsilon)) \tag{3.10}$$

, and the DGD loop is closed.

**Algorithm 1** Dual gradient descent for policy optimization.

Constrained optimization problem defined by (5-6)
Initialize $\lambda = \lambda_0$, $itr = 0$
**while** $itr < maxIteration$ **do**
  Rewrite $L(\theta, \lambda)$ to $\sum_{t=0}^{T-1} c(s_t, a_t)$
  Solve for optimal $\pi_\theta$ using LQG
  Evaluate constraint violation $\Delta = D_{KL}(p(\tau)\|p(\hat{\tau})) - \epsilon)$
  Update $\lambda \leftarrow \lambda + \alpha\Delta$
**end**

We also designed concise and effective cost functions to model the objective of the driving tasks. For the roadblock tracking without considering the obstacle, we define the tracking cost to be

$$c_t(s_t, a_t) = \alpha_l \Delta y^2 + \alpha_y \Delta\phi^2 + \alpha_v (v - v_{ref})^2 + \alpha_a a^2 + \alpha_\sigma \sigma^2 \tag{3.11}$$

where $\Delta y$ is the lateral deviation, $\Delta\phi$ is the yaw angle error, $v$ is the velocity of the autonomous vehicle, $v_{ref}$ is the reference speed for the tracking, $a$ is the acceleration action, and $\sigma$ is the steering action.

When considering the obstacle, we designed a nonlinear cost function that takes effect only when the autonomous vehicle is in the same lane as the obstacle vehicle and the distance $s$ is smaller than 20m, where we add the additional term of

$$c_{aug}(s_t, a_t) = \beta_s(20 - s) + \beta_v v_{approach} \tag{3.12}$$

where $v_{approach}$ is the approaching speed of the ego and obstacle vehicle. If there is a collision of between the ego vehicle and an obstacle vehicle, a large penalty is given and the episode is terminated. The reward function for the MDP and for RL is the negative of the cost function.

### 3.3.3 Compared Methods

We refer to our method as GPS. We adopt a GMM of 20 mixtures to serve as the model prior, and we collect four trajectories every time for the update of the system dynamics and

the linear Gaussian policy. We compare the proposed GPS model-based RL method with a series of popular model-based and model-free RL methods, in terms of their training and testing performance. We describe the setting for the proposed method and briefly introduce the baseline algorithms and their settings in the next subsections.

**The cross-entropy method (CEM)** [31] has been one of the most simple and popular policy search RL methods for policy optimization. In order to optimize the parameterized policy $\pi_\theta$, the CEM adopts the assumption of Gaussian distribution of $\theta \sim \mathcal{N}(\mu, \sigma^2)$. It iteratively samples $\theta$ from the distribution, using it to collect sample trajectories, and then it updates $\mu$ and $\sigma$ using the $\theta$s that produce the best trajectories. Since the CEM is also a policy search method, to make a fair comparison, we also collect four trajectories every time.

**The model-free RL soft actor critic (SAC)** [45] algorithm is the state-of-the-art model-free RL algorithm. It has some of the best sample complexity and convergence properties among the model-free RL algorithms. The SAC maximizes both the expected reward (negative cost) and the entropy

$$\max_\theta \sum_{t=0}^{T-1} E_{\pi_\theta}[-c(s_t, a_t) + \alpha H(\pi_\theta)] \tag{3.13}$$

The SAC adopts a soft Bellman equation to solve for the optimal soft Q value function, resulting in a stochastic neural network policy. In our implementation, the policy network and the critic network are both fully connected neural networks with two hidden layers of 256 neurons. Each time, we collect a batch size of 256 steps for the off-policy model-free RL training and adopt a learning rate of 0.0003.

### 3.3.4 Comparison and Discussion

As described in Section 3.2.1, the experiments are divided into two parts—driving without obstacles and driving with obstacles—since the two sets of tasks have different dimensions of input and different cost functions. In both cases, the autonomous vehicle is randomly initiated from one of the three settings: straight driving, 90° turning, and roundabout entering. We run the proposed GPS-based method and the baselines of the CEM and SAC methods to learn the urban driving policies, and the figures below show the training log of the methods. For GPS and CEM, since the policies are linear Gaussian, we apply a PD controller as the initialization, with large variance. For the model-free RL, the policies are neural networks, so we follow the pure random initialization. Therefore, the initial performance of the GPS and CEM are slightly better compared to the model-free RL methods.

Fig. 3.4 shows the training log of the policy search methods, the GPS and CEM methods on driving tasks without obstacles, and Fig. 3.5 shows the training log of the two methods in tasks with obstacles. In both cases, the GPS method outperforms the CEM in terms of both the speed and the optimum of convergence. The GPS algorithm converges with only

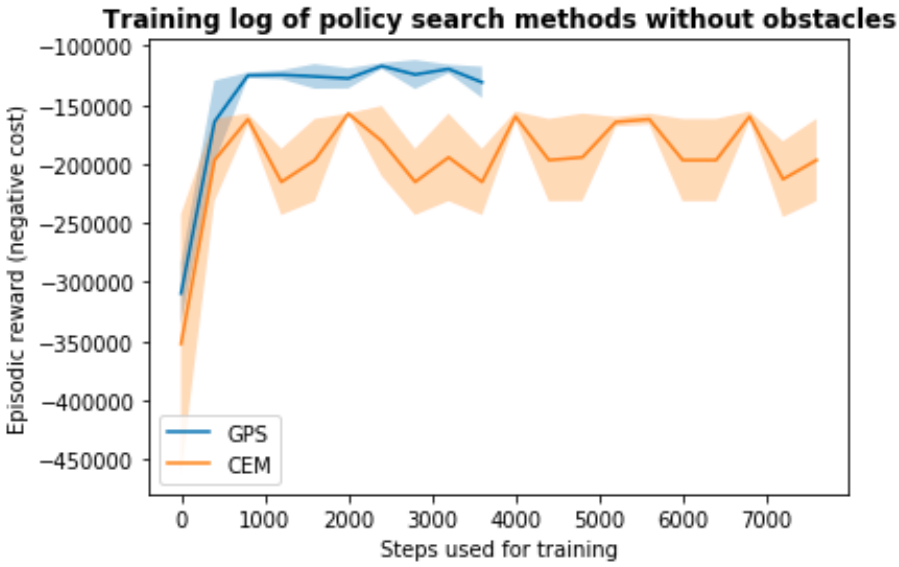


Figure 3.4: The training log of the guided policy search (GPS) and cross-entropy method (CEM) on the driving task without obstacles. The GPS-based method converges faster and to a better driving policy than the CEM.

about 1,000 steps of data, corresponding to 100 seconds of driving, while the CEM takes roughly twice as much time to converge. In terms of qualitative performance, the GPS policy (roughly -100,000 reward) can track the road more stably and can surpass the front vehicle when it affects the speed profile of the autonomous vehicle. The CEM methods (roughly -200,000 reward) can make uncomfortable steering actions and sometimes collide with the obstacle or drive off the road.

Fig. 3.6 shows the comparison of the model-based and policy search methods with the state-of-the-art model-free RL method of SAC. The model-free SAC method achieved a similar performance to our proposed GPS-based method. However, the sample efficiency property of our proposed model-based method is 100 times better, since the SAC takes more than 100,000 steps to converge and GPS only took 1,000 steps to converge. Fig. 3.7 is an example showing the qualitative performance of the GPS policy in a case where the ego vehicle actively changes its lane to surpass the obstacle vehicle.

To summarize, as a model-based RL approach, when applied in urban autonomous driving, the GPS has the advantages of higher sample efficiency, better interpretability, and greater stability. The experiments validate the effectiveness of the proposed method to learn robust driving policy for urban driving in CARLA. We also compare the proposed method with other policy search and model-free RL baselines, showing 100 times better sample efficiency of the GPS compared with the model-free RL baseline. Also, GPS can learn policies for harder tasks that the baseline methods can barely learn.

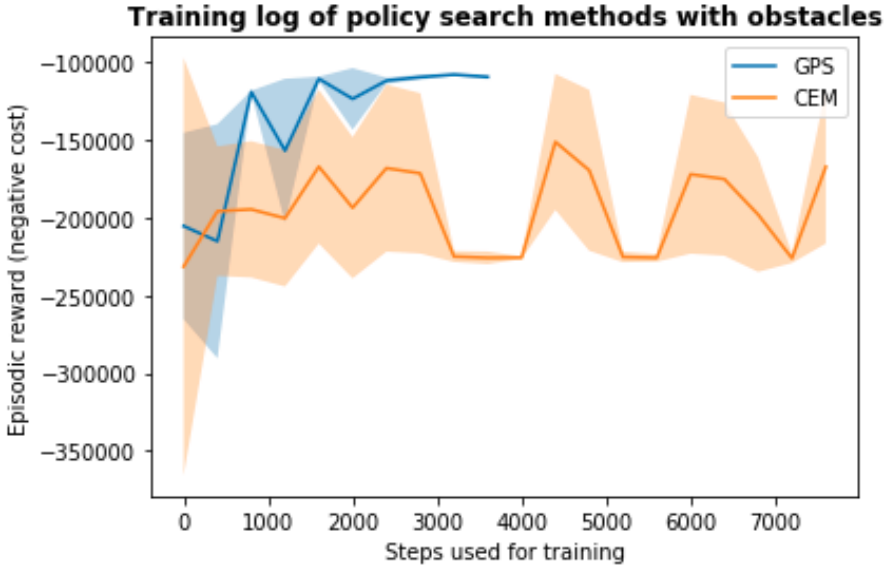


Figure 3.5: The training log of the guided policy search (GPS) and cross entropy methods (CEM) on the driving task with obstacle. Similar to the task without obstacle, the GPS based method converges faster and to a better driving policy compared to the CEM.

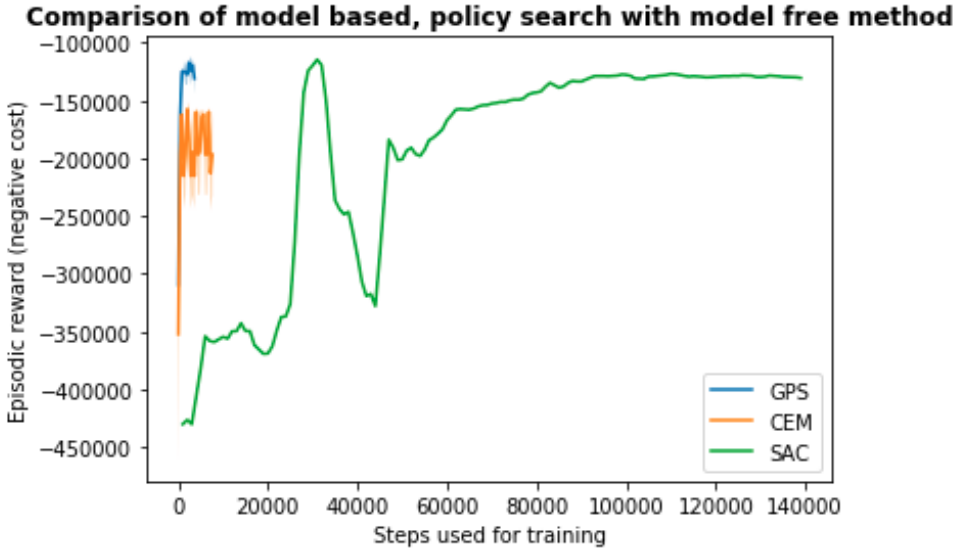


Figure 3.6: The training log of the guided policy search (GPS) and cross-entropy methods (CEM) in comparison to the soft actor critic (SAC) model-free RL method on tasks without obstacles. The model-free RL can learn good driving policy, but it takes more than 100,000 steps of data for training. That is, the model-based method is 100 times more sample efficient.

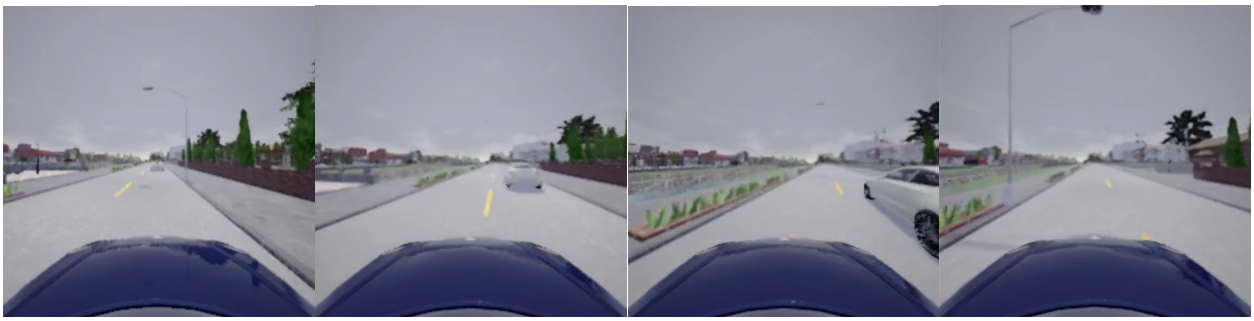

Figure 3.7: Qualitative performance of the GPS policy in a case where the ego vehicle actively changes its lane to surpass the obstacle vehicle. This task involves high-level decision and planning intelligence, which is hard to learn.

## 3.4 Model-free RL: State Representation Formulation

In this section, we investigate the performance comparisons between the usage of different state encoding for the representation of the highly interactive urban driving scenario. As described in section 3.2.1, the CARLA simulator could provide a full-state feature vector of the ego vehicle and the surrounding vehicles, as well as simulated sensor output, such as the front camera and the lidar images. With an end-to-end model-free RL formulation, the encoding of the complicated and interactive driving scenario is very important for understanding the situation and for control of the ego vehicle. We investigate a series of candidate representations and validate that graph neural network (GNN) based representation encoding is the best choice in terms of policy learning performance.

### 3.4.1 Graph Neural Network Representation Formulation

GNN has recently draw significant attention in the research community thanks to its capability of capturing the relational biases between entities in complicated systems [55] [10]. In our autonomous driving task, we also introduce GNN to serve as the representation extractor to encode the interaction information between the traffic participants. The interactive driving scenario is first described using a graph formulation, with vertex states $V_i$ representing each of the traffic participants and edge states $E_{i,j}$ representing the relationship between two traffic participants. Fig. 3.8 shows the graph representation formulation for the driving case. For the sake of simplicity, in our formulation, the edges are directed edges, and only the effect of the surrounding vehicles toward the ego vehicle is considered. That is, there are no directed edges from the ego vehicle toward the surrounding vehicles or between the surrounding vehicles. The vertex state stores the position and velocity of the corresponding vehicle, and the vertex stores the relational distance and velocity difference between the two vehicles.

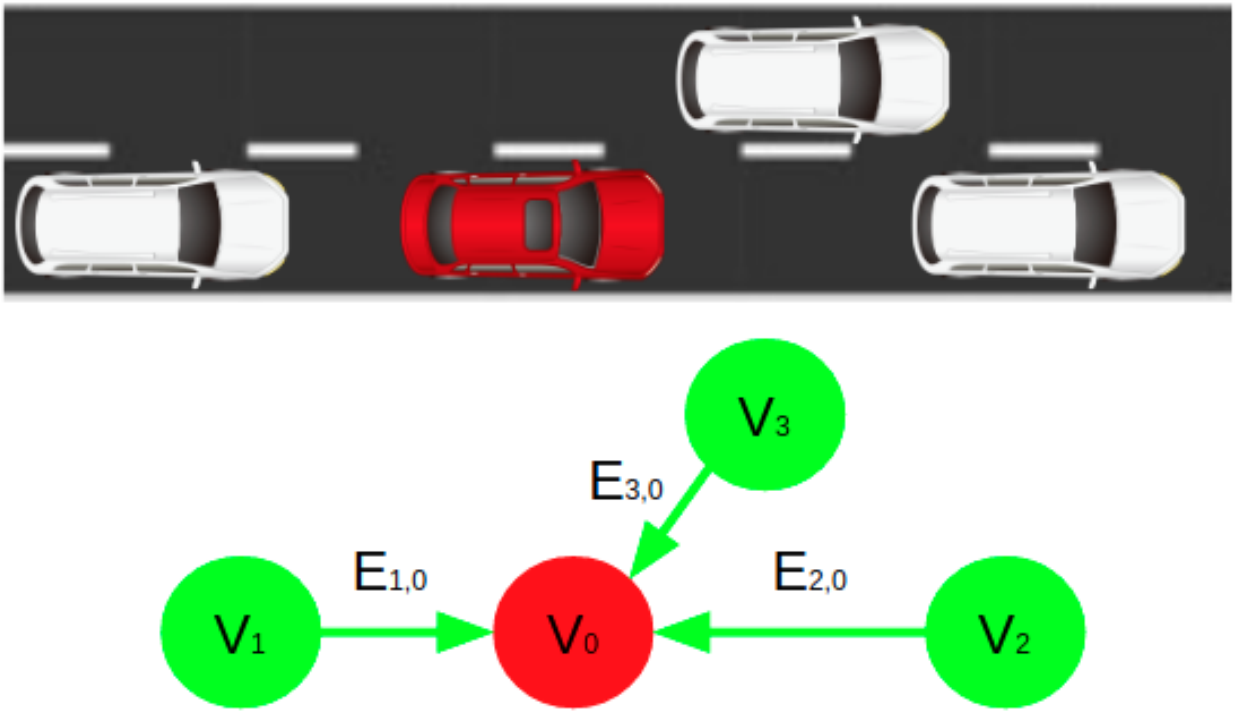


Figure 3.8: The graph representation of the driving scenario with multiple traffic participants.

In the graph neural network, the initial graph formulation is then processed using convolutional graph layers, in which a vertex update operation and an edge update operation are proceeded iteratively. In edge update, the current edge and its starting and ending vertices are fed into a fully connected network to produce the next edge feature. In vertex update, the current vertex and all incoming edges are fed into a fully connected network to produce the next vertex feature. The graph convolutional layer is shown in Algorithm 2

**Algorithm 2** Graph convolutional layer.

Input: kth graph layer $V_i^k, i \in \{0, ..., n\}$, $E_{i,j}^k, (i,j) \in E$, fully connected networks: $\phi_{\theta e}$, $\phi_{\theta v}$
**for** $(i,j) \in E$ **do**
  $E_{i,j}^{k+1} = \phi_{\theta e}(E_{i,j}^k, V_i^k, V_j^k)$
**end**
**for** $i \in \{0, ..., n\}$ **do**
  let $E_i^{k+1} = \Sigma_{(i,j)\in E} E_{i,j}^{k+1}$
  $V_i^{k+1} = \phi_{\theta v}(V_i^k, E_i^{k+1})$
**end**
Return: (k+1)th graph layer $V_i^{k+1}, i \in \{0, ..., n\}$, $E_{i,j}^{k+1}, (i,j) \in E$

### 3.4.2 Comparison between Other Representation Formulations

The proposed graph-based representation formulation is compared with the two following baselines:

**Sensor data:** The CARLA simulator provides rich simulated sensor data, including the front-view camera image and the lidar image, as shown in Fig. 3.2. We concatenate the camera image and the lidar image along the channel axis and feed it into a convolutional neural network feature extractor.

**State vector:** To illustrate the effectiveness of the graph structure, we compare it with a full-state input of all the traffic participants, which is fed into a fully connected network for feature extraction. Since the number of surrounding vehicles can vary, we define a state vector with enough entries for at most 10 vehicles and feed the detectable vehicles to the state vector. If there are not enough vehicles within a detectable distance, we assign 0 to the remaining entries.

We train policy networks using the three state representations using SAC for 35,000 steps until convergence. Fig. 3.9 shows the comparison of the performance with different representations. The training logs show that the GNN-based representation achieves superior performance compared to the sensor data and the state vector. Using the sensor data version representation, SAC cannot learn a driving policy that successfully navigates in the CARLA simulator, while the state vector version policy improves slower than the graph version policy. The final performance is also below that of the proposed graph version policy. We also validated the GNN's performance by varying the number of surrounding vehicles: by training two separate policies within CARLA scenarios—with and without varying number of surrounding vehicles. Training logs in Fig. 3.10 show that the policy trained with a fixed number of surrounding vehicles improves to convergence faster, but the policy trained with a random number of surrounding vehicles reaches similar performance in the end. Therefore, we validate that the GNN representation is helpful for encoding the information within complicated driving scenarios that involve multiple vehicles. Also, the graph formulation is capable of handling a varying number of entities.

## 3.5 Supervised Learning: History Encoding Representation Design

In this section, we investigate the selection of history encoding representation for the application of human behavior prediction in a multistep pick and place task based on supervised learning. We design a novel and interpretable representation formulation to effectively recognize the scene and encode the history information so as to obtain a fast and accurate human behavior prediction module. The task is designed as follows (overview shown in Fig. 3.11): A human worker needs to pick up two objects, A and B. The worker shall put object

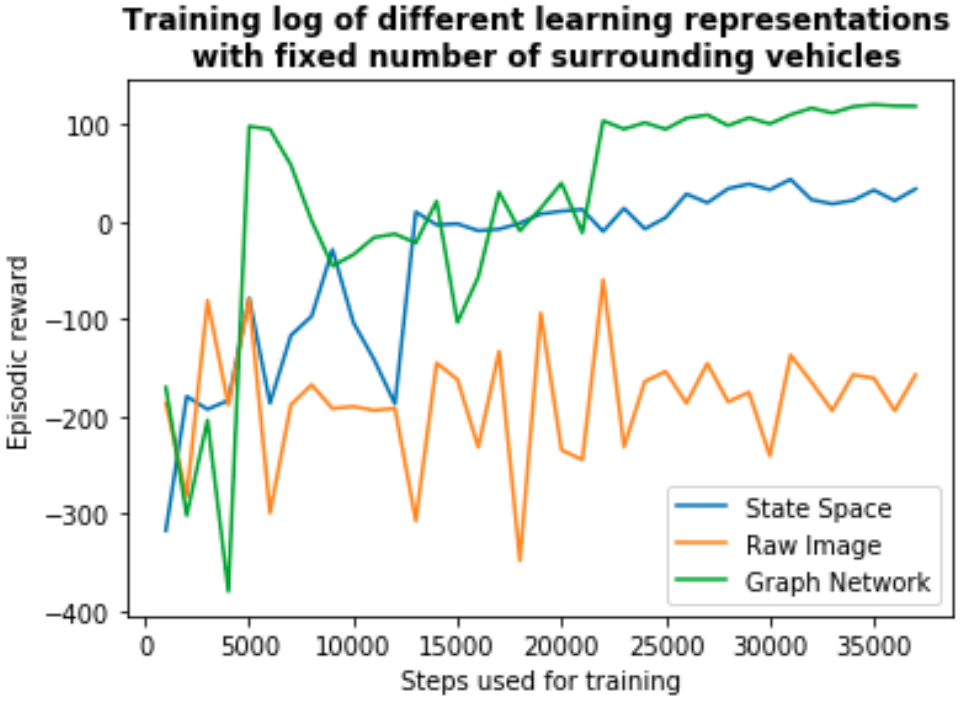


Figure 3.9: Training log comparison of the graph representation and the baseline representations.

A onto either one of the C and E pads and object B onto pad D. The worker has to perform each pick, place, and transport operation one by one, resulting in a sequential operation strategy. We can observe the human performing the task from a third-party-view camera image stream, and the goal of the task is to infer the strategy as soon and as accurately as possible.

To study the behavior prediction methodology, we collect a large video dataset of a human volunteer performing the task, as shown in Fig. 3.12. To finish this task, the human worker has 12 strategies in total; Fig. 3.12 shows one of them. We decompose the whole task into a series of subtasks, such as transporting from one location to another and picking and placing an object. There are 22 total subtasks, and Fig. 3.13 and Fig. 3.14 show two of them.

### 3.5.1 Behavior Prediction Framework and History Encoding Representation

We propose a two-stage behavior prediction framework that is shown in Fig. 3.15. First, the camera image is fed into a VGG convolutional neural network (CNN) [100] classifier to extract a feature for the current operation. We choose the subtask that the human is doing

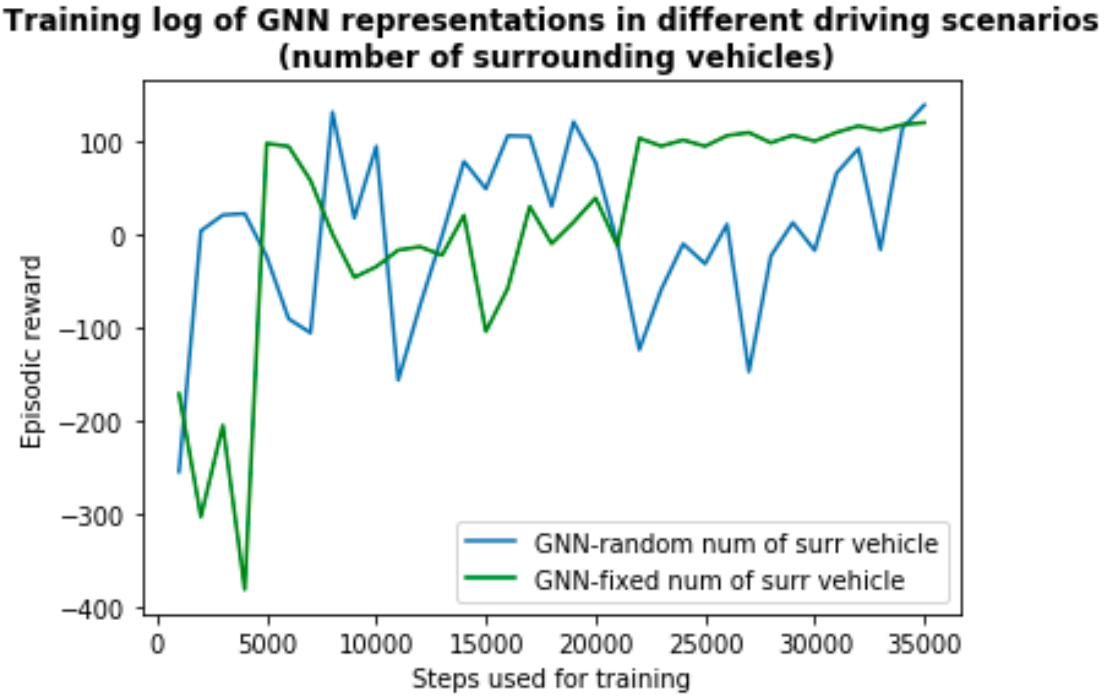


Figure 3.10: Training log comparison of the graph representation policy trained in scenarios with and without varying the number of surrounding vehicles.

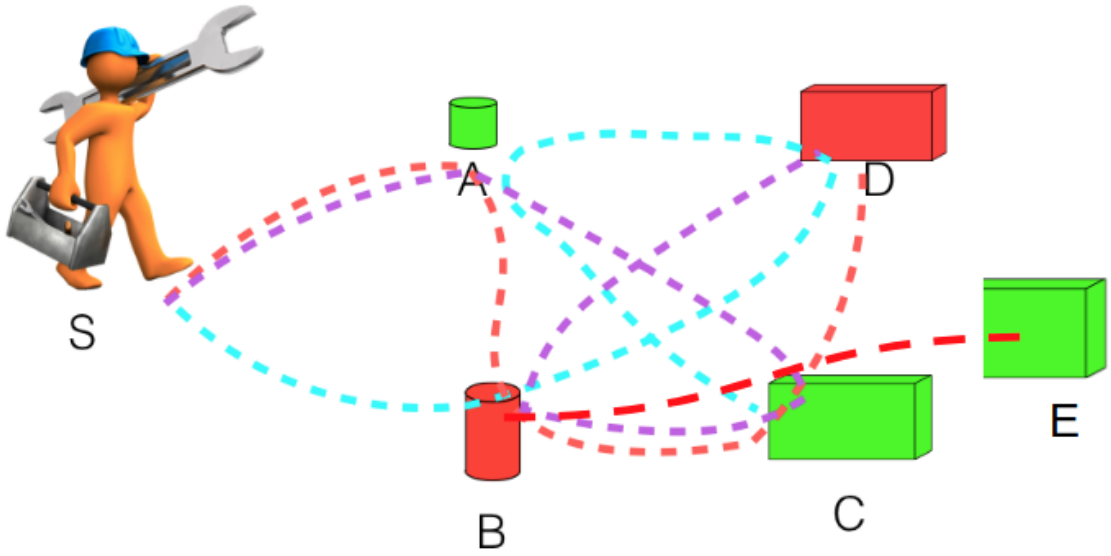


Figure 3.11: Overview of the multi-step pick and place task.

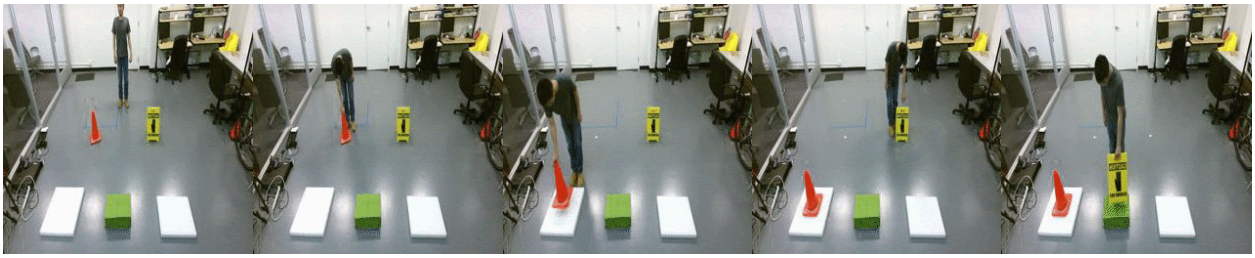

Figure 3.12: One strategy the human worker uses to complete the multistep pick and place task. The human worker first picks the red object and puts it at the left white pad and then picks the yellow object and puts it at the green pad.

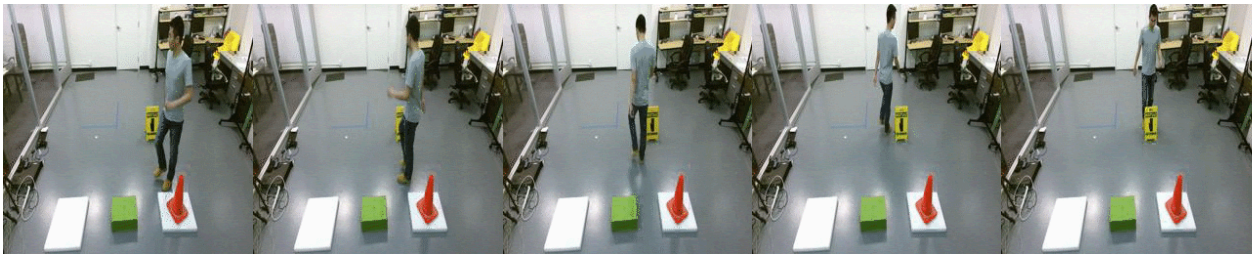

Figure 3.13: One subtask the human worker performs: transport from the right white pad to the yellow object.

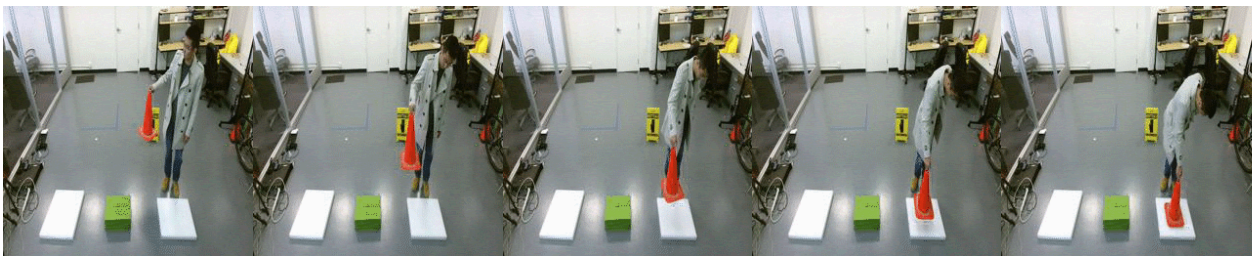

Figure 3.14: One subtask the human worker performs: place the red object onto the right white pad.

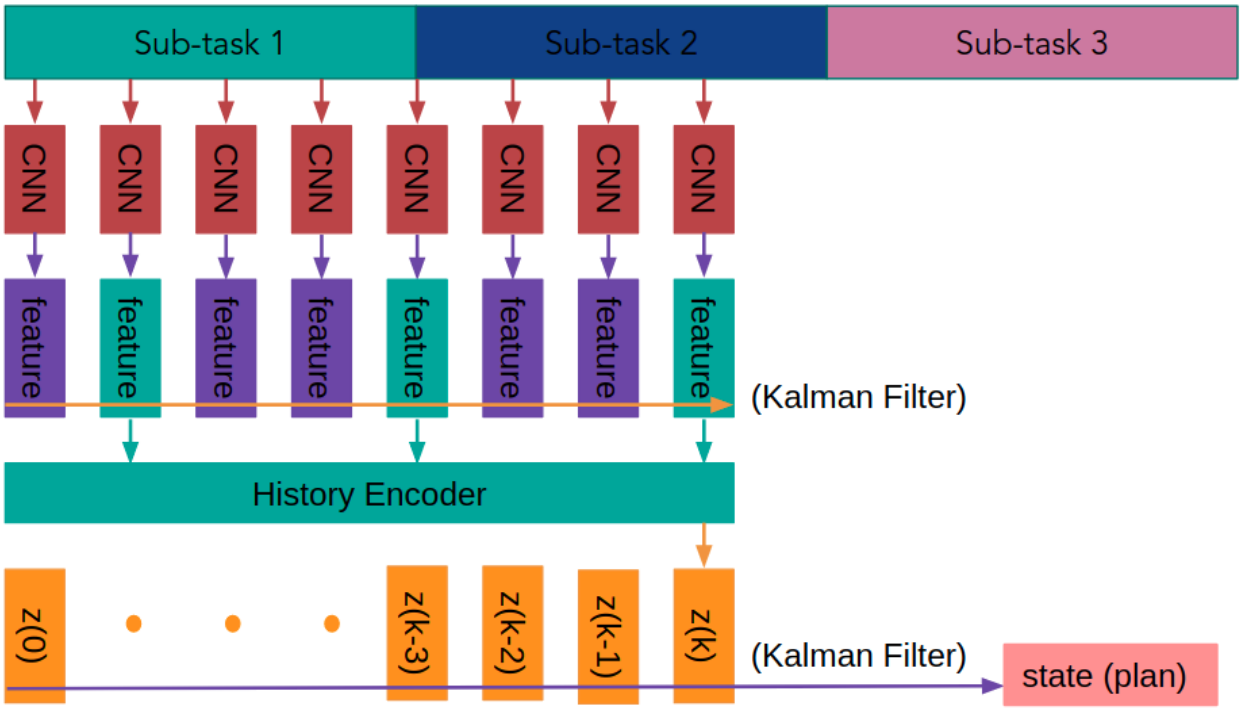


Figure 3.15: The two-stage behavior prediction framework.

in the current frame as the feature; this results in a 22 dimensional one-hot vector encoding the 22 possible subtasks in the current video frame. We then run a steady-state Kalman filter to smooth the feature vectors from the CNN. Then the feature history is encoded into a history encoding representation, which is defined as follows:

The history encoding representation uniformly (sampling time length can be different) selects a fixed number of history features to form the information vector. The fixed number of history features is chosen to be 22, which is the number of dimensions of the feature vector. Therefore, the history encoding representations can be rendered using 22 by 22 square figures, as shown in Fig. 3.16. In Fig. 3.16, the history encoding representations can be interpreted according to a relative time axis and a subtask axis. Concretely, for each row vector with the same relative time, the heat map represents the probability that the human worker is performing one of the subtasks at the relative time. Finally, this history encoding representation is fed into a final neural network to produce the strategy that the human is taking with ground truth strategy label supervision. After acquiring a series of estimations, we run a steady-state Kalman filter to calculate the posterior state estimate. The initial state of the Kalman filter is assumed to be a discrete uniform distribution over all 12 plans.

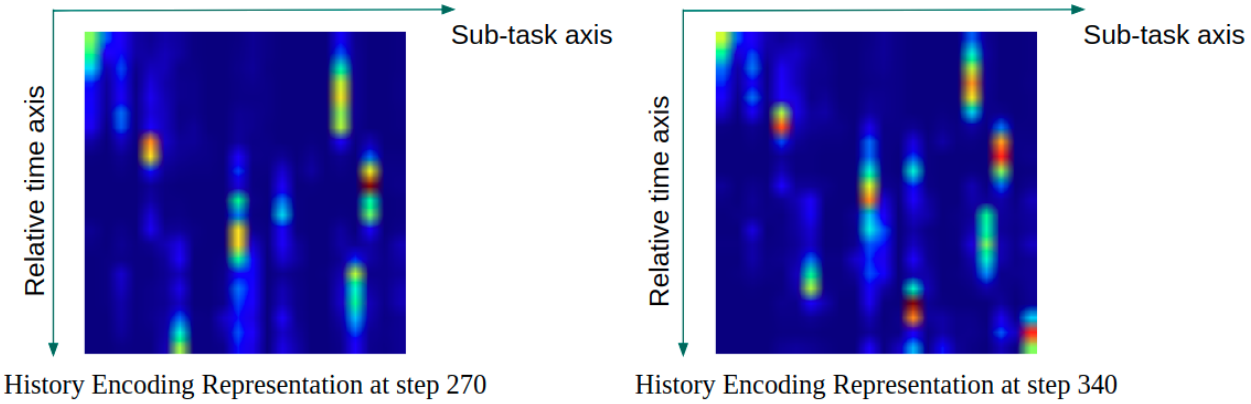


Figure 3.16: Two history encoding representations obtained in one episode at two different time steps.

### 3.5.2 Performance

We collect a large dataset of 50,987 frames of 256 by 256 RGB images recording a human volunteer performing the task. Sample images can be found in Fig. 3.12, 3.13 and 3.14. We randomly separate the dataset into a training set of 46,737 images and 4,250 testing images. We fine-tune a VGG classification network pretrained on ImageNet [33] (with the first four convolutional layers fixed) on the training set (with data augmentation of randomly cropping the images to a size of 224 by 224) and evaluate the results on the testing set. We obtain a classification accuracy on the test set of 51% top 1 accuracy and 75% and 93% top 2 and top 4 accuracy, respectively. The history encoding representations shown in Fig. 3.16 also show clear and contingent subtask classification performance.

The fully connected network that maps from the history encoding representation to the strategy estimation is trained using 12 episodes (one for each strategy) with 4,250 frames of images and tested on another 12 episodes (one for each strategy) with 4,047 frames. We get a 32.4% top 1 classification accuracy and a 85.6% top 5 classification accuracy. The relatively low classification accuracy is because it is not possible to identify the correct strategy in the early stage of the episode. For prediction performance, the model achieves a 100% final classification success rate. During online prediction, the model can achieve human-level performance by converging quickly to the correct strategy once enough evidence is observed. Fig 3.17 and Fig. 3.18 show the behavior prediction model performance in an online episode. We can see that in the first half of the episode (Fig 3.17), when the human volunteer picks the red object and the yellow object, the prediction model cannot determine which object the human is placing first or which strategy the human is taking. Therefore, the thick red probability curve, which corresponds to the correct strategy, is just as high as the thin red curve, which corresponds to a wrong but indistinguishable strategy. In the second half of the episode (Fig 3.18), as the human approaches the right white pad and places the red object,

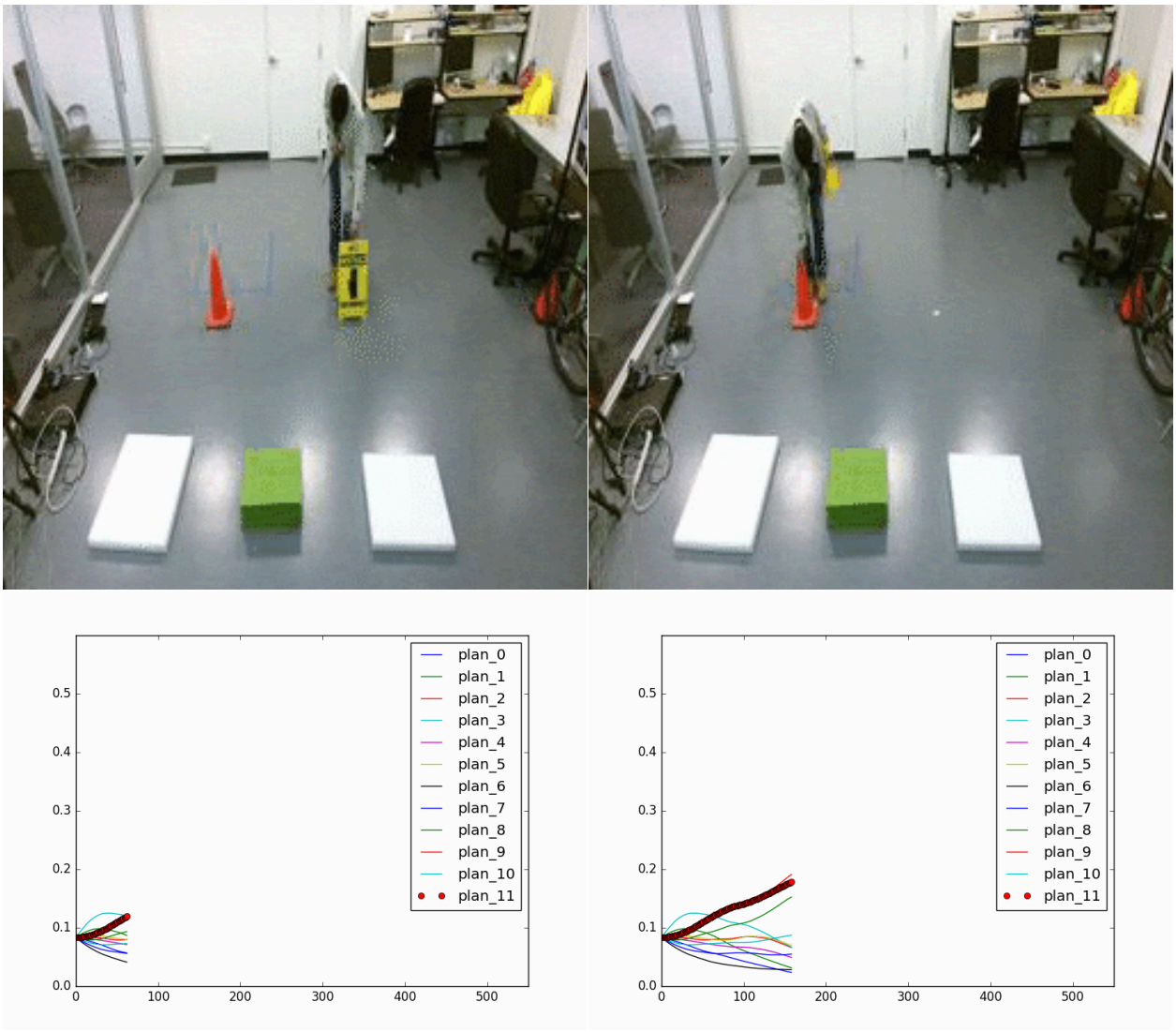


Figure 3.17: First half of the behavior prediction model performance in an online episode.

the model is able to correctly identify the strategy, and the thick red curve rises above the thin red curve, finally reaching to far above all other curves. The experiments show that the history encoding representation and the prediction model are able to capture the history information and achieve satisfying prediction performance.

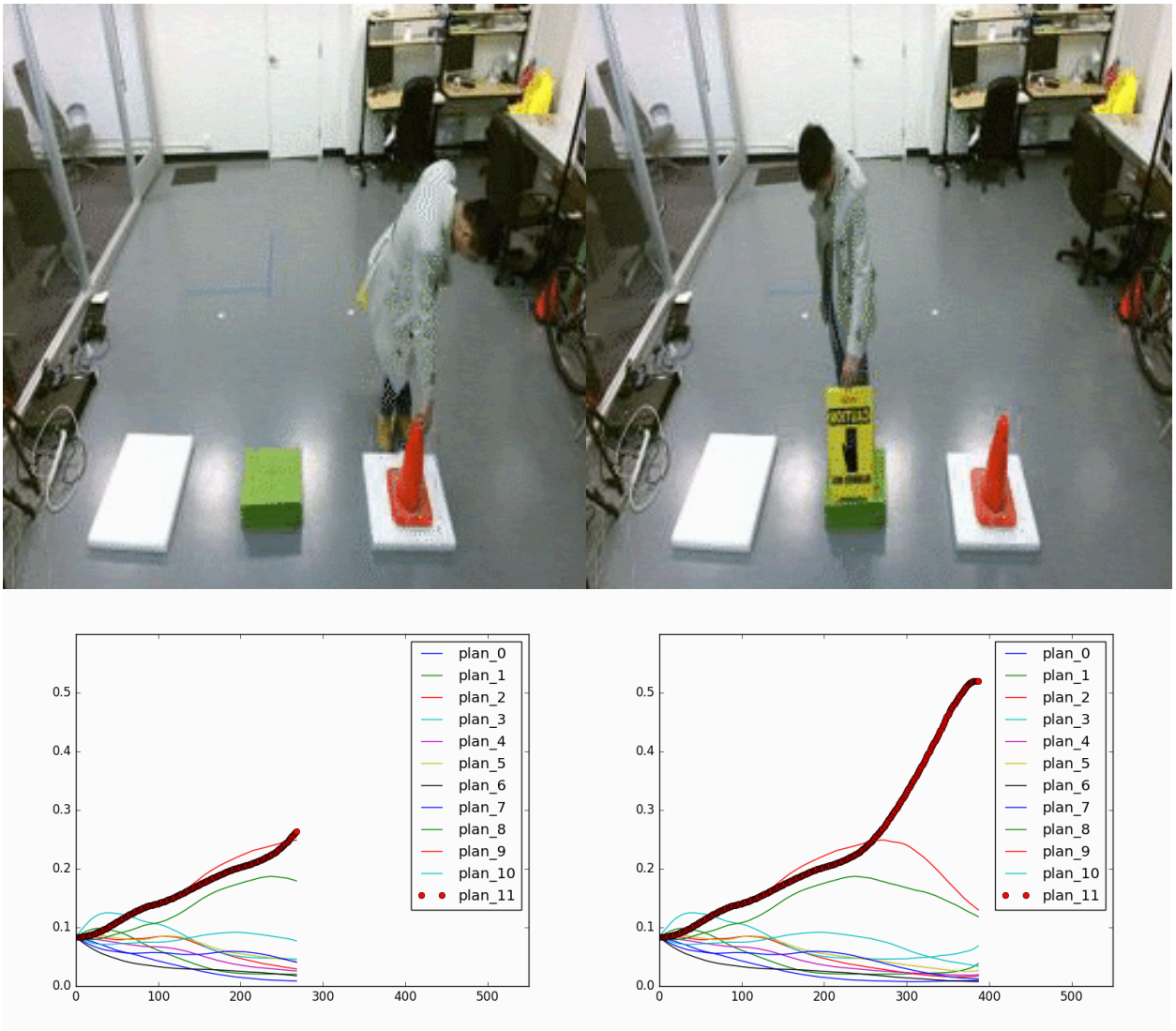


Figure 3.18: Second half of the behavior prediction model performance in an online episode.

# Part II

# Reasoning for Representations at the I/O of the LbC Policies

# Chapter 4

# Cascade Attribute Network

It has been shown that reinforcement learning (RL) methods can train control policies to achieve great success in various automation tasks. However, a main challenge of the wider application of RL in practical automation is that the training process is usually difficult. What makes it worse is that pretrained policy networks are rarely reusable in other similar cases, making it necessary to retrain a new policy network for each new task from scratch. A training framework that can effectively solve a class of similar control tasks should be developed. In this chapter, we propose the cascade attribute network (CAN), which utilizes its hierarchical structure to decompose a complicated control policy in terms of the requirement constraints (what we call attributes) encoded in the control task. The CAN training process is divided into two parts: the training of the base goal-reaching attribute module and the add-on attribute modules that reflect the add-on requirements. Then the attribute modules are connected in a series to provide the overall outcome of the control policy. We tested the effectiveness of our proposed method on two robot control scenarios with various add-on requirements in time, position, velocity, and acceleration phases. The experiment results show that the CAN is capable of learning a robust control policy using the proposed step-by-step procedure, and its training efficiency outperforms the baseline RL train-from-scratch significantly. It is also shown that for some control tasks with more than one add-on attribute requirement, by directly assembling the attribute modules in cascade, the CAN can provide ideal control policies in a zero-shot manner.

## 4.1 Introduction

Reinforcement learning is an artificial intelligence approach that solves for automatic control policies $\pi(a|s)$, usually neural networks that map the state space inputs $s$ to the control command outputs $a$. The training of the RL policies is through the interaction of the agent and the environment, which is often modeled using a Markov Decision Process (MDP) [104]. The RL has been successful in solving many robotics and automation problems in simulations as well as in real-world scenarios [61][96][63][49][51]. However, the wide application of RL

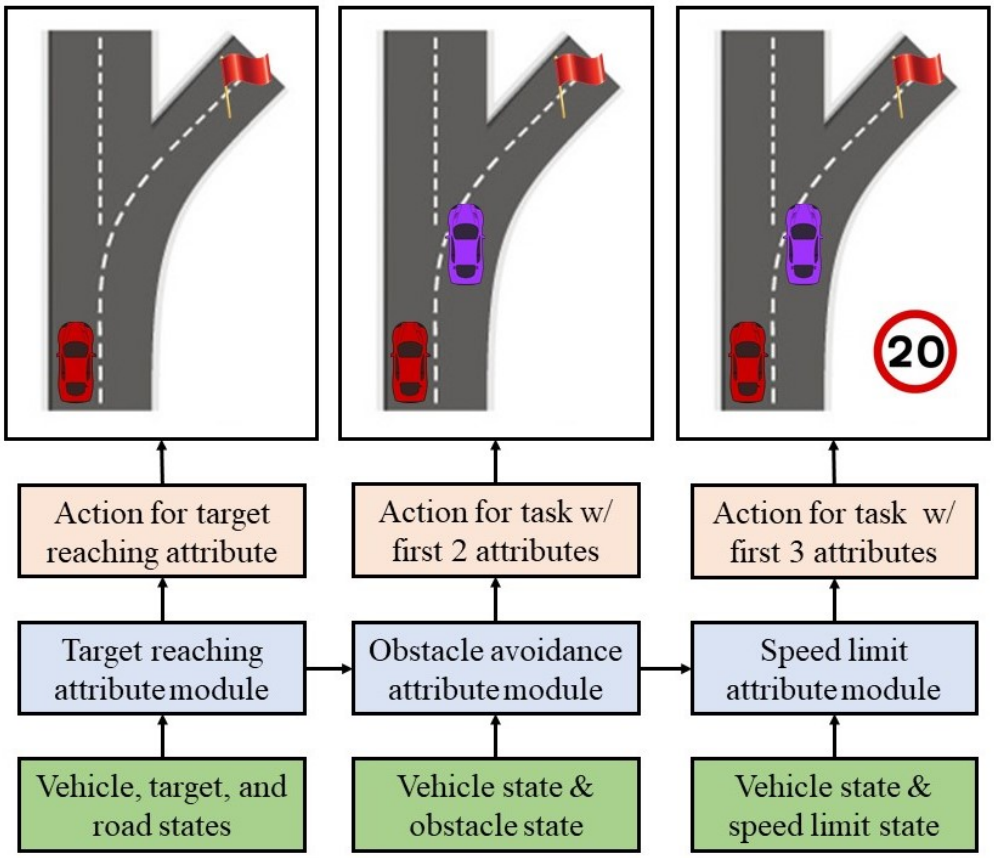


Figure 4.1: Autonomous driving as an example of decomposing a complicated control task into multiple attribute constraints using the cascade attribute networks (CAN).

in automation is slowed down by several challenges, and one of the main challenges is the difficulty of RL policy training. RL policy generally takes a great amount of computation power to train for complicated control tasks. What makes it even worse is that RL policies are optimized based on certain fixed MDPs, and the knowledge encoded inside an optimized control policy are hard to transfer to other similar tasks. That is, new RL policies—even for those control tasks that are very similar to the pretrained task—have to be trained from scratch, which leads to a great amount of computation power waste.

For example, consider an autonomous driving scenario, as shown in Fig. 4.1. The target-reaching task for RL is to train the control policy that can make the autonomous vehicle travel from an origin position to a target position. It takes a great computation cost to optimize for the control policy of the target-reaching task. And, in reality, the task is slightly different when the autonomous vehicle has to avoid obstacle vehicles. Since the control policy is a neural network and agnostic of the hidden layers of the policy network, the knowledge encoded inside the policy network of the pure target-reaching task cannot

transfer to new tasks, even if they are very similar to the learned one. Also, since the input-output architecture of a neural network policy is fixed, the information for the newly added obstacle has no way to enter the old policy network. Therefore, the old policy network is not useful at all, and a new policy network has to be trained from scratch for the new control task. For instance, imagine adding a further speed limit requirement to the autonomous driving task. The learned knowledge in the two previous tasks cannot be transferred to the new task, and there is no input entrance for the speed limit information in the pretrained policy network. Therefore, RL frameworks with fixed policy models cannot address differing tasks in such varying environments. Of course, it would be ideal if the training framework made it possible to transfer control knowledge between similar tasks.

There have been many prior attempts to create versatile intelligence that adjusts to task changes. Transfer learning [106, 78] is a key tool in using previously learned knowledge for the better or faster learning of new knowledge. Rusu et al [92][93] establish a multi-column (network) framework, referred to as progressive network, in which newly added columns are laterally connected to previously learned columns for knowledge transfer. Drafty et al [30] and Braylan et al [19] also design interesting network architectures for knowledge transfer in micro aerial vehicle (MAV) control and video game playing. Barreto et al [9, 8] propose a scheme based on successor features: a value function describes the dynamics of the environment from reward functions and a generalized policy improvement operation with consideration of multiple policies. In this way, the method can provide an efficient transfer among different tasks where the reward function is changed but the dynamics of the system remain the same. As for the combinations of transfer learning and imitation learning, Ammar et al [3] use unsupervised learning to map states for transfer, assuming the existence of distance function between different state spaces. Gupta et al [44] learn an invariant feature between different dimensional states and use demonstrations to increase the density of the rewards. There are other methods seeking to learn a globally general policy. For instance, meta learning [109] attempts to build self-adaptive learners that improve their bias through accumulating experience. One-shot imitation learning [36], for example, is a meta learning framework that is trained using a number of different tasks so that new skills could be learned from a single expert demonstration.

Unlike the prior transfer learning approaches that lack interpretable transferable features, we put emphasis on the modularization of attributes, which are concrete and meaningful modules that can be conveniently assembled into various combinations. Concretely, we propose to decompose the complicated control problem in terms of its requirement constraints, which we refer to as attributes. The concept attributes refer especially to global characteristics or requirements that take effect throughout the task. Take the example in Fig. 4.1: To solve an autonomous driving problem, one first decomposes the requirements of the task into a base target reaching attribute—the add-on obstacle avoidance and speed limit attributes—and then trains the attribute module for each of the attributes and finally assembles the attribute modules together to produce the overall policy. Therefore, our work is different from other research in training modular neural networks. For example, [34] investigates the combinations of multiple robots and tasks, while [4] investigates the combinations

of multiple sequential subtasks; we look into modularization in a different dimension. We investigate the modularization of attributes, the characteristics or requirements that take effect throughout the whole task.

In order to decompose and assemble the attribute modules, we propose a simple but efficient RL framework called the cascade attribute network. In CAN, the attribute modules are connected in cascade series. Each attribute module receives both the output of its preceding module and the related states of the corresponding attribute and then returns the action that satisfies all the attributes ahead of it. Decomposing a task using a series of attributes has two main intriguing advantages:

1. The decomposed attribute modules are much easier and faster to train compared with the overall control policy.

2. The attribute-related knowledge is encoded in interpretable modules, which can be built up to create versatile policies that can adjust to changes in the control tasks.

## 4.2 Reinforcement Learning and Curriculum Learning Backgrounds

### 4.2.1 Proximal Policy Optimization

The objective of RL is to maximize the expected sum of the discounted rewards $R_t = \mathbb{E}\sum_{k=0}^{\infty}\gamma^k \cdot r_{t+k}$ in an agent-environment-interacting MDP. The agent observes state $s_t$ at time $t$ and selects an action $a_t$ according to its policy $\pi_\theta$ parameterized by $\theta$. The environment receives $s_t$ and $a_t$ and returns the next state $s_{t+1}$ and the reward in this step $r_t$. The $\gamma$ in the objective function is a discounting coefficient. The main RL approaches include deep Q-learning (DQN) [72], asynchronous advantage actor critic (A3C) [71], trust region policy optimization (TRPO) [98], and proximal policy optimization (PPO) [97]. Approaches used in continuous control are mostly policy gradient methods (i.e. A3C, TRPO, and PPO). The vanilla policy gradient method updates the parameters $\theta$ by ascending the log probability of action $a_t$ with higher advantage $\hat{A}_t$. The surrogate objective function is

$$L(\theta) = \hat{\mathbb{E}}_t \left[ \log \pi_\theta(a_t \mid s_t) \cdot \hat{A}_t \right] \tag{4.1}$$

Although A3C uses the unbiased estimator of the policy gradient, large updates can prevent the policy from converging. TRPO introduces a constraint to restrict the updated policy from being too far in Kullback-Leibler distance [57] from the old policy. Usually, TRPO solves an unconstrained optimization with a penalty punishing the KL distance between $\pi_\theta$ and $\pi_{\theta_{old}}$, specifically,

$$L(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \cdot \hat{A}_t - \beta \cdot \mathrm{KL}\left(\pi_\theta, \pi_{\theta_{old}}\right) \right] \tag{4.2}$$

However, the choice of the penalty coefficient $\beta$ has been a problem [**ppo**]. Therefore, PPO modifies TRPO by using a simple clip function parameterized by $\epsilon$ to limit the policy update. Specifically,

$$L(\theta) = \hat{\mathbb{E}}_t \left[ \min \left( \frac{\pi_\theta}{\pi_{\theta_{old}}} \hat{A}_t, \mathrm{clip} \left( \frac{\pi_\theta}{\pi_{\theta_{old}}}, 1+\epsilon, 1-\epsilon \right) \hat{A}_t \right) \right] \tag{4.3}$$

This simple objective turns out to perform well while enjoying better sample complexity; thus, we are using PPO as the default RL algorithm in our policy training. We are also inspired by [48] to build a distributed framework with multiple threads to speed up the training process.

The advantage function $\hat{A}_t$ describes how better a policy is versus a baseline. Traditionally the difference between the estimated Q value and value functions is applied as the advantage [71]. Recently Schulman et al [96] use generalized advantage estimation (GAE) to leverage the bias and variance of the advantage estimator.

### 4.2.2 Curriculum Learning

Curriculum learning (CL) [14] trains a model on a sequence of cognate tasks that get more and more challenging gradually so as to solve hard tasks that could not be learned from scratch. Florensa et al [41] apply reverse curriculum generation (RCL) in RL. In the early stage of the training process, the RCL initializes the agent state to be very close to the target state, making the policy very easy to train. They then gradually increase the random level of the initial state as the RL model performs better and better. Sanmit et al [76] formulate the curriculum learning sequence as an MDP process, which can also be learnt from experience. These results show that with a good consideration of the nature of the problem, CL can help RL have a better convergence speed. We adopted the idea of CL in our training strategy and achieved improvements in terms of the performance of the learned policies.

## 4.3 Multi-Attribute Problem Formulation

We consider an agent performing a class of tasks built up with one base attribute and a series of add-on attributes. We model the control tasks and the attribute in a unified form using MDPs. Since the agent is fixed, its action space is a fixed space, which we call $A$. We denote the attributes using their index $\{0, 1, 2, \ldots\}$, where the $0^{th}$ attribute is the base attribute, which usually corresponds to the most fundamental goal of the task, such as the target-reaching attribute in the autonomous driving task. We define the state space of

each attribute to be the minimum state space that fully characterizes the attribute, denoted $S = \{S_0, S_1, S_2, S_3 \ldots\}$. For example, let the base attribute be the target-reaching attribute and the $1^{st}$ attribute be the obstacle avoidance attribute. Then $S_0$ consists of the states of the agent and the target, while $S_1$ consists of the states of the agent and the obstacle, but does not include the states of the target. Using the attribute as elements, one can build various control tasks, such as pure target reaching, target reaching while avoiding an obstacle, target reaching under external force influence, and so on. There is indeed overlapping between the state space of different attributes, and the union of two state spaces $S_i \cup S_j$ is the state space of the assembled control task consisting of attribute $i$ and $j$.

Each attribute has a unique reward function as well, denoted $R = \{R_0, R_1, R_2, R_3 \ldots\}$. Each $R_i$ is a function mapping a state action pair to a real number reward (i.e., $R_i : S_i \times A \to \mathbb{R}$). Similarly, there is a specific transition probability distribution for each attribute, denoted: $P = \{P_0, P_1, P_2, P_3 \ldots\}$. And for each attribute, its transition function takes in the state action pairs and outputs the states for the next timestep—that is, $P_i : S_i \times A \to S_i$.

Our goal is to develop a framework that can easily train the policy networks for the tasks consisting of a series of attributes, especially those consisting two attributes (one base attribute and one add-on attribute). To avoid training each task from scratch, we aim to learn reusable attribute-related knowledge encoded in the attribute modules. A key characteristic in our problem formulation is that the state spaces for different attributes can be different. This formulation enables the assembled attribute network to dynamically manage the state space of the task. Specifically, the state $s_i$, is fed to the attribute module corresponding to the $i^{th}$ attribute. Although the input of the attribute module can be naturally defined, the challenging part is how to define the output of the attribute modules and how to get the overall policy output from the outcome of the attribute modules. And this design is provided in the following subsection.

## 4.4 The Cascade Attribute Networks

The architecture of the CAN is shown in Fig. 4.2. The training for a task with more than one attribute is divided into two parts: the training of the base attribute module and the add-on attribute module. In the training phase, first a RL policy $\pi_0(a_0|s_0)$ is trained to accomplish the goal of the base attribute. The base attribute network takes in $s_0 \in S_0$ and outputs $a_0 \in A$; the reward and transition functions of the MDP are given by $R_0$ and $P_0$. This process is a default RL training process.

Consider the control task with more than one attribute, without loss of generality; the $1^{st}$ add-on attribute module is trained next, in series of the base attribute module. The $1^{st}$ attribute module consists of a compensation network and a weighted sum operator. The compensation network is fed with state $s_1 \in S_1$, and action $a_0$ chosen by $\pi_0$. The output of the compensation network is the compensation action $a_1^c$, which is used to compensate $a_0$ to produce the overall action $a_1$. The reward for the MDP is given by $R_0 + R_1$ so that the requirements for both constraints of attributes are satisfied. Since the parameters of

the base attribute network are pretrained and fixed, the cascading attribute network would extract the features of the attribute by exploring the new MDP under the guidance of the base policy. It is noted that in the add-on attribute module, the weight of the compensation action $a_1^c$ is initially small, then increases over the training time. That is, at the early stage of the training process, mainly $a_0$ takes effect, while $a_1^c$ gradually gets to influence the overall $a_1$ as the training goes on. For the $2^{nd}, 3^{rd}, \cdots$ attributes, the training of their attribute networks is the same.

By decomposing the control policy into base and add-on attribute modules, the proposed CAN architecture is especially efficient for the training of those control tasks with two attributes. Benefitting from the decomposition, the high dimensional control task is reduced to lower dimensional attributes, which are much easier for the RL algorithms to train. The inference model for those tasks with one base attribute and one add-on attribute is the same as the training structure shown in Fig. 4.2. For those tasks with more than one add-on attribute, one approach is to combine all the add-on attributes into one complicated add-on attribute and apply the same method as described. Moreover, empirical results also show that for cases where the number of add-on attributes is small and different add-on attributes do not entangle with each other, the CAN shown in Fig. 4.3 can provide ideal control policy. In the CAN shown in Fig. 4.3, the $i^{th}$ attribute module takes in $s_i$ and $a_{i-1}$, and it outputs $a_i$, which satisfies all the attributes before the $i^{th}$ module. The final output $a_j$ is the overall output that satisfies all the constraints in the constraint array.

## 4.5 Experimental Setup

We carry out a series of experiments to validate the capability to decompose and assemble attributes in multi-attribute tasks and to take advantage of the CAN to learn complicated tasks. The experiments are powered by the MuJoCo physics simulator [108]. The attribute modules in our experiments are all three-layer fully connected networks that output Gaussian distributed stochastic actions, built using TensorFlow [1]. The baseline RL algorithm we use is the PPO [97] method with GAE [96] as the advantage estimator.

We evaluate the capability of the CAN on two types of robot scenarios in our experiments. One is a point robot with a two-dimensional action space, and the other is an articulated robot with a five-dimensional action space. For the point robot, the state space includes the position and velocity of the robot, and the action vector is the driving force applied to it. For the articulated robot, the state spaces are the angle and angular velocity at all the joints and the x-position and speed of the base of the robot, while the action vector includes the torques at all the joints and the force applied to the base of the robot. For each robot agent, we implement one base attribute and four different add-on attributes. For each different attribute, we designed a reward function $R_i$. The attribute settings and reward functions are described as follow:

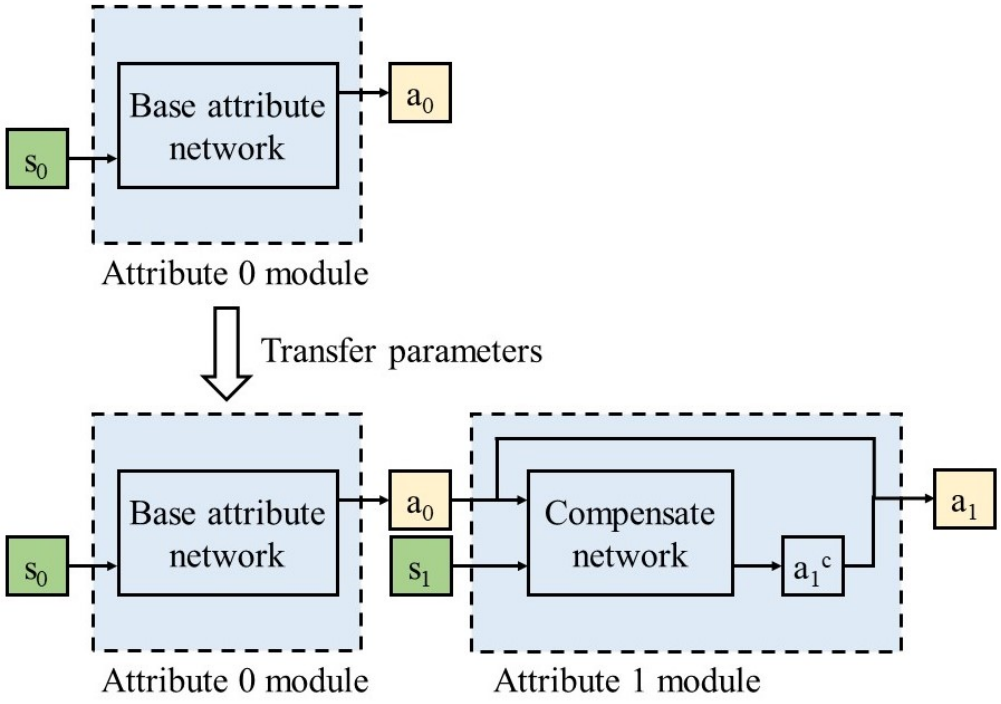


Figure 4.2: The training procedure of an attribute module in the CAN: First train the base attribute module, then train the add-on attribute module based on the fixed pretrained base module.

### 4.5.1 Reaching Attribute (Base Attribute)

The target-reaching task, a common robot task, is defined to be the base attribute for both robot scenarios in our experiment. Given the initial configuration and location of the target, the robots' aim is to reach the target zone using their end effector. As for the reward function of the base attribute, a positive reward is given to the agent only when the robot reaches the required target area. The reward function is shown below:

$$r_b(t) = \begin{cases} 1, \textit{ reaches target} \\ 0, \textit{ other case} \end{cases} \tag{4.4}$$

### 4.5.2 Obstacle Attribute (Position Phase)

In our obstacle avoidance attribute, a circular obstacle is placed in the work area—more specifically, right in the way of the target-reaching path. Furthermore, the obstacle in this experiment is not static, but moving around its initial position randomly, which means that it is necessary for the policy to include the ability to deal with dynamic obstacles. In our implementation, one obstacle avoidance attribute corresponds to one circular obstacle.

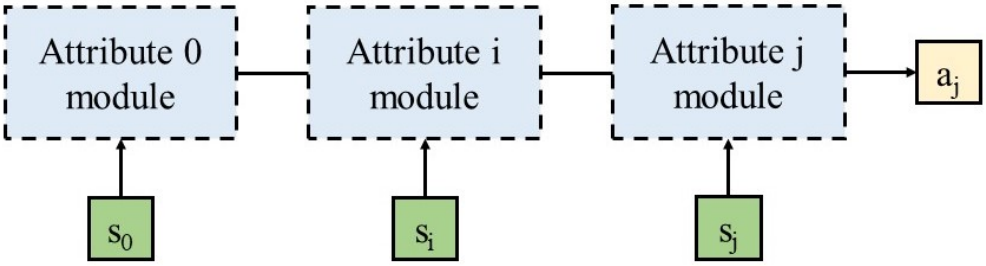


Figure 4.3: One kind of usage of the CAN in multi-add-on-attributes tasks: By assembling add-on attribute modules in cascade to the base attribute module, the output action of the last attribute module is the outcome of the overall hierarchical policy network.

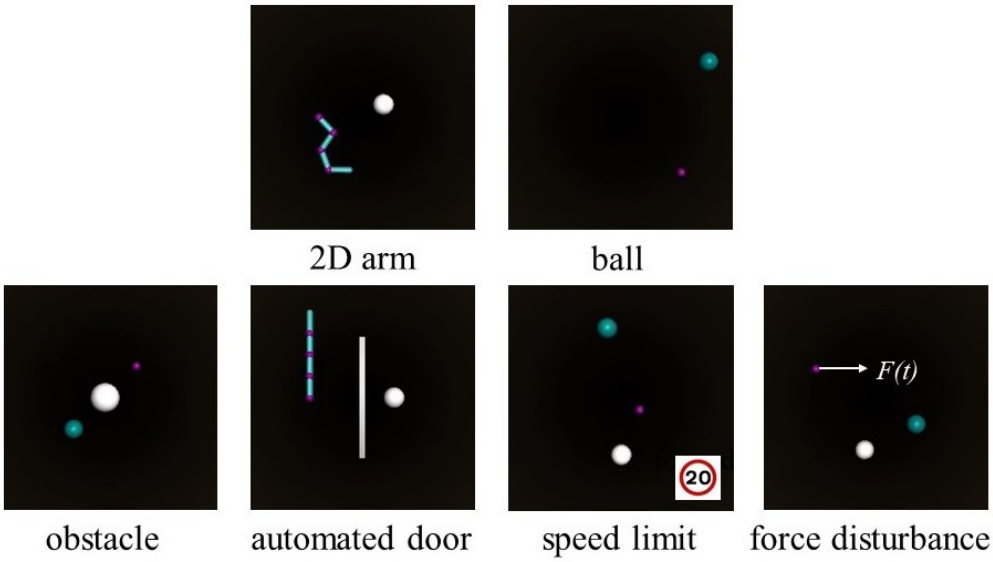


Figure 4.4: The images in the top row show the two robot scenarios, in which the agents are performing the base attribute of target reaching. The bottom images show the four add-on attributes.

Nevertheless, many obstacle attributes are allowed to be assembled together to create a more complicated environment. For each obstacle, whenever the robot collides with it, the MDP will give the robot a negative reward. The reward function is represented as the following:

$$r_o^i(t) = \begin{cases} -0.3, \textit{ touches } i^{th} \textit{obstacle} \\ 0, \textit{ other case} \end{cases} \tag{4.5}$$

### 4.5.3 Automated Door (Time Phase)

The automated door attribute is a time-controlled obstacle that is almost impossible for the robots to pass when it is closed. This door is closed at first and will open at some specific time slots. This attribute is challenging to train using RL since it punishes the agent even if it goes in the right direction at a wrong time. If the reward function is not designed properly, the RL algorithm can accumulate significant negative rewards for the right direction, making it impossible for the robot to learn to move in the right direction. Therefore, the negative reward for hitting the door is designed to be relatively small so the robot does not get discouraged even after receiving the negative signals many times.

$$r_d(t) = \begin{cases} -0.01, \textit{ touches door} \\ 0, \textit{ other case} \end{cases} \tag{4.6}$$

### 4.5.4 Speed Limit (Velocity Phase)

Practical robots always have dynamic constraints on their joints, which limit their speed. Also, for safety reasons, robots' speed must be limited. We define the speed limit attribute in this experiment by defining the reward function to give a punishment once the speed of the robot's joints exceeds the speed limit. In order to make the limitation more realistic, the speed limit at time t, L(t) is designed to be a time-variant function, as shown in Fig 4.5. Denoting the maximum joint velocity as $v_{max}$, the reward function can be written as:

$$r_s(t) = -0.3(max(v_{max} - L(t), 0)) \tag{4.7}$$

### 4.5.5 Force Disturbance (Acceleration Phase)

In working scenarios, robots can be influenced by force disturbance or repulsion force on their joints. The force disturbance attribute in our experiment corresponds to a time-invariant force disturbance added to a certain joint of both point robot and articulated robot. However, the force limitation for actuation is changed, which means the robot needs to find a way out other than simply compensating this force in each joint. The force disturbance affects only the dynamics function of the robot system, with no additional reward function added.

$$r_f(t) = 0 \tag{4.8}$$

## 4.6 Training Schemes

To guarantee the capacity of the CAN, the attribute modules need to meet two requirements:

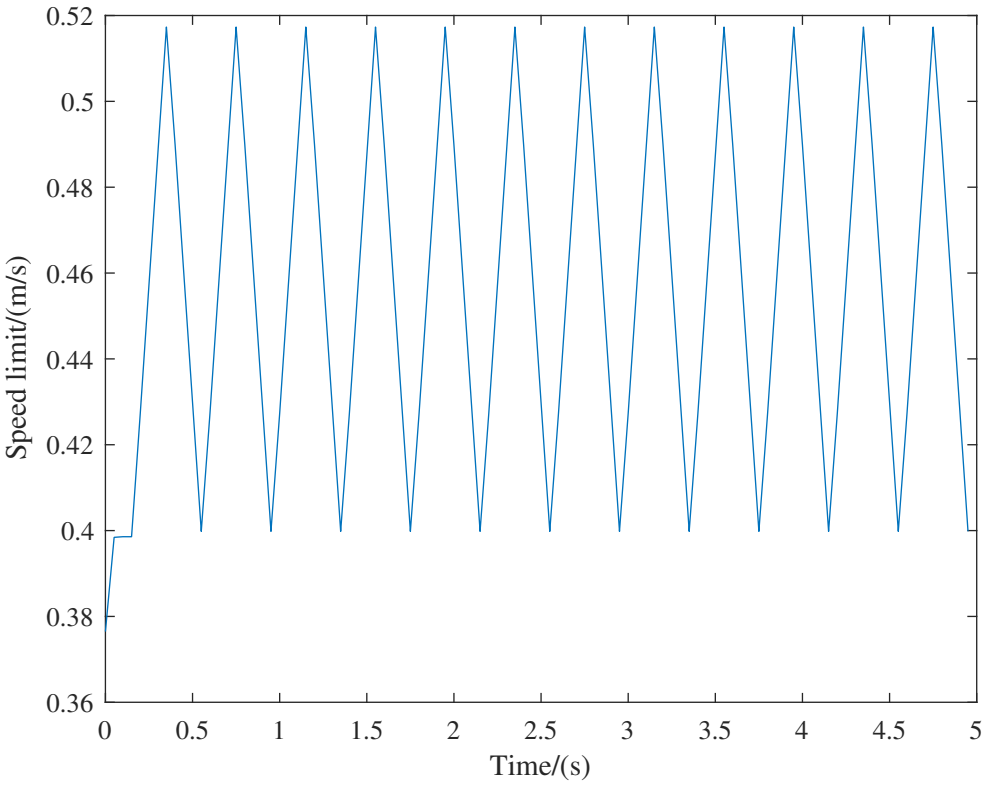


Figure 4.5: The time-variant speed limit function applied in the speed limit attribute.

1. The base attribute policies should be robust over the state space, rather than being effective only at the states that are close to the optimal trajectory. This enables the base attribute policies to be instructive when a compensation action is added on the top of it.

2. The compensation action for a certain attribute should be close to zero if the agent is in a state where this attribute is not active. This property increases the capability of multi-attribute structures.

For the sake of the robustness of the attribute policies, we apply CL to learn a general policy that can accomplish the task starting from any initial state. The CL algorithm first trains a policy with a fixed initial state. As the training goes on, the random level of the initial state is gradually increased until the initial state is randomly sampled from the whole state space. The random level is increased only if the policy is capable enough for the current random level, which is reflected by the increase of the episodic reward.

**Algorithm 3** Curriculum learning.

$RandomLevel$ = Initial Random Level
$\lambda = 1+$ Random Level Increase Rate
$N$ = Batch Number
$LongTermR = Queue()$
**while** $RandomLevel < Terminal\ Random\ Level$ **do**
    Update the policy using PPO
    $Rewards \leftarrow RunEpisode(N)$
    $LongTermR.append(Rewards)$
    **if** $Average(LongTermR) > Threshold$ **then**
        $RandomLevel = RandomLevel \times \lambda$
        $Clear(LongTermR)$
    **end**
**end**

For example, consider the task of moving a point robot to reach a target point in a two-dimensional space. In each episode, the initial position of the point robot is randomly sampled in a circular area. The random level in this case is the radius of the circle. In the early training stage, the radius is set to be very small, and the initial position is almost fixed. As the policy gains more and more generality, the reward in each episode increases. Once the reward reaches a threshold, the random level increases, and the initial position of the point robot is sampled from a larger area. The terminal random level corresponds to the circumstance where the circular sampling area fully covers the working zone. If the policy performs well under the terminal random level, the policy is considered successfully trained. The pseudocode for this process is shown in Algorithm 3.

To guarantee the second requirement, an extra loss term that punishes the magnitude of the compensative action, $l_i^c \propto -\|a_i^c\|^2$, is added to the reward function so as to reduce $\|a_i^c\|$ when attribute $i$ is not active.

## 4.7 Results and Discussions

### 4.7.1 Performance of the CAN

The first set of experiments tests the capability of the CAN to learn attributes and assemble learned attributes. We first train the base attribute module using the baseline RL algorithm with CL, and then we use the cascading modules to decompose the different add-on attributes based on the pretrained base module. In the actor-critic RL training using PPO, the maximum number of episodes is 10,000. Both the actor network and the critic network are trained using the Adam optimizer, with a batch number of 256, an initial learning rate of 0.0001; they are updated 20 times in each training iteration. After training the two robot

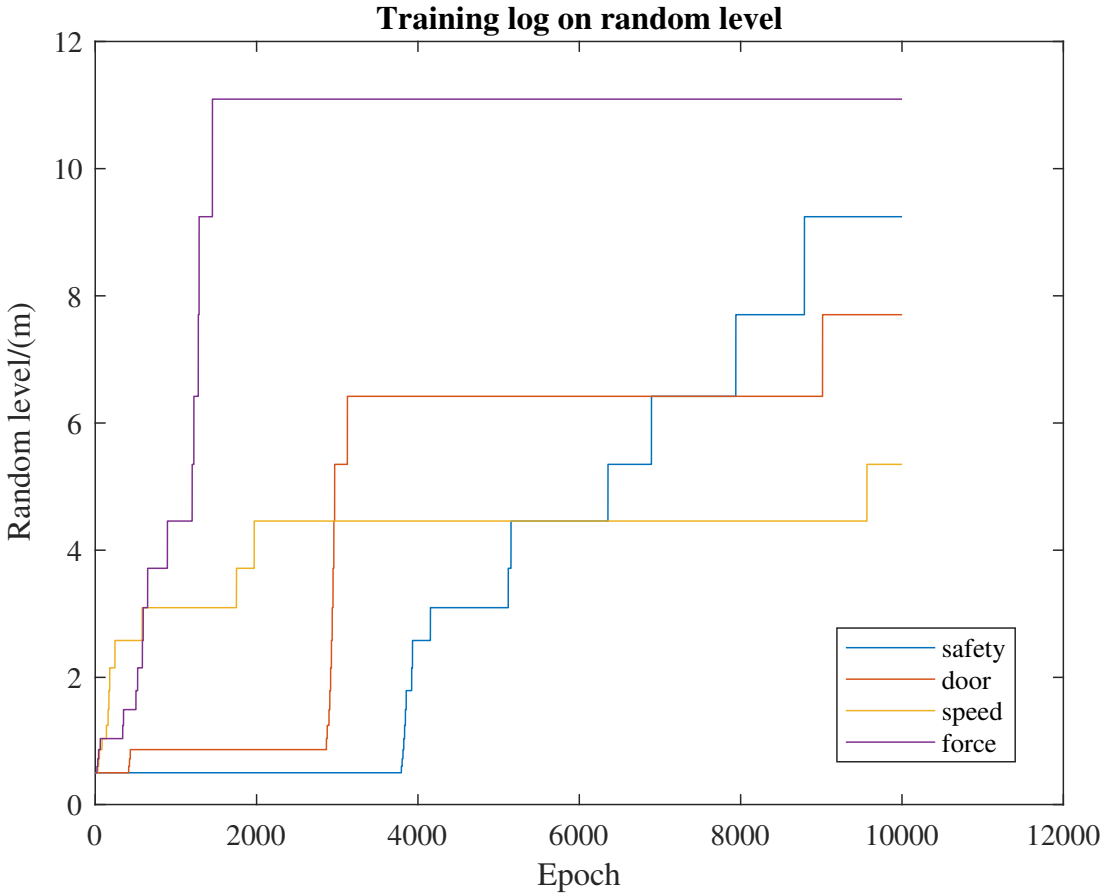


Figure 4.6: Training log on random level for the point mass robot with the four add-on attributes. Because we are using CL training techniques, the random level of the environment is increasing during the training. The higher the random level, the more general the final policy will be. If the random level could be greater than or less than 5 meters—meaning the robot could reach a target within a 5-meter range—then the task would be considered successfully solved.

scenarios and all the four add-on attributes for each scenario, we ran 10 test episodes for the eight combination tasks and received a 10/10 success rate for all the tasks. Therefore, the results show that the add-on attributes can be successfully added to the base attribute using the CAN. Fig. 4.6. shows the random-level training log for the four add-on attribute modules in the point mass scenario. If the random level could be greater than or less than 5 meters—meaning the robot could reach a target within a 5-meter range—then the task would be considered successfully solved. Fig. 4.7 and Fig. 4.8 show the example episodes of the different attribute combinations.

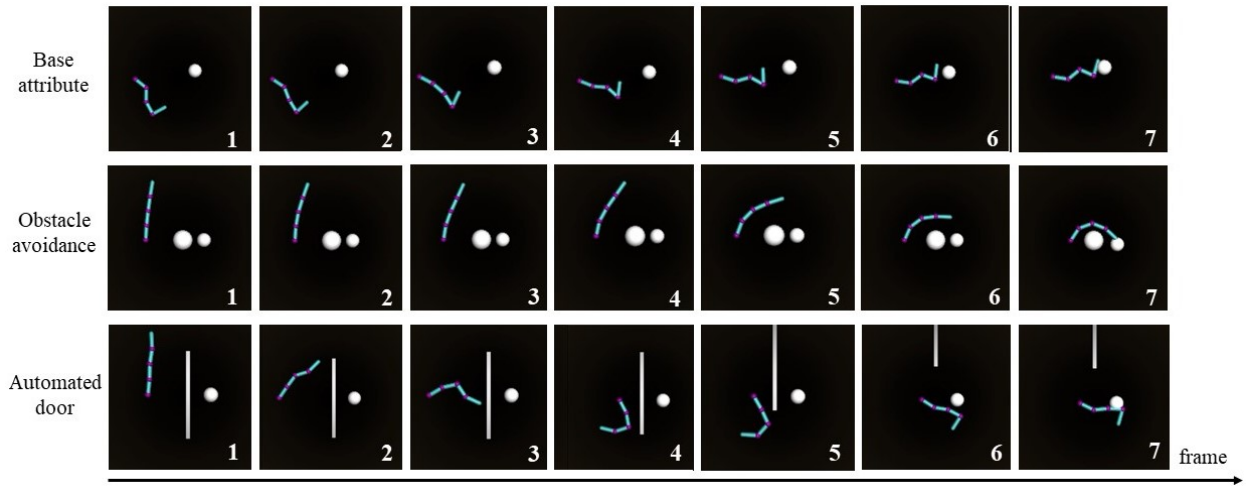


Figure 4.7: Three example episodes for the articulated robot: target reaching, target reaching while avoiding an obstacle, or an automated door.

We also show cases where the CAN assembles more than one add-on attribute module. Concretely, we first train the obstacle attribute module based on the point mass robot. Then we connect two identically parameterized obstacle attribute modules in a series following the base attribute module, each handling one obstacle ball. The CAN structure is identical as the one shown in Fig. 4.3. Fig. 4.9. shows two examples of the CAN zero shooting[1] the task where the moving point robot reaches the target while avoiding two obstacles simultaneously. Nevertheless, as the task gets more complicated and the number of attributes gets larger, it requires a certain amount of fine-tuning. In the next chapter, we will present another attribute-based decomposition approach that is better at handling more attributes at one time.

### 4.7.2 Comparison with Baseline RL Methods

We compare the capability and efficiency of the CAN and the baseline RL by analyzing their training logs. We train two policies to drive the point mass robot to reach a target location while avoiding an obstacle with only a sparse reward function. The CAN is trained with CL, with the base attribute pretrained and fixed. The resulting comparison of the reward and random level are shown in Fig. 4.10, with the maximum training iterations exceeding 120,000. For the baseline RL, the task is trained from scratch. For the baseline RL trained with CL, it is too hard for the robot to reach the target in many cases. Therefore, we also implemented RCL, which let the initial state be very close to the target in the early stage of the training phase. Using RCL, the baseline RL could gain positive rewards in the early

[1]Zero shooting means testing on a new task without extra training using the target domain data

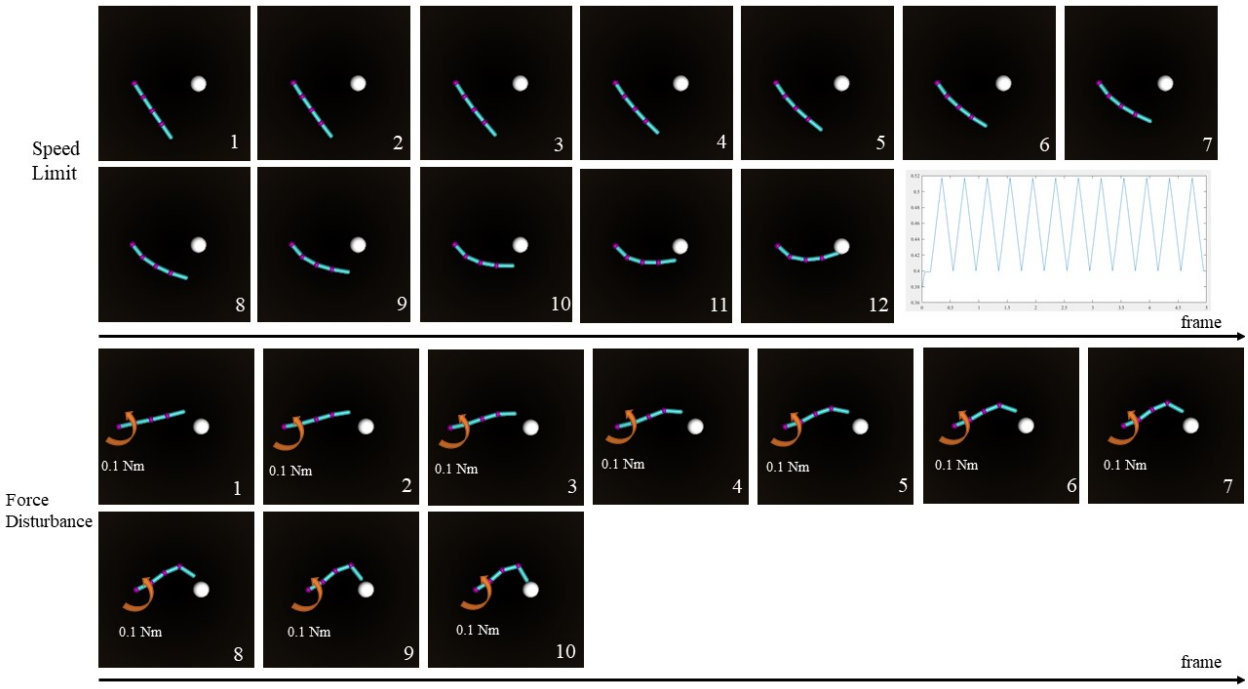


Figure 4.8: Two example episodes for the articulated robot: target reaching under the speed limit or force disturbance. Because of the limit of speed and force disturbance in the actuator, these two tasks take longer than the three previous tasks. The speed limit and external force are also visualized.

stage. The challenge is whether the RL algorithm can maintain the high reward level as the random level increases. The focus of the comparison across different training schemes is placed on the corresponding reward and random level in CL versus the training iterations.

We find that the baseline RL using CL barely learns anything. This is because the reward is too sparse; and the agent is consistently receiving punishment from the obstacle and falls into a local minimum of purely avoiding the obstacle and ignoring the target. For the baseline RL using RCL, in the early stage, the average discounted reward in an episode is high, as expected. But as the random level rises, the performance of the baseline RL with RCL drops. Therefore, the random level increases slowly as the training continues. The CAN, however, is able to overcome the misleading punishments from the obstacle, thanks to the guidance of the instructive base attribute policy. As a result, the random level of the CAN rises rapidly, and the CAN achieves the terminal random level more than 10 times faster than the baseline. These results indicate that the attribute module learns substantial knowledge of the attribute as the CL-based training goes on.

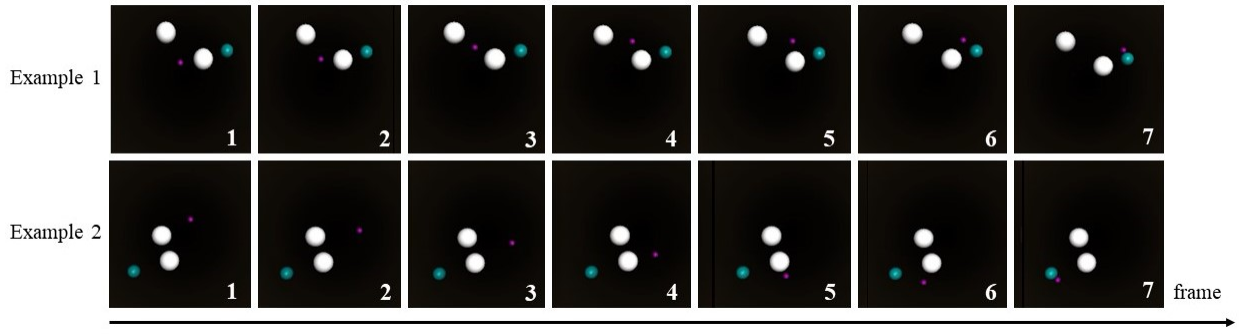


Figure 4.9: The moving point robot (the pink ball) reaches the target (the green ball) while avoiding two obstacles simultaneously. Note that this task is never seen by the CAN. The obstacle attribute module is trained only once, and two identically parameterized obstacle attribute modules corresponding to two different obstacles are assembled together to zero shoot the new task.

## 4.8 Chapter Summary

In this chapter, we introduce the attribute learning concept and present the advantages of using this novel method to decompose the control policies of complicated control tasks. The RL framework we propose, the CAN, uses the cascade attribute module structure to model the characteristics of the attributes. The attribute modules are trained with the guidance of the pretrained base attribute module. We validate the effectiveness of the CAN in decomposing and assembling attributes and show the advantages of the CAN in solving complicated tasks compared to the baseline RL. In the next chapter, we present an extension of the attribute learning work by organizing the attribute networks in a parallel manner for better performance on multi-attribute tasks.

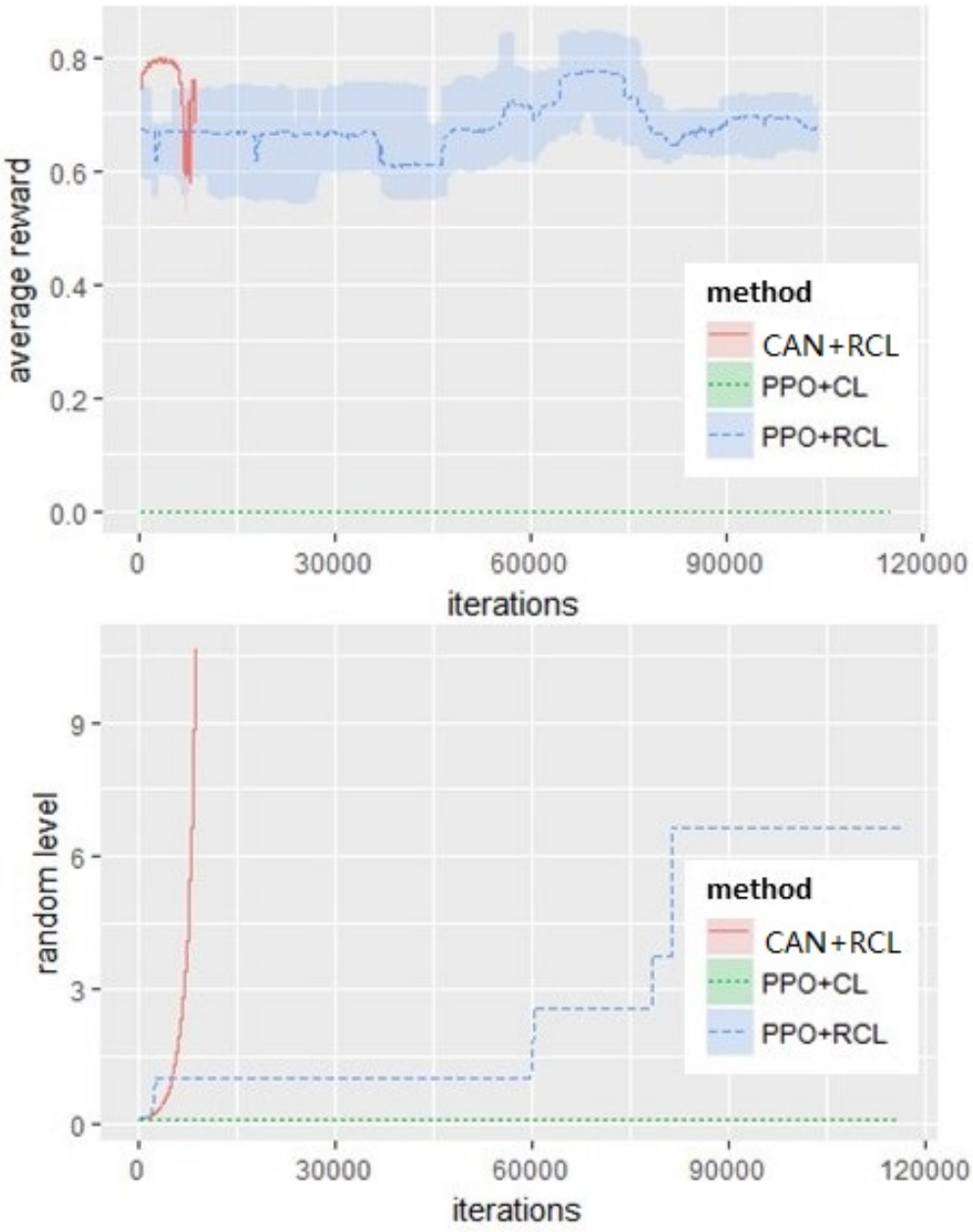


Figure 4.10: Comparison between the performance of the CAN and the baseline RL (PPO) in the training phase.

# Chapter 5

# Parallel Attribute Networks

In the last chapter, we argued that network policies are generally hard to train and that the learned knowledge encoded in neural network policies are difficult to transfer. We proposed the cascade attribute network (CAN), which decomposes complicated policies in a cascade manner. However, the CAN is not robust for problems with multiple attributes. In this chapter, we present the parallel attribute network (PAN), which decomposes a policy in a parallel manner [122]. We study this method in an autonomous driving scenario and modularize the complicated driving policies in terms of the driving attributes to fulfill the requirements of the attributes in the driving tasks separately. Concretely, we first train a policy network to accomplish the base lane tracking attribute. The modules for the add-on attributes, such as avoiding obstacles and obeying traffic rules, are then trained to map the corresponding state to a satisfactory set of the vehicle action space. Finally the reference action given by the base policy is projected into the satisfactory sets so as to satisfy the requirements of all the attributes. Using the PAN, many complicated tasks that are hard to learn from scratch can be easily learned; also, unseen driving tasks can be solved in a zero-shot manner by assembling the pretrained attribute modules. We have validated the capability of our model on a class of autonomous driving problems with attributes of obstacle avoidance, traffic light, and speed limit in simulation. Experimental results based on an obstacle avoidance task are also presented.

## 5.1 Introduction

Researchers have explored neural network policies (NNPs) to drive autonomous vehicles in complicated environments. Generally, NNPs can take in various observational inputs, such as raw sensor data or estimated states, and output the driving commands for autonomous vehicles. In real driving applications, for the sake of reliability and robustness, NNPs often map low-dimensional state space input to the vehicle control commands, leaving the perception and state estimation to other specific modules. NNPs are trained mainly using two kinds of approaches: imitation learning (IL) and reinforcement learning (RL). The IL

approach refers to first building a dataset of expert driving behavior and then training a neural network to clone the behavior of the dataset [17][114]. But the RL approach starts from a random NNP initiation and then optimizes the NNP through its interaction with the environment and the associated reward or cost [103].

Compared with traditional rule-based or optimization-based control policies, NNPs have various advantages: (1) NNPs can represent sufficiently complicated and subtle models that are hard to model analytically; (2) the IL and RL methods involve vast datasets or environment exploration, making the NNPs capable of handling interactive scenarios and rare cases [114]; (3) the NNPs are much faster to run on autonomous vehicles compared with solving online optimization problems.

However, there are a number of challenges that slow down the application of NNPs in autonomous driving. One of the key challenges is the difficulty of transferring knowledge encoded in one NNP to another. This difficulty further impedes NNPs from being adept in solving complicated tasks. Specifically, due to the agnostics of the internal composition and the fixed input-output structure of the neural networks, the knowledge encoded inside a pretrained NNP is hardly reusable for the NNP of another driving task. Given that the training process of the NNP is difficult for either IL or RL, solving the pain spot of transferring the knowledge encoded in the NNPs is essential for enabling the NNPs to address complicated autonomous driving tasks.

Existing neural network knowledge transfer methods can be categorized into three groups: transfer learning [86, 80], meta learning [37, 40], and neural network architecture modification [92]. However, these methods either encode robustness into NNPs and hope for the best in the new tasks or rely on the fine-tuning of the NNPs in the new tasks. Moreover, these methods still lack the interpretable intermediate layers, and the fundamental problem of the agnostics of the NNPs is not solved. Therefore, they are not suitable for autonomous driving scenarios where safety and reliability are of utmost importance.

We propose to address this problem from a new perspective: Modularize NNPs in terms of a series of driving attributes for complicated driving problems. The attributes are defined to be global characteristics or requirements that take effect throughout a driving task. For example, tracking the closest lane, avoiding hitting the wall and pedestrians, and obeying the traffic rules can all be considered as attributes. In our framework, to obtain an NNP for a complicated driving task, one must first train or obtain the pretrained attribute modules for the attributes in the driving task, then assemble the attribute modules together to produce a hierarchical NNP. We also propose a parallel attribute network (PAN) architecture, in which the attribute networks do not tangle with each other, but instead the attribute modules take in some decoupled fractions of the full state and are assembled in a parallel manner to apply influence toward the action command for the autonomous vehicle. The PAN has the following advantages:

1. Decomposing a higher dimensional complicated task into lower dimensional attributes makes the training process easier and faster.

2. The pretrained attribute modules can be assembled into other NNPs, making it possible to build up different policies to adapt to various driving tasks.

3. Since the attribute modules take in only decoupled fractions of the full state, the PAN can dynamically manage the input dimension of the NNP.

4. The PAN architecture can handle more attributes simultaneously thanks to the decoupling formulation of the parallel attribute modules.

The reminder of this chapter is as follows: We formally define the problem setup in Section 5.2, where we also discuss the challenge of knowledge transfer in NNPs. In Section 5.3, we introduce the concept of attribute modularization, and we present the architecture of the PAN and the implementation details. In Section 5.4 and Section 5.5, we present the simulation and experiment results, respectively. Lastly, the chapter summary is given in Section 5.6.

## 5.2 Problem Statement

### 5.2.1 Driving Tasks Consisting of Multiple Attributes

We consider a class of autonomous driving tasks consisting of multiple key attributes (global characteristics or requirements that take effect throughout the task), including lane tracking, obstacle avoidance, traffic light, and speed limit, as shown in Fig. 5.1. By assigning the number and status of the attributes, many autonomous driving tasks can be included in this class of tasks. The vehicles are all modeled using the kinematic model as shown in Fig. 5.2 - left. For the $i^{th}$ vehicle, the state is: $s_i = [p_i^x, p_i^y, v_i, \theta_i]^T$, standing for the $(x, y)$ position, the speed, and the yaw angle of the vehicle. The control commands for the vehicle are the longitudinal acceleration $a_i$ and the steering angle $\gamma_i$. Since the mapping of the steering angle $\gamma_i$ and the yaw rate $\dot{\theta}_i$ are homeomorphic, we choose the action to be $u_i = [a_i, \dot{\theta}_i]^T$. The discretized vehicle dynamic function is:

$$s_i(k+1) = s_i(k) + \begin{bmatrix} v_i cos(\theta_i) \\ v_i sin(\theta_i) \\ u_i \end{bmatrix} \cdot dt \tag{5.1}$$

where $k$ is the time index and $dt$ is the sample time. We denote the autonomous vehicle to be the $0^{th}$ vehicle, the other vehicles (obstacle vehicles) to be the $1^{st}$ vehicle, the $2^{rd}$ vehicle, and so on.

Since we consider structured driving, we consider the performance of the autonomous vehicle tracking the closest lane to it, which corresponds to the lane tracking attribute. We describe a vehicle's lane tracking performance using the lateral deviation $\Delta y_0$ and the yaw error $\Delta\theta_0$, as shown in Fig. 5.2 - Right. The resulting lane tracking state is called $s_{lt} = [\Delta y_0, \Delta\theta_0]^T$.

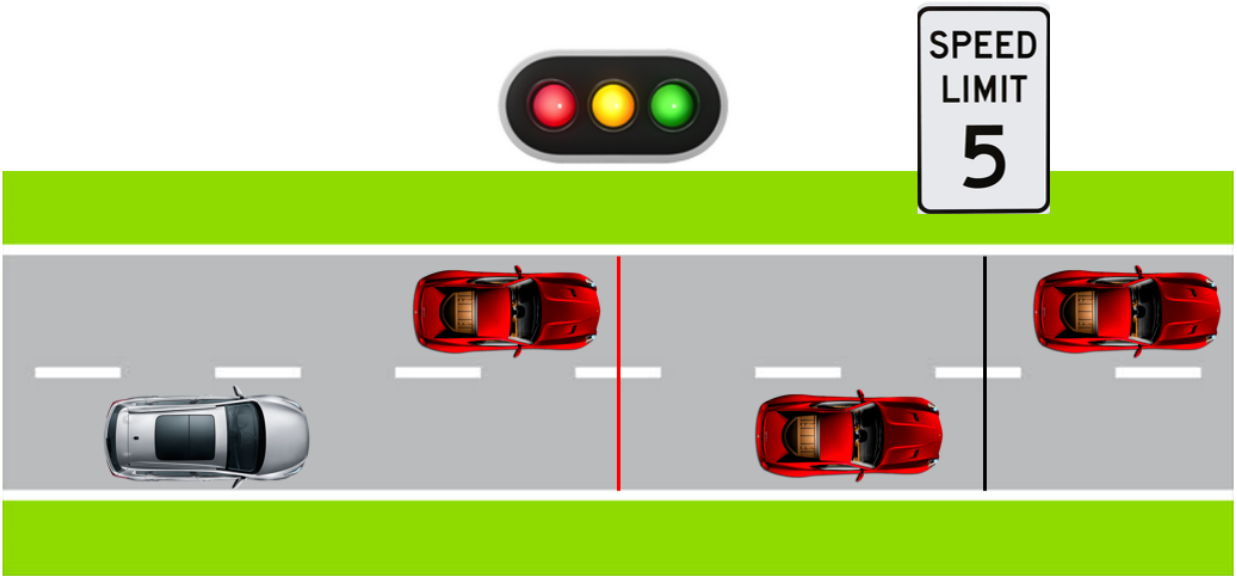


Figure 5.1: A typical autonomous driving task that includes lane tracking, obstacles, traffic light, and speed limit attributes. The red line shows the position and status (red) of the traffic light, and the black line shows the position of the speed limit sign.

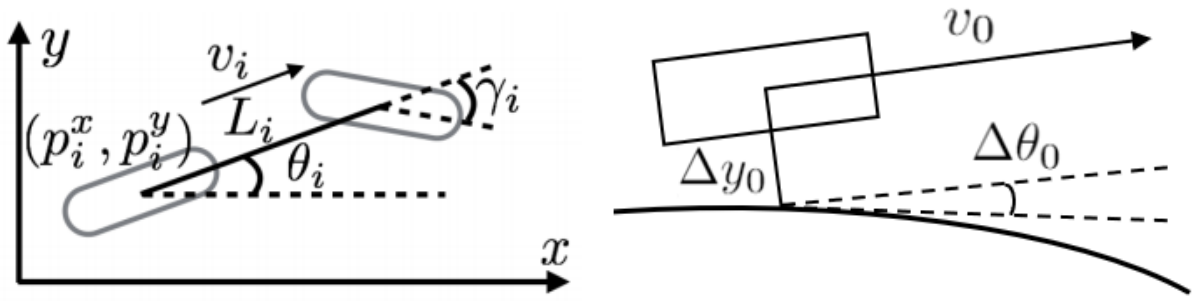


Figure 5.2: Left: The kinematic model for vehicle $i$. Right: The lateral deviation and the yaw angle error of the autonomous vehicle with respect to the lane being tracked.

The traffic rule attributes include the traffic light attribute and the speed limit attribute. For a traffic light, the state vector describing its status is $s_{tl} = [B_{tl}, x_{tl}]^T$, where $B_{tl}$ is the bool variable indicating whether the light is red or green and $x_{tl}$ is the position of the traffic light. For a speed limit, the state vector is $s_{sl} = [x_{sl}, v_{sl}]^T$, where $x_{sl}$ indicates the starting position of the speed limit area and $v_{sl}$ is the speed limit value.

With all the attributes defined, one can define a class of autonomous driving tasks (for

those attributes that are not included, the methodology is similar), such as pure lane tracking, lane tracking while avoiding a certain number of obstacles, and obeying the traffic rules. We denote a complicated task consisting of several attributes using the sum of all the attributes. For a driving task that involves lane tracking ($LT$), while avoiding $n$ obstacles ($OB_i$, including road edges, denoted $OB_{re,i}$), and satisfying $m$ traffic light ($TL$) and $l$ speed limit ($SL$) constraints, we can denote it using the following form:

$$LT \oplus \sum_{i=1}^{n} OB_i \oplus \sum_{j=1}^{m} TL_j \oplus \sum_{k=1}^{l} SL_k \tag{5.2}$$

where $\oplus$ denotes the sum of different attributes and the $\Sigma$'s are in terms of $\oplus$. Our goal is to obtain a series of NNPs that can drive the autonomous vehicle to complete the tasks defined in (5.2).

### 5.2.2 Challenges in NNP Knowledge Transfer

Training NNPs is difficult even when using IL or RL, especially for those complicated tasks with many attribute requirements to satisfy; this is because these NNPs are large in size and require a great amount of computation to train. However, the difficulty of transferring knowledge encoded in the NNPs makes it hard to reuse the NNPs in other similar tasks. This difficulty comes mainly from the agnostics of the internal composition and the fixed input-output architecture of the neural network.

Concretely, the NNP for the lane tracking task ($LT$) has an input size of $dim(s_0)+dim(s_{lt})$ and an output size of $dim(u_0)$, where the $0^{th}$ vehicle is the autonomous vehicle and $dim(\cdot)$ stands for the dimension of a vector. After training the NNP, the task-related knowledge is encoded inside the NNP. Consider a very similar task $LT \oplus OB_1$, which is lane tracking while avoiding the $1^{st}$ obstacle vehicle. Its corresponding NNP output size is still $dim(u_0)$, but its input size is $dim(s_0) + dim(s_{lt}) + dim(s_1)$ instead. There are many similarities and common knowledge that are worth transferring between $LT$ and $LT + OB_1$, but the agnostics of the internal meaning of neural networks prevent any kind of knowledge transfer. The consequence is the NNP for $LT$ is not useful for the new task, and a new NNP has to be trained from scratch for $LT \oplus OB_1$. If we further add a speed limit attribute, the NNP for the $LT \oplus OB_1 \oplus SL$ task has an input size of $dim(s_0) + dim(s_{lt}) + dim(s_1) + dim(s_{sl})$, and the learned knowledge cannot be transferred to the new task either.

In summary, for the infinite number of different driving tasks involving the $LT$, $OB$, $TL$, and $SL$ attributes, no knowledge can transfer among their NNPs. Given that the training of an NNP from scratch is generally difficult, designing a framework that can transfer and reuse the previously learned knowledge encoded inside an NNP would be of great use.

## 5.3 Policy Modularization Methodology

### 5.3.1 Attribute Modularization

We propose to modularize the NNPs for autonomous driving tasks in terms of the attributes involved, with each involved attribute decoupled and handled by a corresponding attribute module. For a driving task described in the last Section, the $LT$, $OB$, $TL$, $SL$ attributes are handled by their corresponding attribute modules. The form of an attribute module is using a neural network to map the attribute's related state to its influence on the NNP. Therefore, an attribute module is also called an attribute network. Attribute modularization involves investigating the two following issues:

1. How to decouple the attribute modules (or how to define the input of an attribute network).

2. How to assemble the attribute modules (or how to define the output of an attribute network and how to get the overall NNP output from the outcome of all the attribute networks).

The first question is easier to answer: The input state to an attribute network is defined to be the minimum state space that fully characterizes the property of the attribute. For example, for the $LT$ module, the input state is $[s_0, s_{lt}]^T$, the input state for the $OB_i$ module is $[s_0, s_i]^T$, the states for the $TL$ and the $SL$ modules are $[s_0, s_{tl}]^T$ and $[s_0, s_{sl}]^T$, respectively[1]. Under this definition, the input information is sufficient for an attribute network to characterize the attribute, while the unrelated information is excluded, avoiding noise information to pollute the training of the attribute network. In Section 5.4, the simulation results also validate the effectiveness of the input state definition.

The challenging part is designing the attribute network outcome and calculating the overall NNP output from modular outcomes. Ideally, the outcome of an attribute module shall depend solely on this attribute, and the influence from different attribute modules shall be easy to combine. In this way, the training of the attribute modules does not rely on one another, and the learned knowledge preserved in an attribute module can be assembled to other NNPs. In the last chapter (papers [116], [21]), the cascade attribute networks (CAN) can handle only a small number of attributes. However, since the attribute modules in CAN are connected in cascade, the latter attribute modules can pollute the influence from the previous modules and deteriorate the overall policy performance. The PAN architecture proposed in this chapter can overcome this problem by letting the attribute networks work strictly in parallel. The frameworks of the CAN and the PAN are compared in Fig. 5.3 and Fig. 5.4.

[1]More precisely, the input states to the attribute networks are derived from the state vectors described above

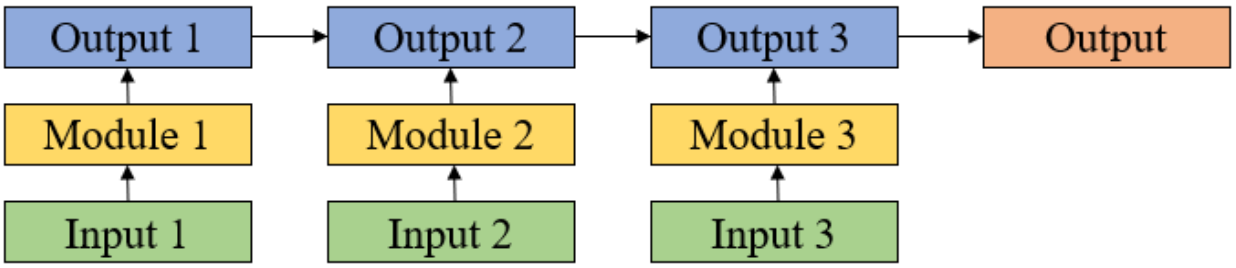


Figure 5.3: The high-level structures of the cascade attribute networks (CAN).

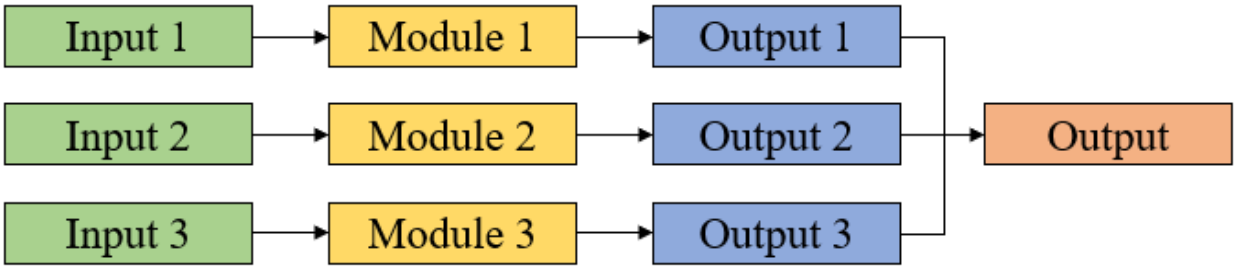


Figure 5.4: The high-level structures of the parallel attribute networks (PAN).

### 5.3.2 The Parallel Attribute Networks (PAN)

In this subsection, we present the hierarchical structure of the PAN, whose architecture is shown in Fig. 5.5. To present the detailed architecture of the PAN, we first divide all the attributes in a driving task into two groups: the base attribute of $LT$ and the other add-on attributes. The set of add-on attributes in the driving task is called $\Phi$, which consists of a series of $OB$, $TL$, and $SL$ attributes.

In the PAN, the base $LT$ attribute network is trained to be an NNP that maps the input state $[s_0, s_{lt}]^T$ to a reference action output $u_0^0$, such that the autonomous vehicle can track the closest lane following the policy. The superscript 0 means this output action is a reference action of the autonomous vehicle. For an add-on attribute $\phi \in \Phi$, the output influence is defined to be a satisfactory set $\Omega_\phi$ in the vehicle action space. The satisfactory set is defined as in (5.3) to be the set of actions that, if taken by the autonomous vehicle, satisfies the corresponding attribute $\phi$. More details on the satisfactory set are included in Section 5.3.3.

$$\Omega_\phi = \{u_0 \mid \phi \text{ is satisfied}\} \tag{5.3}$$

Since the autonomous vehicle action shall be such that all the attributes are satisfied, the PAN performs a projection operation of the reference action $u_0^0$ into the intersection of all the satisfactory sets. The projection is essentially solving a constrained optimization problem

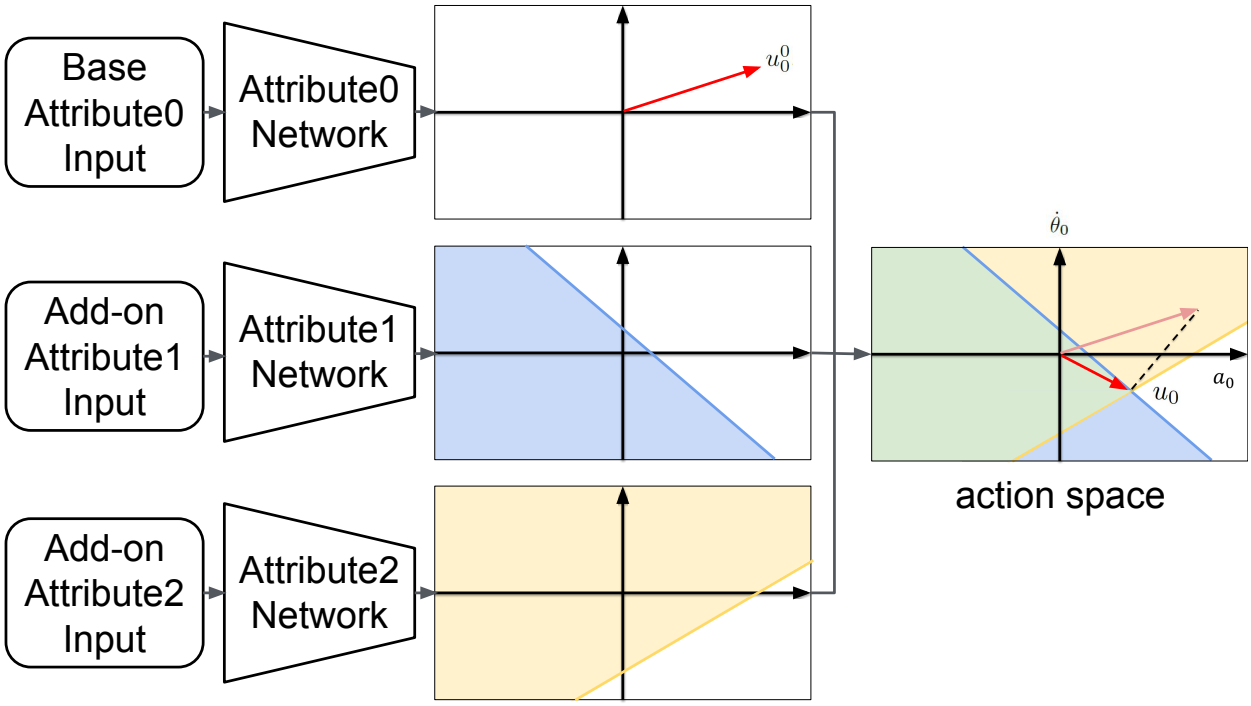


Figure 5.5: The architecture of the parallel attribute networks (PAN). The output of the base attribute network is a reference action in the autonomous vehicle action space (the red vector) and the output of the add-on attribute networks are the satisfactory sets in the action space. The overall NNP output is the projection of the reference action into the intersection of all the satisfactory sets (the green vector).

according to some distance function $dist(\cdot)$:

$$\min_{u_0} \ dist(u_0, u_0^0) \tag{5.4}$$

$$\text{subject to } u_0 \in \Omega_\phi, \forall \phi \in \Phi \tag{5.5}$$

As shown in (5.4), (5.5), and Fig. 5.5, the outcome of the base attribute module, $u_0^0$, and the outcome of the add-on attribute modules, $\phi$'s, work strictly in parallel to produce the overall policy output $u_0$. Therefore, the independence and transferability of the attribute modules is guaranteed.

### 5.3.3 The Satisfactory Set

A satisfactory set $\Omega_\phi$ is defined to be the set of vehicle actions that, if taken, can make the requirements of attribute $\phi$ be satisfied (avoid collisions, obey traffic rules, and so on).

However, quantitatively defining the satisfactory terms remains an open question. Therefore, we define approximate satisfactory sets according to the training approaches for the attribute modules. For example, if an attribute is trained using IL with human-labelled satisfactory sets, a convenient form of the satisfactory set would be:

$$\Omega_\phi = \left\{ u_0 |\ a_{min} < a_0 < a_{max},\ \dot{\theta}_{min} < \dot{\theta}_0 < \dot{\theta}_{max} \right\} \tag{5.6}$$

If the IL expert satisfactory sets are analytically calculated, their form shall comply with the corresponding theory. If the attribute module is trained using RL, then more freedom is available for the form of the satisfactory set.

In this work, the half-space approximate satisfactory set is applied. Under such a setting, the output of the $\phi$ attribute network is defined to be a three-dimensional vector $[a_\phi, b_\phi, c_\phi]^T$, that defines a half space of:

$$\Omega_\phi = \left\{ u_0 |\ a_\phi \cdot a_0 + b_\phi \cdot \dot{\theta}_0 \leq c_\phi \right\} \tag{5.7}$$

We further define the distance function $dist(\cdot)$ in (4) to be a quadratic loss function. Therefore, the constrained optimization problem defined in (4) and (5) is then a quadratic program:

$$\min_{u_0}\ \frac{1}{2}(u_0 - u_0^0)^T \Lambda (u_0 - u_0^0) \tag{5.8}$$

$$\text{subject to } [a_\phi, b_\phi] \cdot u_0 \leq c_\phi, \forall \phi \in \Phi \tag{5.9}$$

where the positive definite matrix $\Lambda$ defines a metric in the action space. The quadratic program is much faster to solve compared with general constrained optimization, making the forward stream of the PAN fast enough for onboard implementation.

## 5.4 Simulation Results

### 5.4.1 Simulation Setup

We conduct the simulation studies based on a class of autonomous driving tasks defined in Section 5.2.1, involving attributes of $OB_i$ ($OB_{re,i}$), $TL$, and $SL$. The autonomous vehicle is assumed to start in the rightmost lane with a desired longitudinal speed $v_{tar} = 10m/s \approx 22mph$. The sampling time is $0.02s$. In Section 5.4.1 and 5.4.2, the base attribute module is trained using RL, and the add-on attribute modules are trained using IL, with analytical models serving as expert demonstrations. In Section 5.4.3, we discuss other training methods for the attribute modules.

The base attribute network is trained using RL in a lane-tracking case $LT$. The reward function of the Markov Decision Process (MDP) for RL is a quadratic function of the lateral deviation and the speed error:

$$r(t) = -2 \cdot \Delta y_0^2 - (v_0 - v_{tar})^2 \tag{5.10}$$

The state-of-the-art policy gradient algorithm, proximal policy optimization (PPO) [97], is applied to optimize the base attribute network.

For IL of an add-on attribute network, the training data is collected from the cases with only the base $LT$ attribute and the attribute to be trained, so as to guarantee its independence from the other add-on attributes. For the $OB$ ($OB_{re,i}$) module, the collision avoidance constraint is approximated using the safety set algorithm (SSA) [67] [68]; for the $TL$ and the $SL$ modules, the speed constraints are derived using the intelligent driver model (IDM) [54]. We then reform the constraints into the form of half-plane satisfactory sets and use them as the IL training data. After the IL converges, the trained attribute modules are fixed and can be assembled into PAN policies as needed. That is, the performance of the PAN policies in the cases described in Section 5.4.2 are all in a zero-shot manner. The projection of the reference action $u_0^0$ into the satisfactory sets $\Omega_\phi$'s is a quadratic program problem, and it is solved using the CVX solver.

### 5.4.2 Simulated Cases

We test the performance of the PAN in a variety of driving scenarios by simulation. The driving scenarios are categorized into two groups: cases with only one kind of add-on attribute (not necessarily only one attribute) and those that involve multiple kinds of add-on attributes.

#### Cases with Only One Kind of Attributes ($LT \oplus \Sigma_i OB_i$, $LT \oplus TL$, or $LT \oplus SL$)

These are cases where we collected the IL training data. Therefore, the success of PAN policies in these tasks illustrates that attribute modules can be trained to handle their corresponding attributes.

**Obstacle Avoidance ($LT \oplus OB_1 \oplus OB_2 \oplus OB_{re,1} \oplus OB_{re,2}$)** Fig. 5.6 and Fig. 5.7 show the case when the autonomous vehicle tracks the closest lane while avoiding two obstacles vehicles and two road edges. The obstacle vehicles are driving at a constant speed of $5m/s \approx 11mph$. In all, the PAN policy contains four pretrained $OB$ networks and one base $TL$ attribute network. Fig. 5.6 shows the behavior of the autonomous vehicle under the PAN policy, as it makes two lane changes in order to surpass the slow obstacle vehicles. In Fig. 5.7, we show the interpretable meanings of the influences of the $OB$ attribute modules. The first column represents when an autonomous vehicle is far from obstacle vehicles, only the satisfactory set of the right road edge $OB$ module limits the steering of the autonomous vehicle, preventing

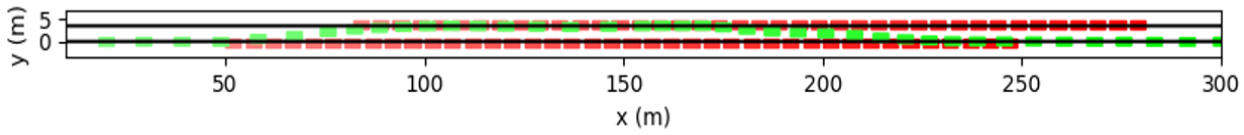


Figure 5.6: The behavior of the autonomous vehicle (the green vehicle) in task $LT \oplus OB_1 \oplus OB_2 \oplus OB_{re,1} \oplus OB_{re,2}$. Lighter colors indicates earlier in time.

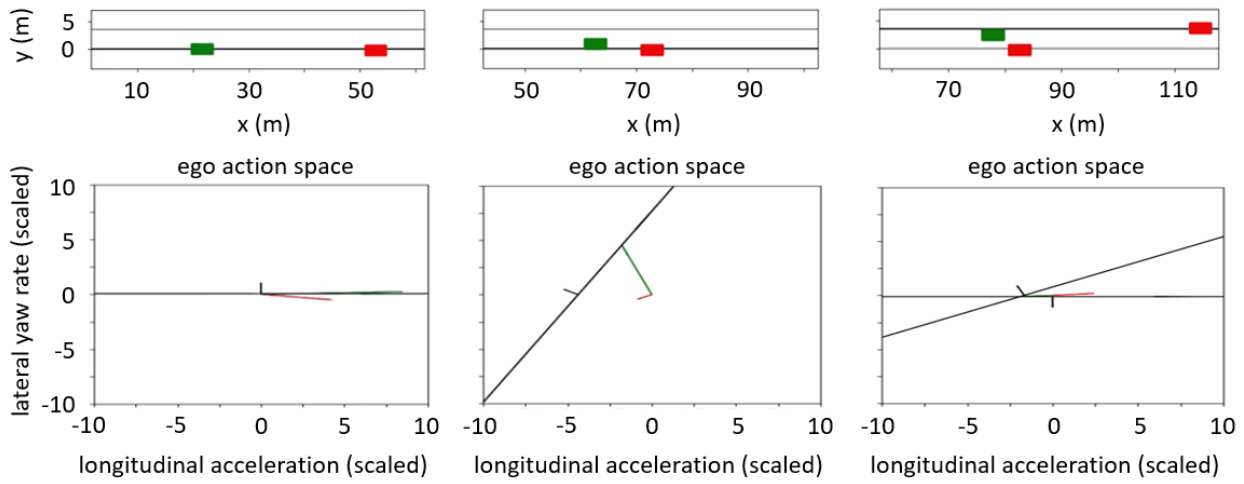


Figure 5.7: (Three typical frames in the task: the visualization of the reference action $u_0^0$ (red vector), the half-plane satisfactory sets $\Omega_\phi$ (black line boundary with normal vector indexing the inner side), and the real action $u_0$ (green vector).

it from colliding with the right road edge. In the second column, when the autonomous vehicle approaches the first obstacle vehicle, the major influence comes from this obstacle vehicle, as its satisfactory set forces the autonomous vehicle to decelerate and sheer off to avoid collision. In the last column, when the autonomous vehicle changes to track the left lane, in addition to the influence from the obstacle vehicle, the left road edge also limits the steering of the autonomous vehicle to prevent collision.

**Traffic Light Handling ($LT \oplus TL$)** Fig. 5.8 shows the case when the autonomous vehicle is driving under the traffic light rule. The traffic light, placed 80 meters ahead of the autonomous vehicle, is initially red and turns green at time $t = 12sec$. The first two figures show the behavior of the autonomous vehicle when traffic is red or green, and the last figure shows the speed profile of the autonomous vehicle. In the PAN policy, the $TL$ attribute network generates a satisfactory set that limits the maximum acceleration of the autonomous vehicle and forces it to decelerate to a full stop in front of the red light. After the traffic

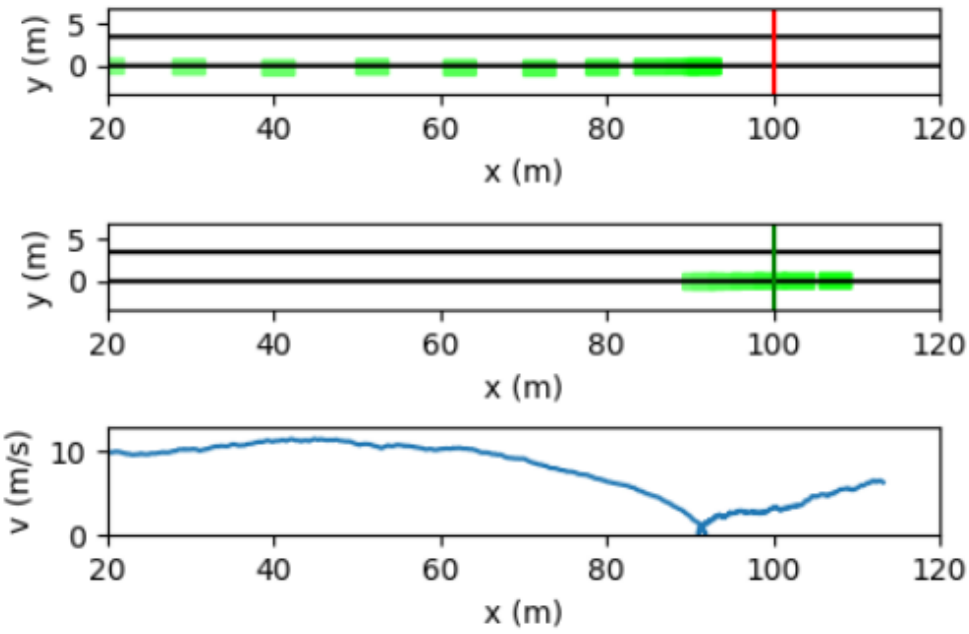


Figure 5.8: Behavior and speed profile of the autonomous vehicle in $LT \oplus TL$ task. The first two figures indicate when the traffic light is red and after it turns green.

light turns green, the constraint from the $TL$ module is removed. The autonomous vehicle then accelerates and passes the traffic light.

**Speed Limit Handling ($LT \oplus SL$)** Fig. 5.9 shows a case when the autonomous vehicle is driving while obeying the speed limit rule. The speed limit area starts at 40 meters ahead of the autonomous vehicle. The speed limit value is $v_{sl} = 5m/s \approx 11mph$. In Fig. 5.9, the behavior and speed profile of the autonomous vehicle is shown. In the PAN policy, the $SL$ attribute network generates a satisfactory set that limits the maximum acceleration of the autonomous vehicle and forces it to decelerate to $v_sl$ inside the speed limit area.

### Cases with Multiple Kinds of Attributes

Such cases, however, are unseen in the training phase. They involve more than one kind of add-on attribute out of $OB$, $TL$, and $SL$. These cases are more complicated compared to the cases in the previous group. To build up the NNPs for these tasks, the involved pretrained attribute modules are assembled together without fine-tuning. The success of these tasks demonstrates the capability to transfer the knowledge encoded inside the attribute networks and to zero-shoot new tasks.

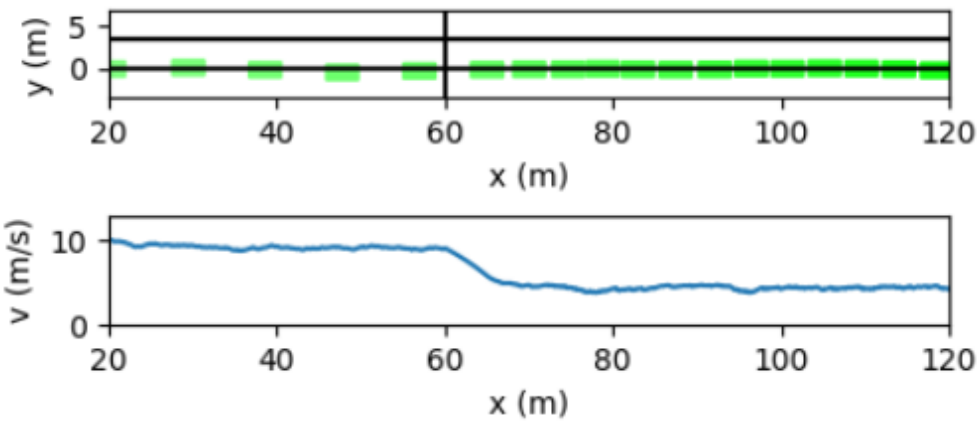


Figure 5.9: Behavior and speed profile of the autonomous vehicle in $LT \oplus SL$ task. The vertical black line indicates the position of the speed limit sign.

**Avoiding Obstacles while Obeying Traffic Light** ($LT \oplus OB_1 \oplus OB_2 \oplus OB_{re,1} \oplus OB_{re,2} \oplus TL$) Fig. 5.10 shows a case when the autonomous vehicle is in an environment including two obstacle vehicles, two road edge obstacles, and a traffic light. The obstacle vehicles start at a speed of $5m/s \approx 11mph$ and are then controlled by IDM. The traffic light is green for the first four seconds, then turns red and maintains red until time $t = 12sec$. The first three figures in Fig. 5.10 show the behavior of the autonomous vehicle under the PAN policy, including one $LT$ module, four $OB$ modules, and a $TL$ module. The autonomous vehicle surpasses the first obstacle vehicle, then as the traffic light is red, it stops at the stop line with the obstacle vehicle. The foremost obstacle vehicle passes the traffic light before $t = 4sec$, when the light is still green. In the end, as the traffic light turns green again, the autonomous vehicle starts to accelerate. Fig. 5.10 also shows the speed profile of the autonomous vehicle in the last figure.

**Avoiding Obstacles while Obeying the Speed Limit** ($LT \oplus OB_1 \oplus OB_2 \oplus OB_{re1} \oplus OB_{re2} \oplus SL$) Fig. 5.11 shows a case when the autonomous vehicle handles obstacles and obeys the speed limit. The speed limit sign is 40 meters ahead of the autonomous vehicle. The obstacle vehicles start at a speed of $5m/s \approx 11mph$ and are then controlled by IDM. As in the previous case, there are five attribute modules working in parallel in the PAN policy. In the first figure of Fig. 5.11, the autonomous vehicle first approaches the closest obstacle vehicle and can change lanes and surpass the obstacle vehicle. However, since the autonomous vehicle enters the speed limit area, the $SL$ attribute forces it to decelerate to $5m/s$, and it no longer tries to surpass the obstacle vehicle.

**Avoiding Obstacles while Obeying Traffic Light and Speed Limit** ($LT \oplus OB_1 \oplus OB_2 \oplus OB_{re1} \oplus OB_{re2} \oplus TL \oplus SL$) Fig. 5.12 shows a case when the autonomous vehicle

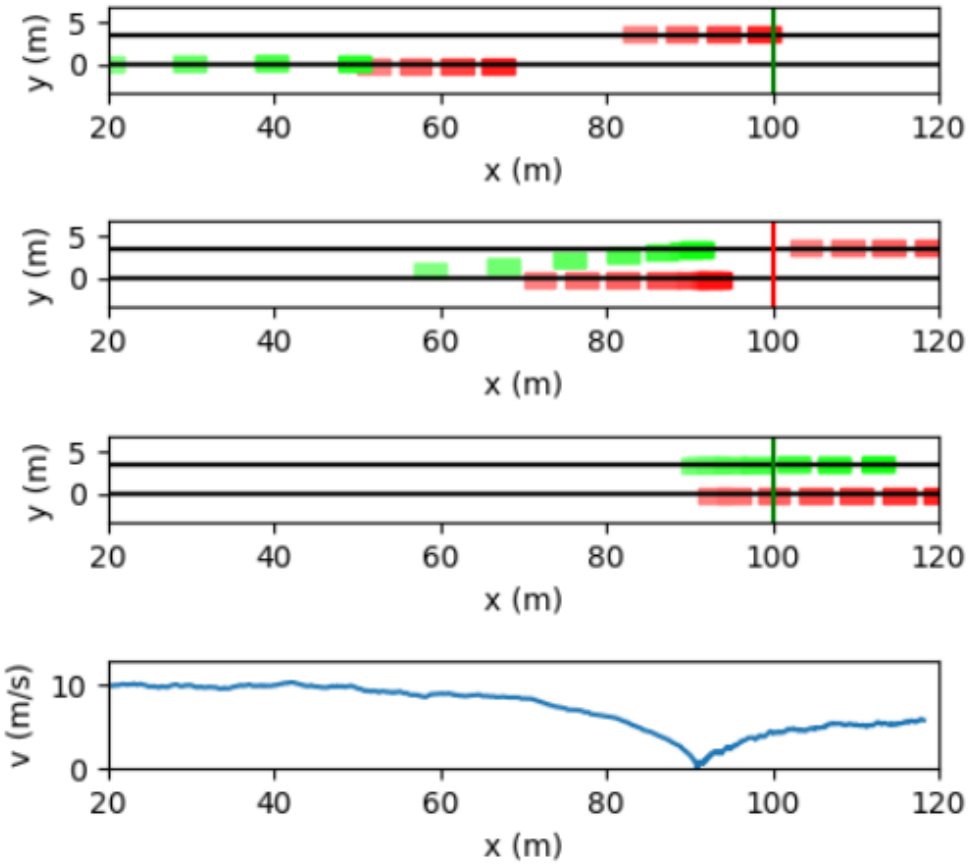


Figure 5.10: Behavior and speed profile of the autonomous vehicle in $LT \oplus OB_1 \oplus OB_2 \oplus OB_{re,1} \oplus OB_{re,2} \oplus TL$ task. The vertical green and red lines indicate the position and status of the traffic light.

handles the obstacles, the speed limit, and the traffic light rules. There are six pretrained attribute modules assembled inside the PAN policy, each handling one of the attributes in the driving task. The resulting PAN policy can handle all the attributes in a zero-shot manner. The autonomous vehicle in this case enters the speed limit area following the obstacle vehicle. Different from the last case, since the obstacle vehicle decelerates to stop in front of the traffic light, the autonomous vehicle makes a lane change and stops at the traffic light. Then after the traffic light turns green, the autonomous vehicle and the obstacle vehicle both accelerate to pass the traffic light. The foremost obstacle vehicle passes the traffic light when it is green. The last figure in Fig. 5.12 shows the speed profile of the autonomous vehicle.

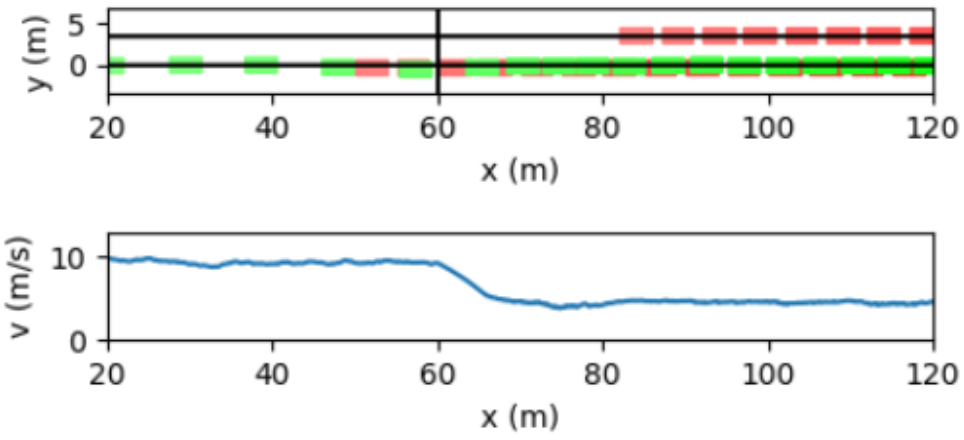


Figure 5.11: Behavior and speed profile of the autonomous vehicle in $LT \oplus OB_1 \oplus OB_2 \oplus OB_{re1} \oplus OB_{re2} \oplus SL$ task. The vertical black line indicates the position of the speed limit sign.

### 5.4.3 Training Approaches for Attribute Modules

In this subsection, we compare the performance of RL and IL for training the $SL$ attribute module, and we discuss the selection of the training approaches for the attribute modules. The $SL$ attribute module can be easily trained using RL because one can conveniently define a proper reward function for the $LT \oplus SL$ task. Making minor modifications from (5.10), we can design the reward function to be:

$$r(t) = \begin{cases} -2 \cdot \Delta y_0^2 - (v_0 - v_{tar})^2, & \text{out of speed limit} \\ -2 \cdot \Delta y_0^2 - max(v_0 - v_{sl}, 0)^2, & \text{inside speed limit} \end{cases} \tag{5.11}$$

Based on the reward function (5.11), we fix the base LT module and train the add-on $SL$ attribute network using RL and IL; we then record and compare their performance (shown in Fig. 5.13). We see that the $SL$ module trained using RL can achieve better performance than the analytical solution of IDM, and the $SL$ module trained using IL can achieve comparable performance as the IDM.

In the PAN architecture, theoretically one can use either RL or IL to train an attribute network. Further for IL, the training data can come from either human labeling or analytical solutions. In practice, the difficulty of the attribute module training is different if one uses different kinds of training approaches, and resulting performance can be different as well. Generally, RL training is the most difficult, but it has the best performance. IL with human labeling can achieve ideal performance, but the human labor cost would be a disadvantage. IL with analytically calculated labels is the easiest approach, but its performance is often not as good as RL. Therefore, the training approach shall be selected based on needs.

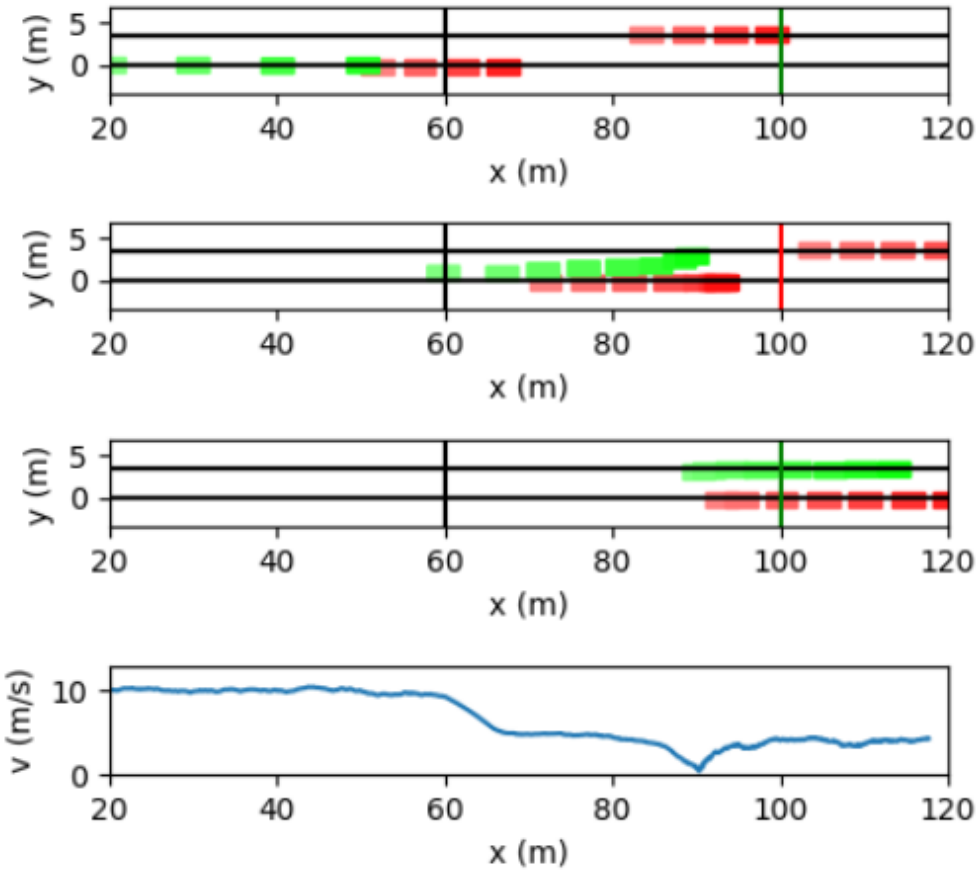


Figure 5.12: Behavior and speed profile of the autonomous vehicle in $LT \oplus OB_1 \oplus OB_2 \oplus OB_{re1} \oplus OB_{re2} \oplus TL \oplus SL$ task. The vertical green and red lines indicate the position and status of the traffic light, and the vertical black line indicates the position of the speed limit sign.

## 5.5 Real World Experiments

### 5.5.1 Experiment Setup

We validated the capability of the PAN policy to generate online driving commands for real autonomous vehicles in an experiment carried out in the Richmond Field Station of the University of California, Berkeley. The tested case is an obstacle avoidance task ($LT \oplus OB_1 \oplus OB_{re1} \oplus OB_{re2}$) with a static obstacle vehicle parked in front of the autonomous vehicle. The autonomous vehicle starts with a speed of $10m/s \approx 22mph$. The autonomous vehicle incorporates the GPS and the IMU sensors that can measure its states, while the obstacle vehicle states are assumed known. The screenshots of the experiment setting and

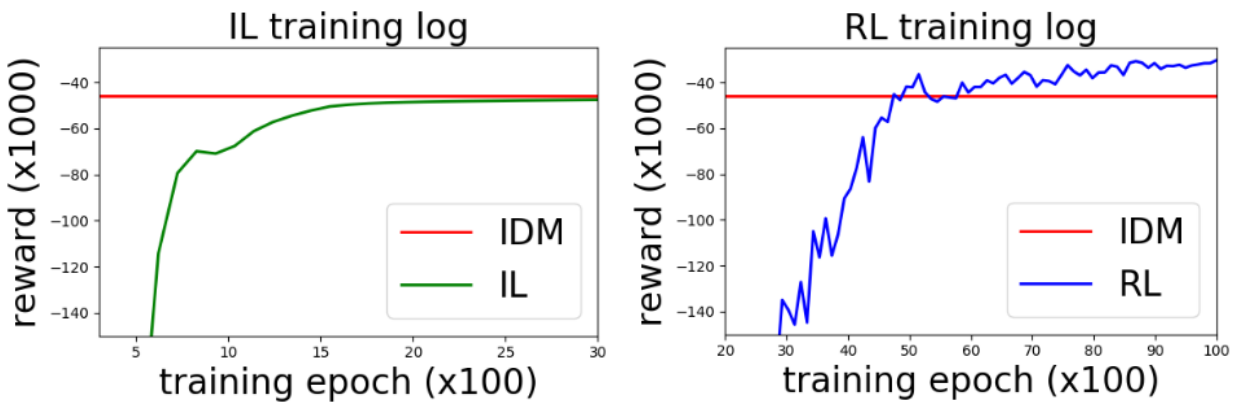


Figure 5.13: Left: The training log using IL. Right: The training log using RL.3.

the onboard tracking visualization are shown in Fig. 5.14.

For the motion planning part, a PAN policy with three pretrained $OB$ modules (one for the obstacle vehicle and two for the road edges) is applied to produce the longitudinal acceleration $a_0$ and the lateral yaw rate $\dot{\theta}_0$. For the longitudinal control, we use a PID controller to have the autonomous vehicle track the longitudinal speed profile derived using the $a_0$ commands. For the lateral control, we first have the PAN policy to generate an imaginary trajectory for 80 timesteps (1.6 seconds) in the simulator, then apply a tracking controller to produce the steering commands. In order to handle the sim-to-real modelling gap of the autonomous vehicle, we applied a disturbance observer (DOB)-based robust tracking controller to reject the model error and the other disturbance. The details are included in Chapter 7 (paper [118] [105]). In the experiment, we used the robot operating system (ROS) for collecting sensor data, transmitting information, and publishing the control commands.

### 5.5.2 Lane Tracking and Obstacle Avoiding Performance

The online performance of the PAN policy is stable and fast in the task, even though the task had not been trained before. The PAN policy node in the ROS publishes the commands at 50Hz, the imaginary trajectory used to produce the steering command publishes at a frequency of 5Hz, and the DOB-based robust tracking controller produces the steering commands at 50Hz. We performed the experiment 10 times with different starting positions and received a 10/10 success rate. Fig. 5.15 shows a typical experimental trajectory. The PAN policy, which serves as the motion planning part, can zero-shoot the new case and recursively produce robustly effective reference trajectories for the autonomous vehicle.

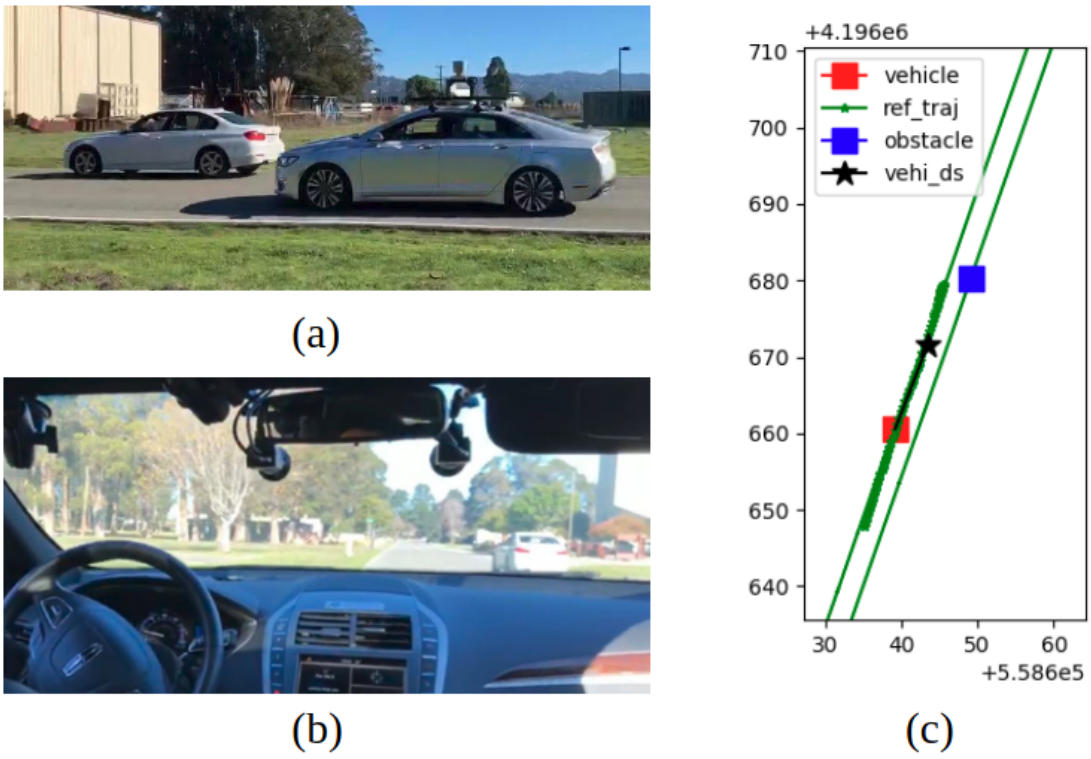


Figure 5.14: (a)(b) The experiment setting and the obstacle avoidance task; (c) the onboard tracking visualization screenshot.

## 5.6 Chapter Summary

The cascade attribute networks we proposed cannot handle a larger number of attributes. In this chapter, we present a new approach—attribute modularization—that decomposes complicated autonomous driving policies in a parallel manner. The attribute modules can be trained using either RL or IL; further, for IL, the expert data can come from either analytical theory or directly labelled by human. The attribute modules that are trained with data from one case can be transferred to other tasks. These properties provide simplicity and flexibility for the training of the neural network policies. Also, many unseen tasks can be solved in a zero-shot manner using the PAN. We further introduced the PAN architecture, in which the attribute modules work in parallel to combine their influences. We tested the capability of the PAN structure in a variety of simulation cases and carried out a real vehicle experiment to show the ability of the PAN policy to produce robustly effective onboard driving instructions.

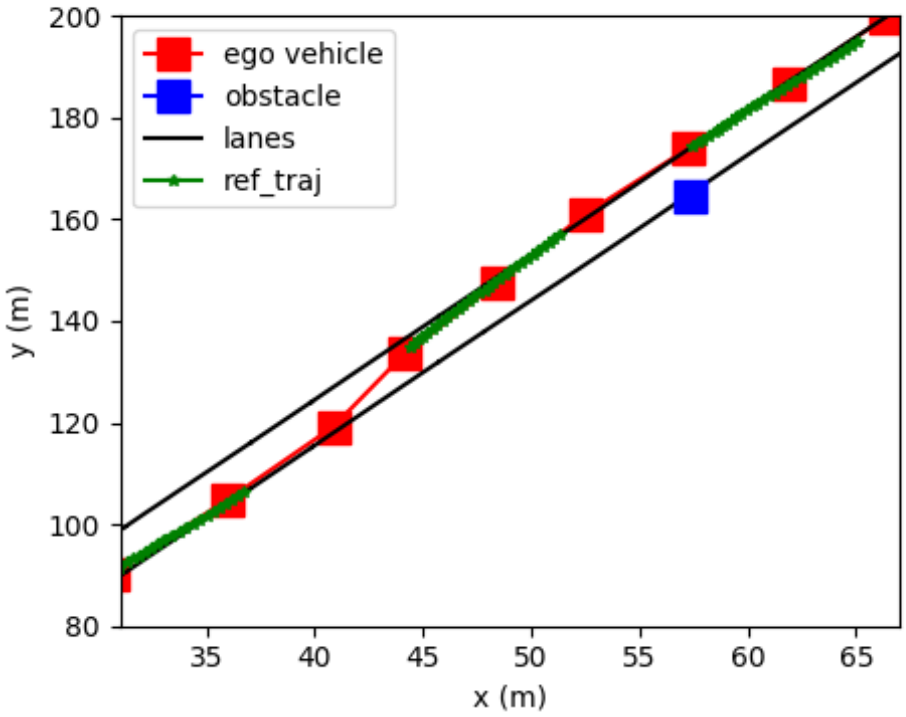


Figure 5.15: Behavior of the autonomous vehicle in the $LT \oplus OB_1 \oplus OB_{re1} \oplus OB_{re2}$ task in one of the real vehicle experiments. The red squares indicate the real trajectory of the autonomous vehicle, the blue block is the parked obstacle vehicle, and the green lines are a few reference trajectories generated by the PAN policy in the imaginary simulation.

# Part III

# Reasoning for Representations Outside the LbC Policies

# Chapter 6

# Inverse Vehicle Dynamics Adaptation for Driving Policy Transfer

Low-level controllers for autonomous vehicles need to be tuned for the different dynamics of real vehicles to achieve satisfying experiment performance. This is difficult and time consuming. Moreover, all the tuning and testing work has to be repeated from scratch for new vehicles. It is therefore worthwhile to investigate methodologies to adapt previously tuned controllers (either in simulation or real experiments) to new vehicles without further tuning. In this chapter, we introduce a meta learning-based inverse dynamics adaptation framework to solve this problem. In this framework, a driving policy is decomposed into two parts. First, we propose a vehicle-invariant kinematic representation for transfer and, second, the vehicle inverse dynamics are adopted to map the desired kinematic features to dynamic control commands. While the vehicle-invariant part is transferred directly, the inverse dynamics model is learned for each new vehicle. To boost learning efficiency, model-agnostic meta learning (MAML) is adopted to achieve faster inverse vehicle dynamics adaptation. We present experimental results on transferring the lane-keeping controller between vehicles with different dynamics properties to validate the proposed method.

## 6.1 Introduction

Many control policies are tuned for a specific dynamics or environment setting, which makes it difficult to adapt the policies to different settings or overcome the discrepancies between simulation and the real world. In practice, a lot of tuning is required when the learned policies are applied to a different situation. Therefore, we need to develop a system that can automatically adapt the policy tuned in one system setting to another system. We study this issue by developing transferable controllers for autonomous driving applications. Since there are a variety of different vehicles (with different dynamics models) that operate in various complicated scenarios, it would be challenging to tune and test the control systems for all possible cases.

To tackle such challenges, classic research has focused mainly on adaptive model-based control, including system identification and adaptive control [6]. However, such methods often require a sophisticated model that precisely captures the properties of the system dynamics. This requirement is usually not satisfied for vehicle dynamics because of the vehicle powertrain and tire dynamics complications; in practice, the controller parameters are tuned case by case.

In this chapter, we seek to handle this problem with machine learning. There is a significant body of work on this topic, which includes reinforcement learning for policy learning and transfer learning for policy adaptation. Transferring a reinforcement learning policy requires that the experience gained when learning in a source domain helps improve the learning in a target domain [106]. Therefore, a core associated problem is the reasoning of a task-invariant representation that serves as the transferable feature between tasks. We adopt the idea of the adaptation of the actions and lift the transferable action representation from the low-level dynamics action to a higher-level kinematics action. Concretely, apart from the forward system dynamics that map the current state action pair to the next state, we also learn an inverse dynamics model that maps from current and next states to an action, which achieves the transition between the two states. When deploying a driving policy to a new vehicle (target vehicle), we leverage the forward dynamics of the source vehicle to obtain the transferable kinematic action and leverage the inverse dynamics model of the target vehicle to obtain a desired action, which drives the target vehicle to achieve the desired kinematic action. Based on empirical driving practice, if the source vehicle and the target vehicle are similar, it can be assumed that the kinematic-level driving policy that is executed by the source vehicle can be reproduced by the target vehicle. Therefore, the driving policy transfer loop is closed, as long as the inverse dynamics model can be well approximated.

The inverse dynamics model of the target vehicle can be approximated in a data-driven manner. There are various model candidates for the approximator, such as the linear function [73, 124], the Gaussian process[16, 32], and deep neural networks [84][42]. We adopt the deep neural network model as the inverse dynamics approximator. The learning of a neural network inverse dynamics model can be slow as a supervised regression. Therefore, to further boost the adaption procedure, we adopt meta learning for fast adaption of the inverse dynamics model. Specifically, we adopt a model agnostic meta-learning algorithm (MAML) [40] to achieve a few-shot adaptation of the inverse dynamics model for the target vehicle. Experiments show that the meta learning approach can achieve fast adaptation using only one episode in the target domain.

## 6.2 Problem Setting and Reinforcement Learning Backgrounds

### 6.2.1 The Driving Policy Transfer Problem

In this chapter, we aim to deal with a driving policy adaptation problem for different vehicle dynamics, as shown in Fig. 6.1. For policy learning and evaluation purposes, we set up this problem in a high-fidelity vehicle dynamics simulator. In simulators, the links are usually modeled as reduced coordinate rigid bodies [39]. However, the simplified models are unable to capture some physical effects, such as area contact [43] and interaction with fluids [79]. However, more accurate simulators can be extremely computationally expensive and numerically ill conditioned. Therefore, we create a vehicle simulator based on a bicycle model with a Pacejka tire model [18] for our application.

The kinematic part of the complicated vehicle dynamics model used for the source and target vehicles are simulated using the continuous nonlinear bicycle model [85]:

$$\dot{v}_x = a_x \tag{6.1}$$

$$\dot{v}_y = -v_x\omega_z + \frac{1}{m}\left[F_f(\alpha_f,\mu)\cos(\delta) + F_r(\alpha_r,\mu)\right] \tag{6.2}$$

$$\dot{\omega}_z = \frac{1}{I_z}\left[aF_f(\alpha_f,\mu)\cos(\delta) - bF_r(\alpha_r,\mu)\right] \tag{6.3}$$

$$\dot{X} = v_x\cos\psi - v_y\sin\psi \tag{6.4}$$

$$\dot{Y} = v_x\sin\psi + v_y\cos\psi \tag{6.5}$$

$$\dot{\psi} = \omega_z \tag{6.6}$$

$$\dot{\delta} = \dot{\delta}, \tag{6.7}$$

where the state variables are the longitudinal speed $v_x$, lateral speed $v_y$, yaw rate $\omega_z$, vehicle global coordinates $X, Y$, yaw angle $\psi$, and steering angle $\delta$. The dynamics-level control commands are the longitudinal force applied onto the rear wheels and the steering rate, which simulate the throttle, brake, and steering operations of the driver. These dynamics-level control commands directly drive the nonlinear bicycle model inputs, which are the longitudinal acceleration $a_x$ and the derivative of steering angle $\dot{\delta}$. More complicated longitudinal dynamics models, including the powertrain and engine dynamics, are not taken into account in the simulator. There are magnitude saturation constraints defined for $a_x, \delta, \dot{\delta}$. The maximum magnitude of $\delta$ is $0.573rad$. The maximum magnitude of $\dot{\delta}$ is $0.927rad/s$. The maximum allowed magnitude of $a_x$ is $5m/s^2$. The other parameters of the nominal vehicle model in the source domain are shown in Table 6.1.

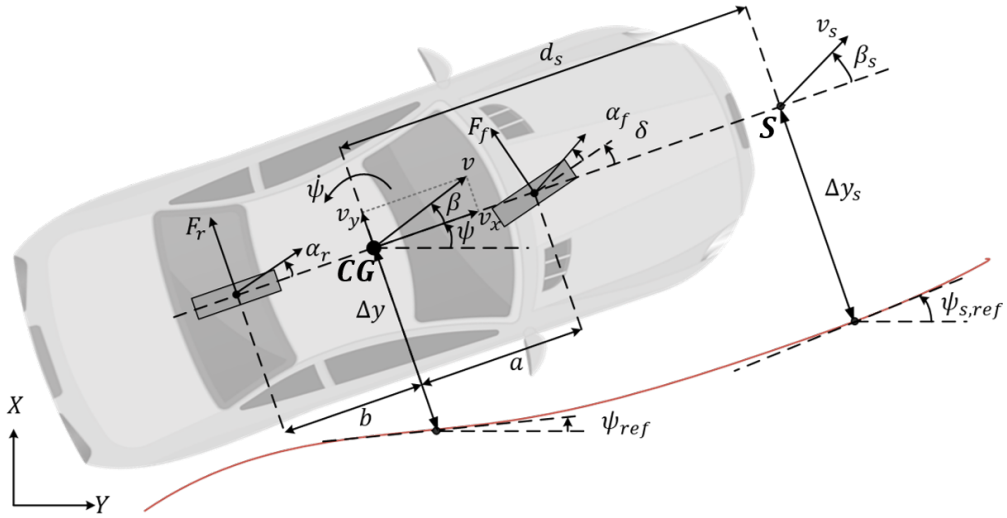


Figure 6.1: Illustration of the terminology for the simulated environment and the linear tracking model.

Table 6.1: Nominal vehicle dynamics parameters.

| | |
|---|---|
| Moment of Inertia $I_z$ | 3270 $kg \cdot m^2$ |
| Vehicle Mass $m$ | 1800 kg |
| Front Wheel to Center of Mass $a$ | 1.2 $m$ |
| Rear Wheel to Center of Mass $b$ | 1.65 $m$ |

Pacejka's Magic Formula [77] is used as the tire force function $F_f(\cdot)$, which is a function of slip angle $\alpha$ and road friction coefficient $\mu$:

$$F_f = \frac{\mu mga}{a+b} \sin(C \tan^{-1}(B\alpha_f - E(B\alpha_f - \tan^{-1}(B\alpha_f)))) \tag{6.8}$$

$$F_r = \frac{\mu mgb}{a+b} \sin(C \tan^{-1}(B\alpha_r - E(B\alpha_r - \tan^{-1}(B\alpha_r)))) \tag{6.9}$$

where coefficients $B = 10$, $C = 1.9$, $E = 0.97$ and $\mu = 1.0$.

The vehicle's task is to keep on a sine curve lane, as shown in Fig. 6.1. We observe the vehicle states $[v_x, v_y, \omega_z, \delta, F_{xR}/F_{xR,max}\alpha_f]$, and the tracking performance is represented using the errors $[\Delta y \Delta y_s \psi_{ref} \psi_{s,ref}]$, as shown in Fig. 6.1. When training the policy, we will reward for moving forward along the trajectory and punish for deviating from the centerline of the road; we terminate the simulation if the deviation is too large. Specifically, we define

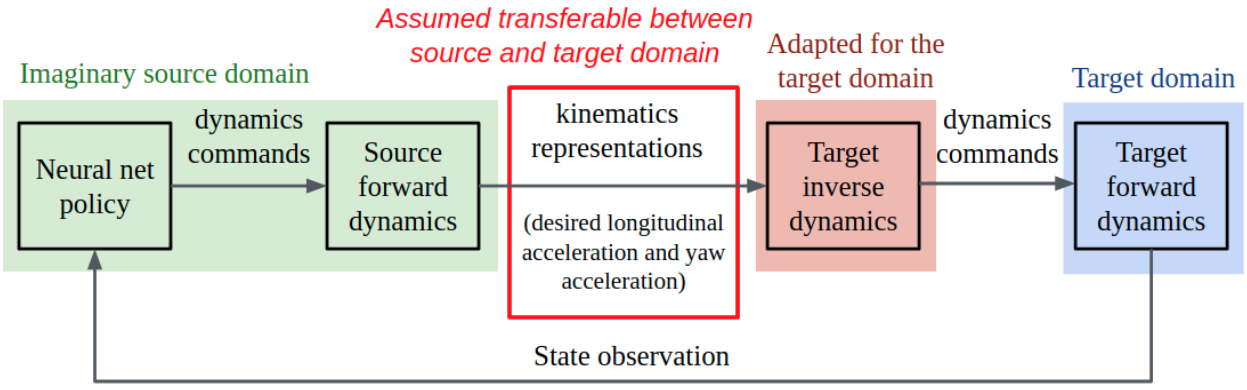


Figure 6.2: Architecture of the fast inverse vehicle dynamics adaptation system.

the vehicle's speed along and perpendicular to the reference curve: $v_{\parallel} = v\cos(\Delta\psi)$ and $v_{\perp} = v\sin(\Delta\psi)$. And the step tracking reward is defined to be:

$$r_{tra}(t) = v_{\parallel}(t) - |v_{\perp}(t)| - \eta \cdot \Delta y(t)^2, \tag{6.10}$$

## 6.3 Transfer of Kinematic Representation based on Meta Learning

The proposed dynamics adaptation framework based on the kinematic representations is shown in Figure 6.2. The adaptation process is as follows:

1. First, we train a driving policy using model-free reinforcement learning for a source vehicle, which maps the state observations to the dynamics actions: the longitudinal force that is applied onto the rear wheels and the steering rate. Both of these are proxies of the throttle, break, and steering commands. Specifically, proximal policy optimization (PPO) [97] is adopted.

2. We then use the forward dynamics model of the source vehicle to obtain the corresponding kinematics actions, the acceleration, and the yaw acceleration, derived by applying the dynamics actions.

3. When adapting to a target vehicle, we directly transfer the kinematic representation to the target domain. We then compute the desired future state of the target vehicle using the desired kinematic representation and feed the desired future state to the inverse model to obtain the desired dynamics action.

We can easily obtain the forward dynamics of the source vehicle based on the dynamics model in Section 6.2.1, but the neural network approximator of the inverse dynamics of the target has to be learned through experimenting on the target vehicle. The supervised regression learning of a neural network approximator can take a huge amount of labeled data. However, in practice, the inverse dynamics learning shall be completed within a few shots of experiments on the target vehicle. Therefore, we propose to use meta learning to boost the inverse dynamics adaptation process.

Meta learning and few-shot learning methods have been developed for specific tasks, such as generative modeling and image recognition [107, 38, 90, 111]. Prior meta-learning methods include learning a meta model that is suitable for adaptation [12, 13], learning both the weight initialization and the optimizer [89], and training a memory-augmented model on many tasks based on a recurrent neural network [95][75]. In this chapter, we adopt the model-agnostic meta-learning algorithm (MAML) [40] to boost the adaptation efficiency of the inverse dynamics model. MAML trains a meta model based on a variety of different vehicle dynamics and updates the weights in a gradient-based manner. The core idea of MAML is to learn a meta inverse dynamics model which, after a few gradient-descent-based adaptations based on a specific target vehicle, achieves minimal error compared with the true inverse dynamics model. It can be viewed as maximizing the sensitivity of the adaptation loss to the inverse dynamics model parameters.

Formally, suppose we use a neural network $f_\theta$ parameterized by $\theta$ to approximate the meta inverse dynamics model. The inverse dynamics model adaptation process for a specific target vehicle is a supervised regression problem. Suppose we have the supervision data of the input and output of the inverse dynamics of a specific target vehicle, with the data denoted $X$ and $Y$. After one step of gradient-descent based adaptation, the parameters $\theta^{'}$ for the updated inverse dynamics model is:

$$\theta_i^{'} = \theta - \alpha \bigtriangledown_\theta Loss(f_\theta(X), Y) \tag{6.11}$$

where $\alpha$ is the step size. $\alpha$ can be fixed as a hyperparameter or meta learned. The $Loss(\cdot)$ is the loss function of the regression, such as the mean square error function. The meta inverse dynamics model parameter $\theta$ is learned in such a way that the performance of $f_{\theta'}$ is optimized based on the data $D$ collected for a large variety of target vehicles. Thus, the meta objective is defined as follows:

$$\min_\theta \sum_{X,Y\sim D} Loss(f_{\theta_i'}(X), Y) = \min_\theta \sum_{X,Y\sim D} Loss(f_{\theta - \alpha \bigtriangledown_\theta Loss(f_\theta(X),Y)}(X), Y) \tag{6.12}$$

While meta optimization is conducted on the parameters $\theta$, the objective is generated using the updated parameters $\theta^{'}$; so the optimization is done in such a way that within a few gradient steps, it can produce maximally effective behavior on a new task. The optimization algorithm for meta optimization is stochastic gradient descent, and the model parameter $\theta$

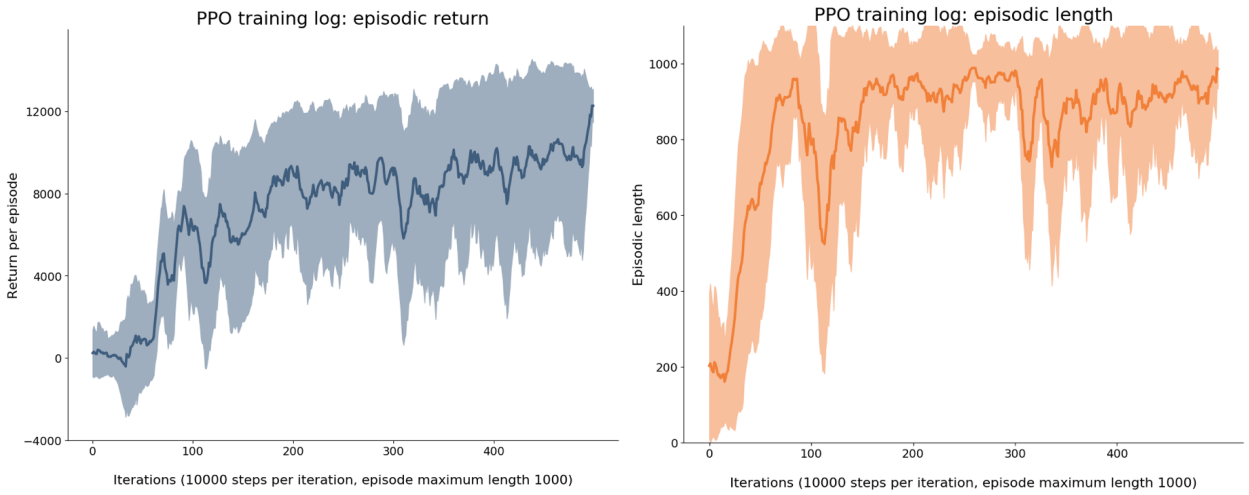


Figure 6.3: The PPO training logs. Left: In terms of episodic return; Right: In terms of episodic length.

is updated as follows:

$$\theta \leftarrow \theta - \beta \bigtriangledown_{\theta} \sum_{X,Y \sim D} Loss(f_{\theta_i'}(X), Y) \tag{6.13}$$

where $\beta$ is the meta step size and is usually set to be much smaller than $\alpha$.

## 6.4 Simulation and Results

We carry out a series of simulation experiments to validate the effectiveness of our proposed method. First, we train the policy network using proximal policy optimization algorithms (PPO). We randomly initialized the starting location on the track and set the episode length to be 1,000 steps (corresponding to 20 seconds). For the policy training performance, Fig. 6.3 illustrates the episodic return and episodic length over iterations, and empirical evidence validates that the vehicle can track the sine curve lane quite well if the episodic return is higher than 9,000.

The forward dynamics can be obtained directly from the simulator and are combined with the policy network to produce the kinematic-level action. In order to generate the training dataset for the meta learning of the inverse dynamics model, we randomize the dynamics parameters with $\pm 10\%$ error with respect to the source vehicle parameters, and we generate 20 simulated target vehicles. Then the data is utilized for the learning of inverse dynamics model. To prove the effectiveness of the proposed method, we compare MAML with the two following methods:

1. Pretrain-fine-tune: The whole dataset is used as a training set and gradient descent-based regression is adopted. When adapting to a specific target vehicle, the inverse dynamics model is fine-tuned for the target vehicle. It serves as a baseline to verify the performance of MAML.

2. MAML with first-order approximation: When computing the gradient of the meta objective function, the second-order term is omitted. The purpose of the approximation is to shorten the computational time required to compute the gradient.

For the adaptation to a specific target vehicle, we collect data using the target vehicle, and we use it to fine-tune the inverse dynamics model. For each episode, we can obtain a dataset that contains 1,000 steps of data points. We fine-tune for 100 iterations, with each iteration based on a randomly sampled data batch with a batch size of 100.

The comparison of the three approaches in the meta training phase is shown in the training log in Fig. 6.4. In order to have a closer look at the later iterations, we zoom in and obtain Figure 6.5. The solid lines represent the losses prior to fine-tuning at each iteration. And the dashed lines represent the losses after one step of fine-tuning for the target vehicle of the training set. We can see that losses decrease after fine-tuning for models trained using MAML and approximate MAML methods, while losses increase after fine-tuning for the pretrained model. This proves that MAML and approximate MAML learn a prior model that is suitable for fast fine-tuning, while the vanilla pretraining overfits to the meta dataset, and fine-tuning the pretrained model can hurt performance.

Finally, we test the online adaptation performance of the three methods. The fine-tune loss curve is shown in Fig. 6.6, and the return curve is shown in Fig. 6.7. The two figures prove that the model meta trained by MAML and approximate MAML can be adapted to a new vehicle in one episode; this is much better than the pretrain-fine-tune method, which takes two episodes for fine-tuning. This is because the pretrain-fine-tune method overfits to fine-tuning data for each episode and results in a decreased episodic return after the fine-tuning.

## 6.5 Chapter Summary

In this chapter, we propose transferring driving policy through fast inverse vehicle dynamics adaption, assuming the transferability of the vehicle-invariant kinematic representations. The driving policy that produces the kinematic-level actions is trained using proximal policy optimization (PPO). Although the inverse dynamics can be obtained using vanilla regression, this process can take lots of episodes, which is cost prohibitive for real vehicles adaptation. Therefore, we propose to apply meta learning for faster inverse dynamics adaption. Specifically, we applied model agnostic meta learning (MAML). The experimental results validate the efficiency of our method to achieve fast adaptation in only one episode.

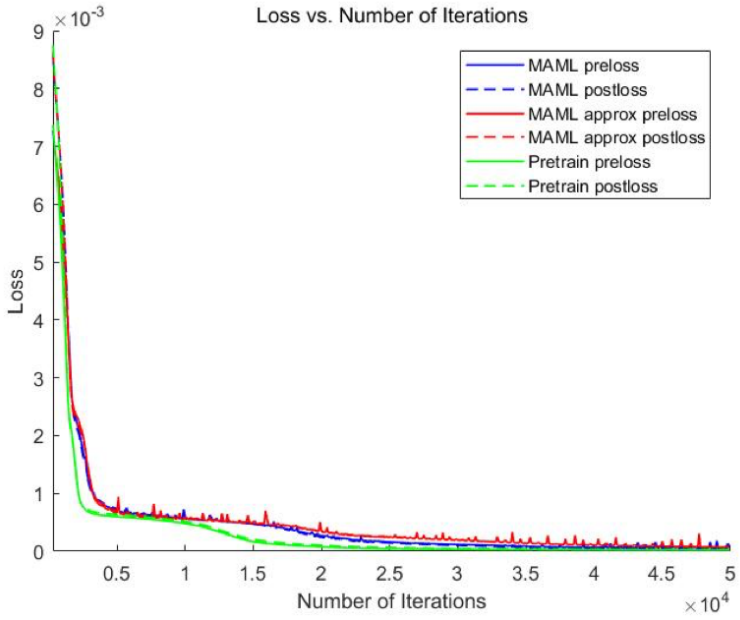


Figure 6.4: The training log comparison of the three approaches in the meta training phase.

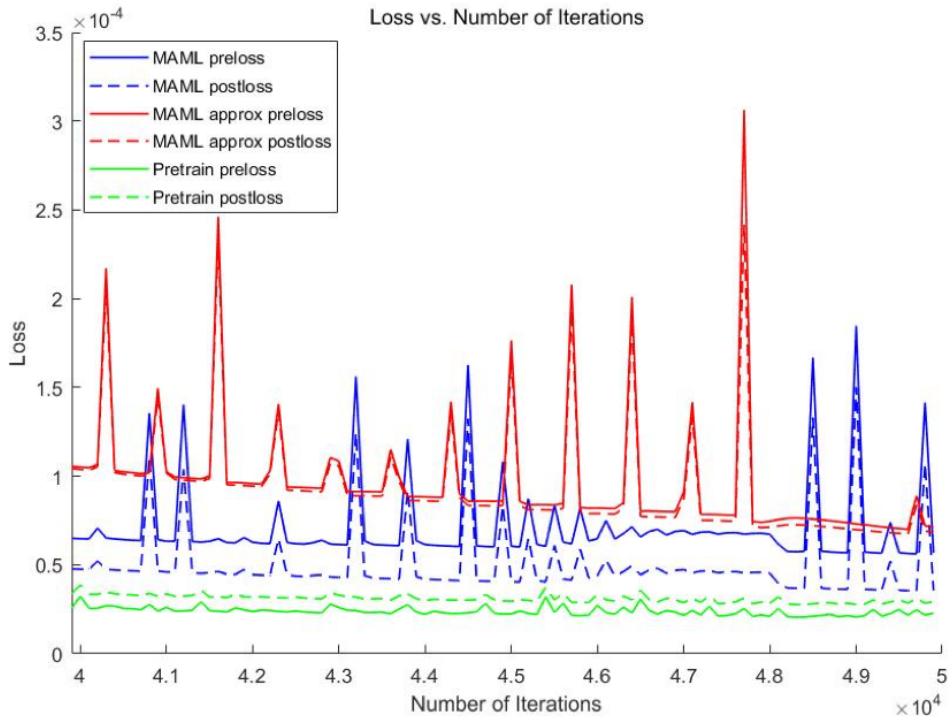


Figure 6.5: Zoom in of the training log comparison of the three approaches in the meta training phase.

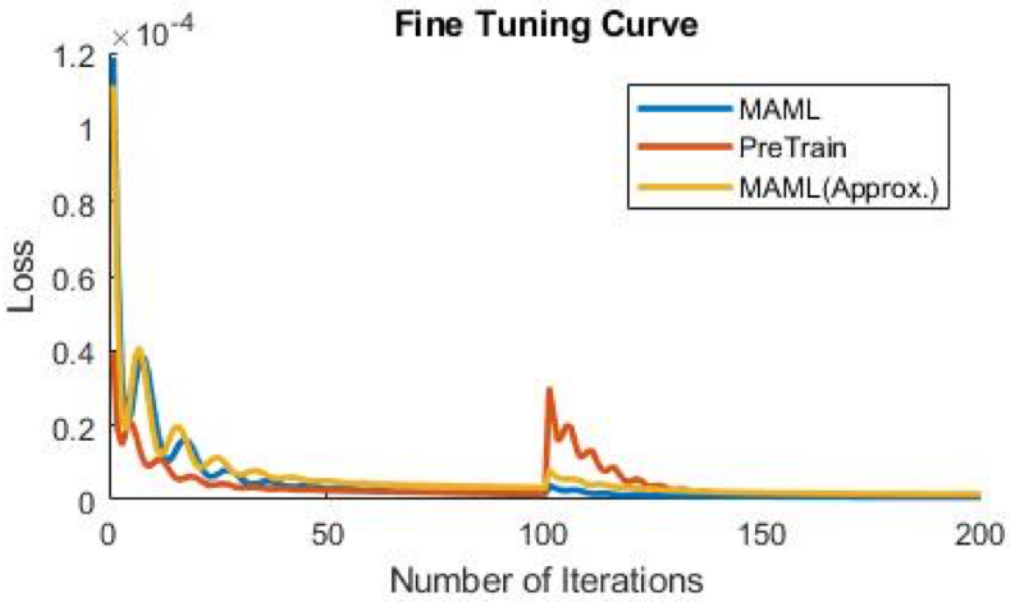


Figure 6.6: The inverse dynamics model regression loss vs. fine-tuning iterations in an online adaptation process. Note that for each episode, we obtain a dataset that contains 1,000 steps of data points and use it to fine-tune for 100 iterations. Therefore, at iteration 100, the next episode is used.

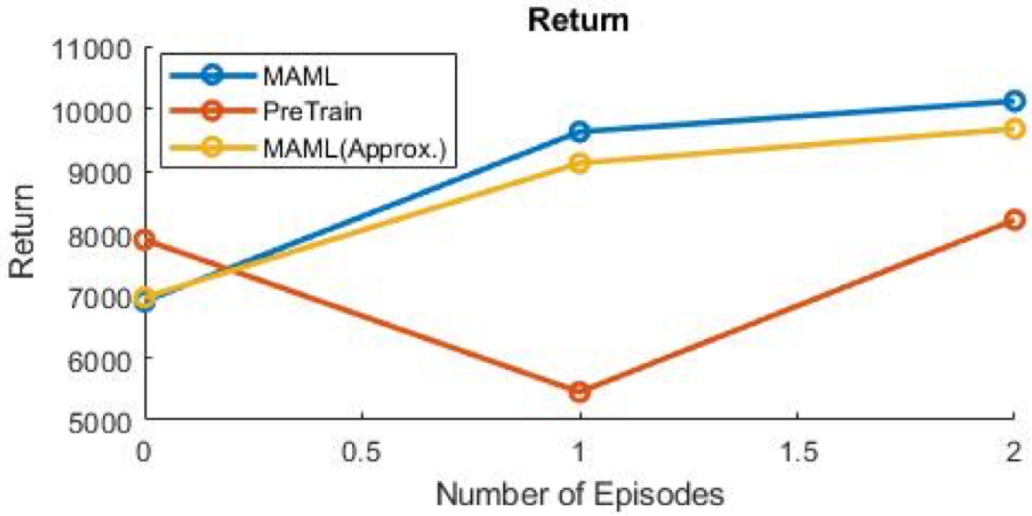


Figure 6.7: The episodic return vs. number of episodes in an online adaptation process.

# Chapter 7

# Autonomous Driving Policy Transfer based on Robust Control

Although deep reinforcement learning (deep RL) methods have many strengths that could potentially be used in autonomous driving, applying them has been difficult due to the modeling gap between the source (training) domain and the target (deployment) domain. Current policy transfer approaches generally limit usage of uninterpretable neural network representations as transferred features. In this chapter, we propose transferring concrete kinematic quantities in autonomous driving. We propose a robust control-based (RC) generic transfer architecture, which incorporates a transferable hierarchical RL trajectory planner and a robust tracking controller based on disturbance observer (DOB). The deep RL policies trained with a known nominal dynamics model are transferred directly to the target domain; DOB-based robust tracking control is applied to tackle the modeling gap including the vehicle dynamics errors and the external disturbances such as side forces. We provide simulation and experiment results validating the capability of the proposed method to achieve zero-shot transfer across multiple driving scenarios, such as lane keeping, lane changing, and obstacle avoidance.

## 7.1 Introduction

Learning intelligent and reliable driving policies has been an ongoing challenge for both deep learning and control. Although conventional planning-control [64] and imitation-oriented learning [83, 17, 114, 74] approaches have the capability of controlling autonomous vehicles, deep reinforcement learning-based (deep RL) methods show promise in tackling more complicated and interacting scenarios that conventional methods cannot solve (as well as rare cases outside the demonstration dataset for the supervised imitation learning). Therefore, we see an increasing number of efforts to apply deep RL methods to learn lane-keeping policies with low-dimensional feature vectors or image pixels as inputs [66, 71, 94].

While the vast exploration and flexible hierarchical configurations enable deep RL meth-

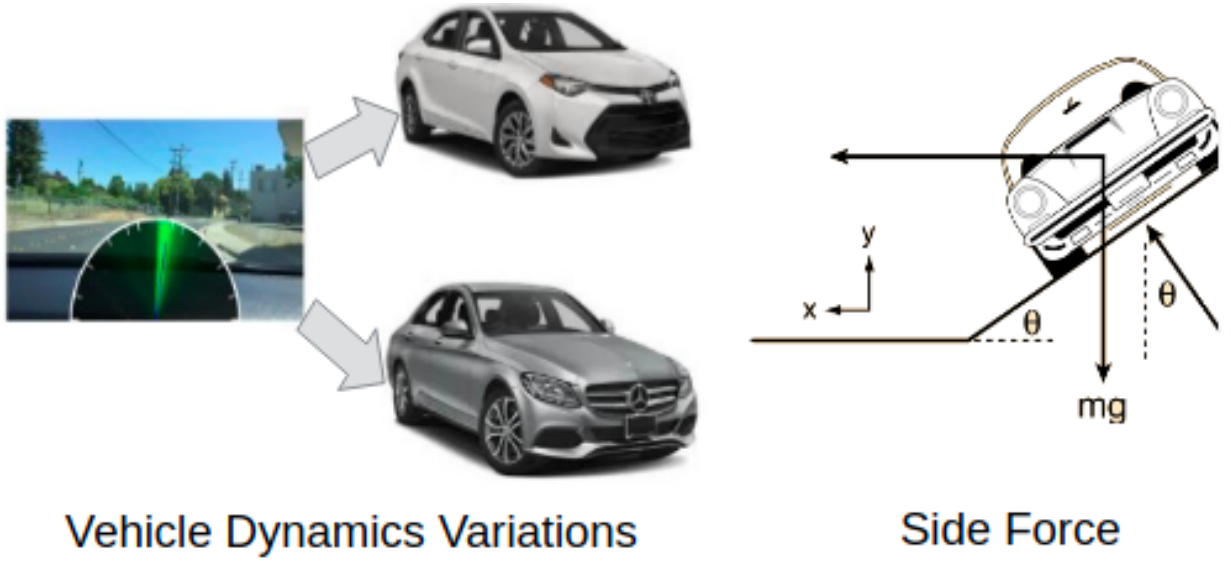


Figure 7.1: Examples of dynamics variations across autonomous driving training and testing domains. Left: Vehicle dynamics difference between different simulated real vehicles. Right: External side force disturbances in the testing domain due to body incline.

ods to obtain versatile policies more easily, robustness is a drawback that prevents the application of deep RL in autonomous driving. Specifically, pre-optimized deep RL policies are overspecialized to training vehicle settings and, thus, often fail when the target vehicle has dynamics variation or when the target vehicle is affected by force disturbances due to strong winds or body inclines (Fig. 7.1). These differences between the source and target settings together are called the modeling gap. As the modeling gap is an inevitable barrier to the deployment of deep RL in autonomous driving, we aim to bridge this gap by achieving fast and safe transfer of deep RL autonomous driving policies.

Prior efforts in deep RL attempt to achieve this objective using transfer learning and meta learning. Various contributions in this area have indeed found source policies to work in the target settings, but their application in autonomous driving is limited due to safety concerns. Overall, such deep RL transfer methods embed transferable representations into uninterpretable neural networks and hope for the best in the target domain; thus, they are not transparent and reliable. We seek to solve the transfer problem using an alternative tool—robust control (RC)—and propose a generic RL-RC transfer framework. In this framework, deep RL policy is applied to an imaginary setting in the source domain to generate a reference trajectory for the target vehicle. A robust controller is applied to track the reference trajectory tolerating the modeling gap. A method developed on an idea similar to the one in this chapter is in [46]. In this work, an MPC controller is designed to stabilize the target system around the nominal trajectory generated by consecutively applying the policy in the source system. Theorems on tube-based MPC ensure that the states are

bounded under certain modeling errors. However, the bound cannot be explicitly found, and no asymptotic stability can be guaranteed. Moreover, solving the on-line optimization problem is computationally expensive, while robust controllers are usually easier and faster for the trajectory tracking problem of automated vehicles.

The proposed framework is generic in that: (1) it can be generalized to any deep RL control policy transfer tasks; and (2) various kinds of robust controllers can be used for tracking. Compared to other transfer learning methods, the proposed framework has two fundamental advantages:

1. The transfer of the interpretable kinematic features makes the transfer framework transparent and reliable;

2. Stability and response in time and frequency domains can be well defined and analyzed.

The contribution of the work in this chapter is fourfold. First, we propose a generic approach for deep RL driving policy transfer. Second, we implement a hierarchical RL model which serves as the transferable trajectory planner. Third, we develop a disturbance observer-based (DOB) robust tracking controller so as to actively reject the disturbances induced by the modeling gap. Finally, we report simulation and experiment results validating that the RL-RC architecture can zero-shoot transfer the policy under certain levels of parameter variation and external disturbances.

## 7.2 Design of the Transferable Representation

The proposed RL-RC policy transfer architecture consists of a deep RL-based high-level planning module and an RC-based low-level tracking controller, as shown in Fig. 7.2. The overall methodology consists of the off-line deep RL policy training and the online policy transfer. In the off-line source domain, we pretrain a deep RL policy mapping the perception input to control commands, which can drive the source vehicle to produce a trajectory completing the control task.

For the online transfer in the target domain, Fig. 7.2 shows how the system works to transfer the source driving policies. In short, the RL-RC system performs closed-loop tracking of a finite-horizon previewed trajectory generated by the pretrained policy. Specifically, at each time step, the target agent obtains the perception observation. Then, according to this observation, the system constructs an imaginary source agent in the same setting as the target agent. In the imaginary source domain, the pretrained policy is used to control the imaginary source agent to perform the driving task for a finite horizon, resulting in a trajectory of kinematic states. Based on empirical driving practice, if the source vehicle and the target vehicle are similar vehicles, it can be assumed that the reference trajectory generated using the source vehicle can serve as the transferable representation and can be tracked by the target vehicle.

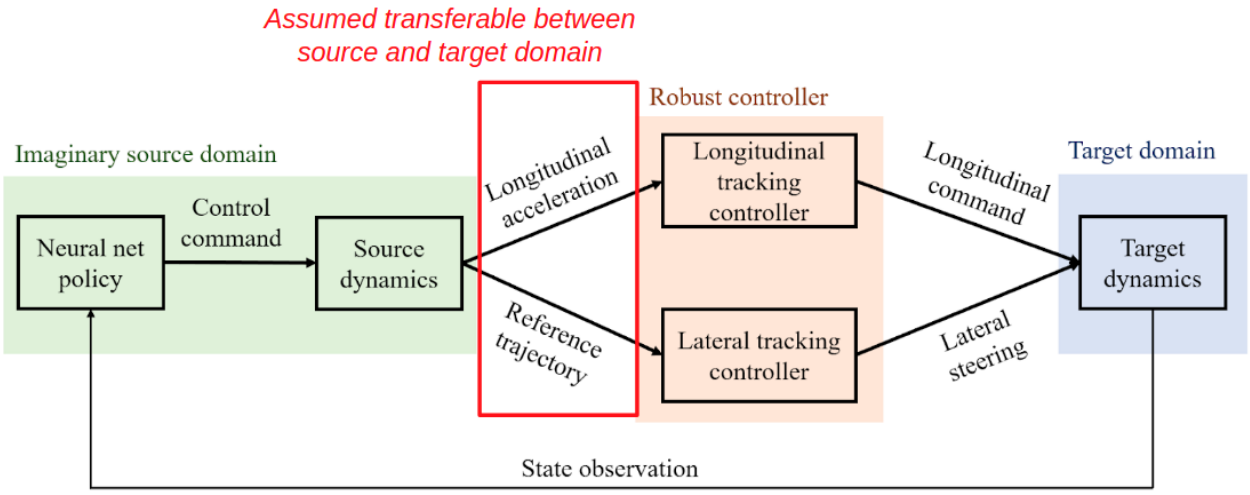


Figure 7.2: The RL-RC architecture.

Therefore, we adopt the reference trajectory as the reference for the target vehicle. Given the reference trajectory, the target agent uses a closed-loop robust tracking controller to produce the actual control command for the target vehicle. As the target environment makes one step forward, the observation for the new timestep is collected, and the system repeats the same procedure. In this architecture, the kinematic features are transferred without change, and the modeling gap is compensated by RC.

To generate the reference trajectory in the imaginary source domain, one can use the cascaded high-level planner, which consists of the pretrained policy and the source environment, or train a deep RL policy network that directly maps perception input to the trajectory of kinematic states. We choose the latter approach so as to optimize the policy for the whole procedure, enabling both dynamical feasibility and optimality of the planned trajectory.

There are two key underlying assumptions for the RL-RC system: (1) the trajectory planned by the imaginary source agent is comparably satisfying for the target task; and (2) it is feasible for the target vehicle to track the trajectories produced by the source vehicle. These assumptions are reasonable, as the source and target vehicles and settings are similar, and the RL-RC is effective for such cases.

Our proposed RL-RC framework is generic in terms of the following aspects:

1. This system can perform policy transfer for various kinds of tasks in autonomous driving. It can also be applied in other robotics and control scenarios if the assumptions are satisfied and a robust controller can be designed. In this work, we evaluate the RL-RC approach for three typical driving tasks: lane keeping (LK), lane changing (LC), and obstacle avoidance (OA).

2. The proposed method places no restrictions on the structure of the deep RL policy. In

this chapter, we use a hierarchical RL model to generate reference trajectories in the imaginary source domain.

3. Any kind of RC method is compatible with the RL-RC architecture. Among various RC algorithms, we use a disturbance observer-based (DOB) tracking controller to actively reject the disturbance induced by modeling error.

## 7.3 The Modeling of the Transfer Dynamics

We adopt the nonlinear bicycle model for the vehicle dynamics modeling, as described in the previous chapter in Section 6.2.1. The model is illustrated in Fig. 7.3. We define the control inputs to be the longitudinal acceleration $a_x$ and the derivative of steering angle $\dot{\delta}$. The seven state variables are longitudinal speed $v_x$, lateral speed $v_y$, yaw rate $\omega_z$, global coordinates $X, Y$, yaw angle $\psi$, and steering angle $\delta$. In the simulation part, to obtain the simulated vehicle dynamics with high fidelity, we include tire force saturation and restrict magnitudes of $a_x, \delta, \dot{\delta}$ to reasonable ranges. Moreover, we omit longitudinal powertrain dynamics, assuming nearly perfect longitudinal tracking by a low-level longitudinal controller.

Apart from the vehicle dynamics, we also define terms related to the driving scenarios. We define lateral displacements, $\Delta y$ and $\Delta y_s$, and yaw errors, $\Delta\psi$ and $\Delta\psi_s$, to represent the tracking errors relative to lane or reference trajectory (Fig. 7.3). $\Delta y$ and $\Delta\psi$ are defined with respect to the center of gravity (CG), and $\Delta y_s$ and $\Delta\psi_s$ are defined at a point $S$ specified by a look-ahead distance $d_s$. We denote the angle of the vehicle speed $v$ as $\psi_v = \psi + \beta$, where $\beta$ is the angle between vehicle speed and yaw. Similarly, the angle of $v_s$ is $\psi_{v_s} = \psi + \beta_s$. The yaw errors are the angle between velocity and the tangent direction of the reference curve, specifically, $\Delta\psi = \psi_v - \psi_{ref}$ and $\Delta\psi_s = \psi_{v_s} - \psi_{s,ref}$. With the yaw error $\Delta\psi$, one can easily derive the ego vehicle's speed along and perpendicular to the reference curve: $v_{\parallel} = v\cos(\Delta\psi)$ and $v_{\perp} = v\sin(\Delta\psi)$.

For the collision avoidance task, we consider a simple scenario with a single surrounding vehicle that perfectly tracks the lane with constant speed $v_{srd}$. We denote the relative longitudinal and lateral position of the surrounding vehicle by $\Delta x_{srd}$ and $\Delta y_{srd}$.

## 7.4 Definitions of the Hierarchical Markov Decision Process (MDP) and RL Policies

For the ease of the Markov Decision Process (MDP) and the RL modeling, in both the source domain and the target domain, we discretize the vehicle dynamics using the forward Euler method with $T_s = 0.02s$. In order to learn a variety of different driving behaviors, we define a general formulation of MDP and RL policies for a series of driving tasks. Since different tasks have different observations, we define clusters of observations as entities. Specifically, we define the observation of the ego vehicle state to be $o_{vh} = [v_x\ v_y\ \omega_z\ \delta]^T$. For each lane,

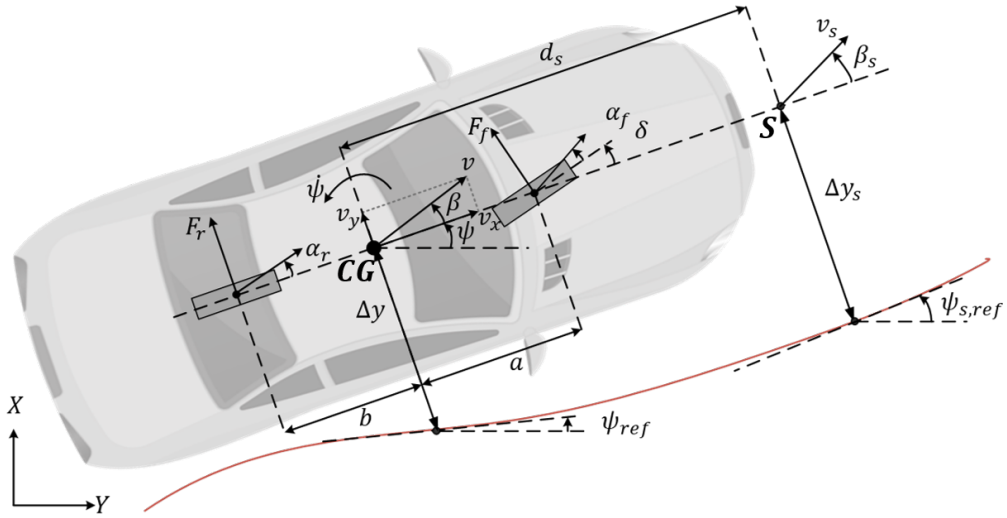


Figure 7.3: Illustration of the terminology for the simulated environment and the linear tracking model for the controller.

denoted $l_i$, $i \in N$, we define tracking observations as $o_{ref,i} = [\Delta y_i \ \Delta\psi_i \ \Delta y_{s,i} \ \Delta\psi_{s,i}]^T$. The lane that the ego vehicle is tracking is denoted using $l_{i^*}$. The obstacle observation is $o_{srd} = [v_{srd} \ \Delta x_{srd} \ \Delta y_{srd}]^T$. For simplification, we assume perfect and deterministic observations. For all the tasks, control command $a_{vh} = [a_x \ \dot{\delta}]^T$ is defined as the actions in the MDP. The actions are continuous and normalized by their maximum allowable values.

For performance evaluation, we define the step-tracking reward function at timestep $t$:

$$r_{tra}(t) = v_{\parallel,i^*}(t) - |v_{\perp,i^*}(t)| - \eta \cdot \Delta y_{i^*}(t)^2, \tag{7.1}$$

where $\eta > 0$ is an importance factor. The step rewards are discounted with factor $\gamma$. An episode is defined to have maximum of 1,000 timesteps. The episode will be terminated when there is large deviation from the lane or collision constraints are violated. Meanwhile, the agent will be punished with a large negative reward, denoted $r_{dev}$ and $r_{col}$. The interfaces of lane keeping (LK), lane changing (LC), and obstacle avoidance (OA) are shown in Table. 7.1.

In our implementation, we apply a hierarchical RL model that can modularize distinct driving attributes. The attributes refer to driving behaviors such as obstacle detection, lane selection, and lane tracking. Table 7.2 and Fig. 7.4 gives the detailed module interfaces and their usage. Theoretically, all the three basic modules can be optimized using any kind of method including deep RL. In the implementation, both lane-selection and obstacle-detection modules are rule based and defined and optimized for the sake of simplicity. The lane-tracking module is optimized using model-free deep RL. The benefit of such hierarchical implementation over end-to-end training is that basic attribute modules are much easier to optimize compared to end-to-end training of complicated high-level driving policies.

## 7.5 DOB-based Tracking Controller

DOB is a robust control technique to reject disturbances with guaranteed robust stability for linear systems [26]. Moreover, given a stabilizing nominal controller, Q-filter in DOB can be an arbitrary stable filter to make the sensitivity function shaped as desired [27]. It has been applied to robust lateral trajectory tracking, and experiments [88] have verified its effectiveness.

We design the reference trajectory as a sequence of $(X, Y, \psi_v, v_x)$. While the longitudinal speed profile is followed directly by executing corresponding $a_x$, the lateral path defined by the series of $(X, Y, \psi_v)$ is tracked by the robust lateral tracking controller. To design the controller, first we need to obtain an approximate linear model for the tracking problem. We adopt the constant speed linear bicycle model for lateral dynamics [85] to the continuous version of the nonlinear bicycle model defined in Section 6.2.1. The linearized version model is:

$$\frac{d}{dt}\begin{bmatrix} v_y \\ \psi \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & a_{13} \\ 0 & 0 & 1 \\ a_{31} & 0 & a_{33} \end{bmatrix} \begin{bmatrix} v_y \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} b_1 \\ 0 \\ b_3 \end{bmatrix} \delta \tag{7.2}$$

Table 7.1: Definition of driving tasks.

| Task | Observation | Evaluation |
|---|---|---|
| LK | $o_{vh}, o_{ref,i^*}$ | $\Sigma\gamma^t r_{tra}, r_{dev}$ |
| LC | $o_{vh}, \{o_{ref,i}\}, i^*$ | $\Sigma\gamma^t r_{tra}, r_{dev}$ |
| OA | $o_{vh}, \{o_{ref,i}\}, o_{srd}$ | $\Sigma\gamma^t r_{tra}, r_{dev}, r_{col}$ |

Table 7.2: Definition of RL modules.

| Control module | Input | Output |
|---|---|---|
| Lane tracking | $o_{vh}$, $o_{ref,i^*}$ | $a_{vh}$ |
| Lane selection | $\{o_{ref,i}\}$, $i^*$ | $o_{ref,i^*}$ |
| Obstacle detection | $o_{vh}$, $o_{srd}$ | $i^*$ |

where

$$a_{11} = -\frac{\mu C_{\alpha f} + \mu C_{\alpha r}}{m v_x} \tag{7.3}$$

$$a_{13} = -v_x - \frac{\mu C_{\alpha f} l_f - \mu C_{\alpha r} l_r}{m v_x} \tag{7.4}$$

$$a_{31} = \frac{-\mu C_{\alpha f} l_f + \mu C_{\alpha r} l_r}{I_z v_x} \tag{7.5}$$

$$a_{33} = -\frac{\mu C_{\alpha f} l_f^2 + \mu C_{\alpha r} l_r^2}{I_z v_x} \tag{7.6}$$

$$b_1 = \frac{\mu C_{\alpha f}}{m} \tag{7.7}$$

$$b_3 = \frac{\mu C_{\alpha f} l_f}{I_z} \tag{7.8}$$

where $C_{\alpha f}$ and $C_{\alpha r}$ are the cornering stiffness of front and rear wheels, respectively. Define:

$$A = \begin{bmatrix} a_{11} & 0 & a_{13} \\ 0 & 0 & 1 \\ a_{31} & 0 & a_{33} \end{bmatrix}, B = \begin{bmatrix} b_1 \\ 0 \\ b_3 \end{bmatrix}, x = \begin{bmatrix} v_y \\ \psi \\ \dot{\psi} \end{bmatrix} \tag{7.9}$$

and one can get a clearer linear vehicle dynamics form as:

$$\dot{x} = A \cdot x + B \cdot \delta, \tag{7.10}$$

where the state variable is $x = [v_y \ \psi \ \dot{\psi}]^T$. Under the small angle assumption, $\beta \approx v_y / v_x$, we approximate $\beta_s$ as:

$$\beta_s \approx \frac{v_x \beta + d_s \dot{\psi}}{v_x} = \frac{1}{v_x} v_y + \frac{d_s}{v_x} \dot{\psi}. \tag{7.11}$$

Thus:

$$\psi_{v_s} = \psi + \beta_s = \begin{bmatrix} \frac{1}{v_x} & 1 & \frac{d_s}{v_x} \end{bmatrix} \begin{bmatrix} v_y \\ \psi \\ \dot{\psi} \end{bmatrix} = C \cdot x. \tag{7.12}$$

Following [88], the derivative of $\psi_{s,ref}$ is approximated as $\dot{\psi}_{s,ref} \approx v_x \kappa_{s,ref}$, where $\kappa_{s,ref}$ refers to the curvature of the curve at the reference point. Consequently:

$$\Delta \dot{\psi}_s = \dot{\psi}_{v_s} - \dot{\psi}_{s,ref} \approx \dot{\psi}_{v_s} - v_x \kappa_{s,ref}. \tag{7.13}$$

Finally, $\Delta \dot{y}_s$ can be approximated as $\Delta \dot{y}_s \approx v_x \Delta \psi_s$.

Using the equations above and applying forward Euler discretization, the overall tracking model can be obtained as the block diagram shown in Fig. 7.5, where $T_s$ is the sampling time,

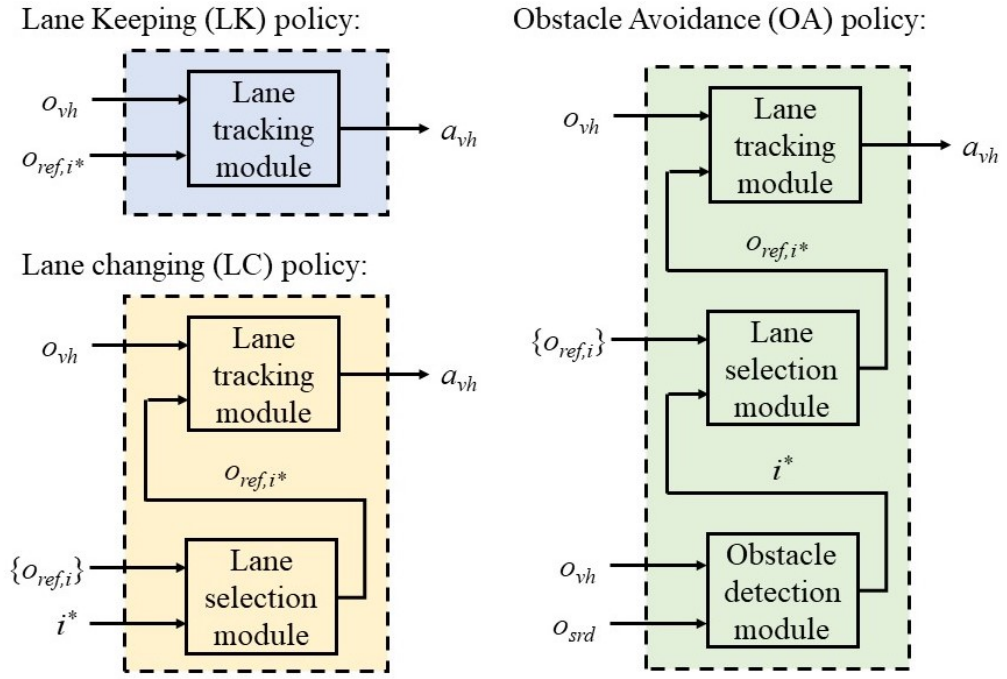


Figure 7.4: Usage of hierarchical RL modules to assemble policies for the lane keeping (LK), lane changing (LC), and obstacle avoidance (OA) tasks.

0.02 second, and $\frac{T_s z^{-1}}{1-z^{-1}}$ is the transfer function of the discretized integrator. $G_v(z^{-1})$ is the vehicle dynamics that map $\delta$ to $\psi_{v_s}$, of which we can derive the nominal transfer function $G_{nv}(z^{-1})$ using (7.10) and (7.12). As shown in the diagram, $\psi_{s,ref}$ can be considered a disturbance to the system.

The robust controller is designed based on the nominal model of the source vehicle. First, we design a proportional feedback controller as: $u_c = -k_1\Delta\psi_s - k_2\Delta y_s$. For analysis, we can further write the control law as:

$$u_c = -k_2(\frac{k_1}{k_2} + \frac{v_x T_s z^{-1}}{1-z^{-1}})\Delta\psi_s = -k_2 C_1(z^{-1})\Delta\psi_s \tag{7.14}$$

where $C_1(z^{-1}) = \frac{k_1}{k_2} + \frac{v_x T_s z^{-1}}{1-z^{-1}}$. In the implementation, we hold constant $\frac{k_1}{k_2}$ and tune $k_2$ to achieve stability of the closed-loop system using root locus. We inspect stability for various $v_x$ such that the resulting controller can stabilize the vehicle for the range of velocity in the given driving tasks.

Then a DOB is added to the nominal feedback controller. The block diagram for the overall closed-loop system is shown in Fig. 7.6. DOB is inserted between $C_1(z^{-1})$ and $k_2$ so that disturbance due to modeling error between $\Delta y_s$ and $\Delta\psi_s$ can be rejected. $P_n(z^{-1})$ and $\hat{P}_n(z^{-1})$ are the nominal plant and nominal plant without delay, defined as:

$$P_n(z^{-1}) = z^{-2}\hat{P}_n(z^{-1}) = G_{nv}(z^{-1})C_1(z^{-1}) \tag{7.15}$$

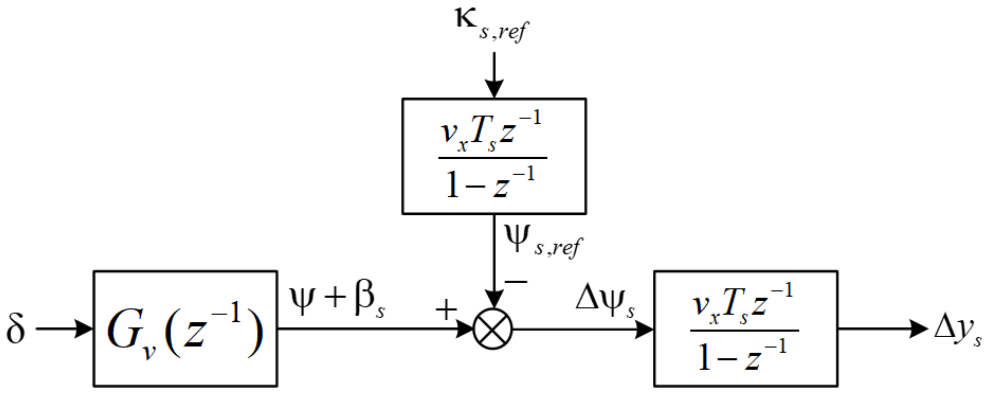


Figure 7.5: Block diagram of linear tracking model.

In our case, the nominal plant $P_n(z^{-1})$ has a two-timesteps delay. We make $\hat{P}_n(z^{-1})$ free of delay so that $\hat{P}_n^{-1}(z^{-1})$ is realizable. The closed-loop sensitivity function with DOB is:

$$S = \frac{1}{1 + k_2 G_v C_1 + z^{-2}(\frac{G_v}{G_{nv}} - 1)Q}(1 - z^{-2}Q). \tag{7.16}$$

We design $Q$ as a second-order low-pass filter to reject low-frequency disturbances, because $\kappa_{s,ref}$ should have relatively low frequency components for smooth reference trajectory.

When modeling errors exist, we can use the robust control theorem to ensure robust stability [27]. However, modeling uncertainty between the linear model and the nonlinear model is not involved in our problem, which complicates the analysis. One last note is that $P$ is time varying, as $v_x$ is not constant in general. In our implementation, we designed the DOB based on the final steady speed of the vehicle in simulations for simplification. In practice, an adaptive DOB can be designed.

## 7.6 Experiments

### 7.6.1 Training of the RL Policies

In the off-line training phase, the nominal vehicle dynamics is applied, and the environment includes parallel sinusoidal lanes with lane width of 3 meters. Both the policy network and the value network are three-layer fully connected neural networks. We used the proximal policy optimization (PPO) [97] algorithm as the model-free deep RL algorithm. Training techniques such as advantage normalization and reparameterization were applied. The RL training parameters are provided in Table 7.3.

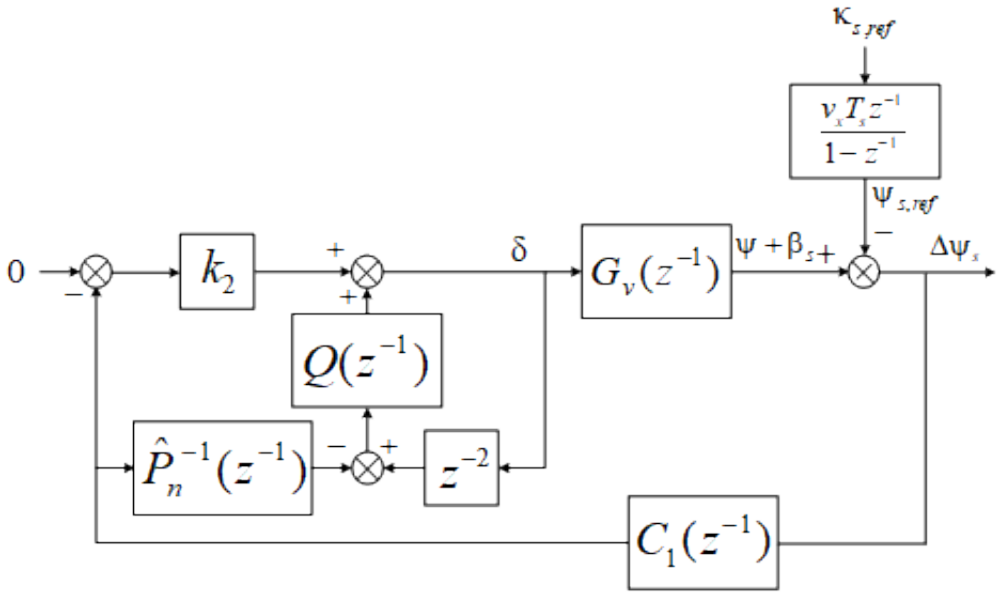


Figure 7.6: Block diagram of closed-loop system.

Table 7.3: RL training parameters.

| | |
|---|---|
| Batch size | 100 |
| PPO clip factor | 0.2 |
| Learning rate | $1 \times 10^{-3}$ |
| Weighting factor $\eta$ | 20 |
| Discount factor $\gamma$ | 0.99 |

In the training phase, the agent achieves successful driving for 1,000 timesteps in around 7,000 iterations, and the policy optimization converges in around 12,000 iterations. After the RL modules are optimized, they can be flexibly assembled to complete different driving tasks using structures shown in Fig. 7.4. Each control module is then cascaded with the source environment to get the transferable trajectory planner. The performance of the hierarchical RL policies for all the three tasks are satisfying. The performance will be presented later in Section 7.6.3 and Section 7.6.4.

### 7.6.2 Analysis of DOB-based Tracking Controller

Given the parameters of the nominal vehicle model, we specified the look-ahead distance as $d_s = 15m$ and designed the controller. Then we analyzed the closed-loop system at

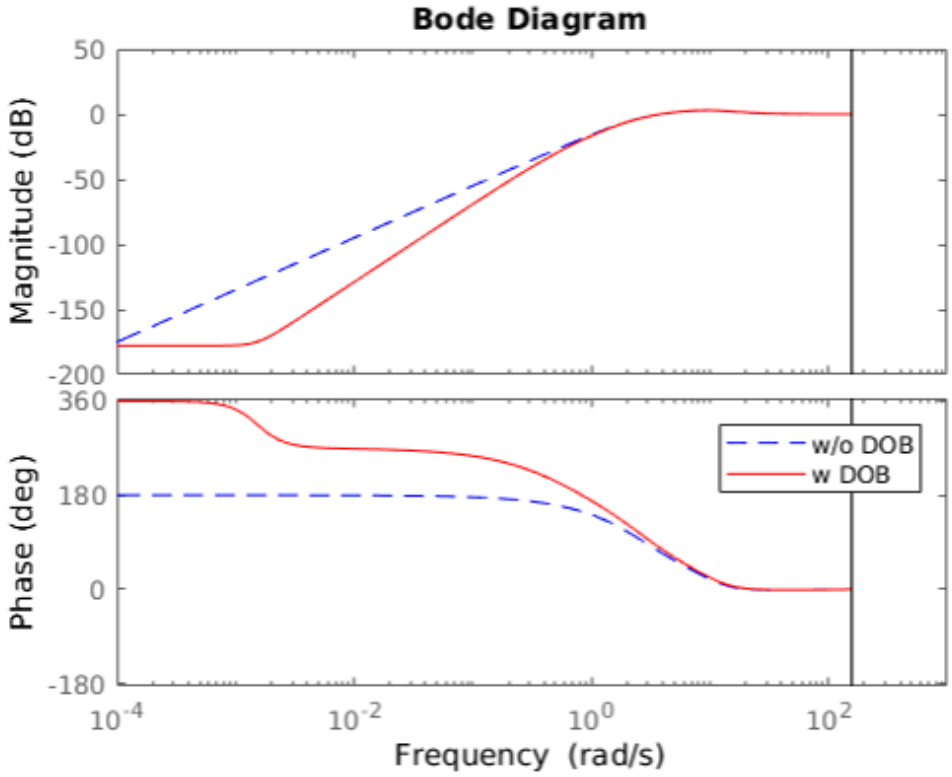


Figure 7.7: Bode plot of sensitivity function.

$v_x = 20m/s$, which is approximately the final steady speed in our simulations. Fig.7.7 shows the bode plots of sensitivity function with and without the designed DOB. Magnitude at low frequency is suppressed due to the low-pass Q-filter, accomplishing our goal of rejecting low-frequency disturbances.

Fig. 7.8 shows the step responses of $\Delta y_s$ and $\Delta\psi_s$ given a step input of previewed reference curvature $\kappa_{s,ref}$ with a magnitude of $10^{-3}m^{-1}$. DOB enables smaller tracking errors for both $\Delta y_s$ and $\Delta\psi_s$. Particularly, the steady state error of $\Delta y_s$ is eliminated by adding DOB.

### 7.6.3 Simulation Performance

In this phase, we evaluated the performance of the proposed RL-RC transfer architecture and compared it with the baseline RL policies. We tested the models in all the three tasks (i.e., LK, LC, and OA). For source domain tests, we ran the baseline policies and their corresponding RL-RC systems on the nominal vehicle settings with random initial conditions (10 times). For target domain tests, both driving strategies were deployed and tested from random initial conditions in 10 target settings with the modeling gap.

We considered two types of modeling gaps. The first was variation in model parameters.

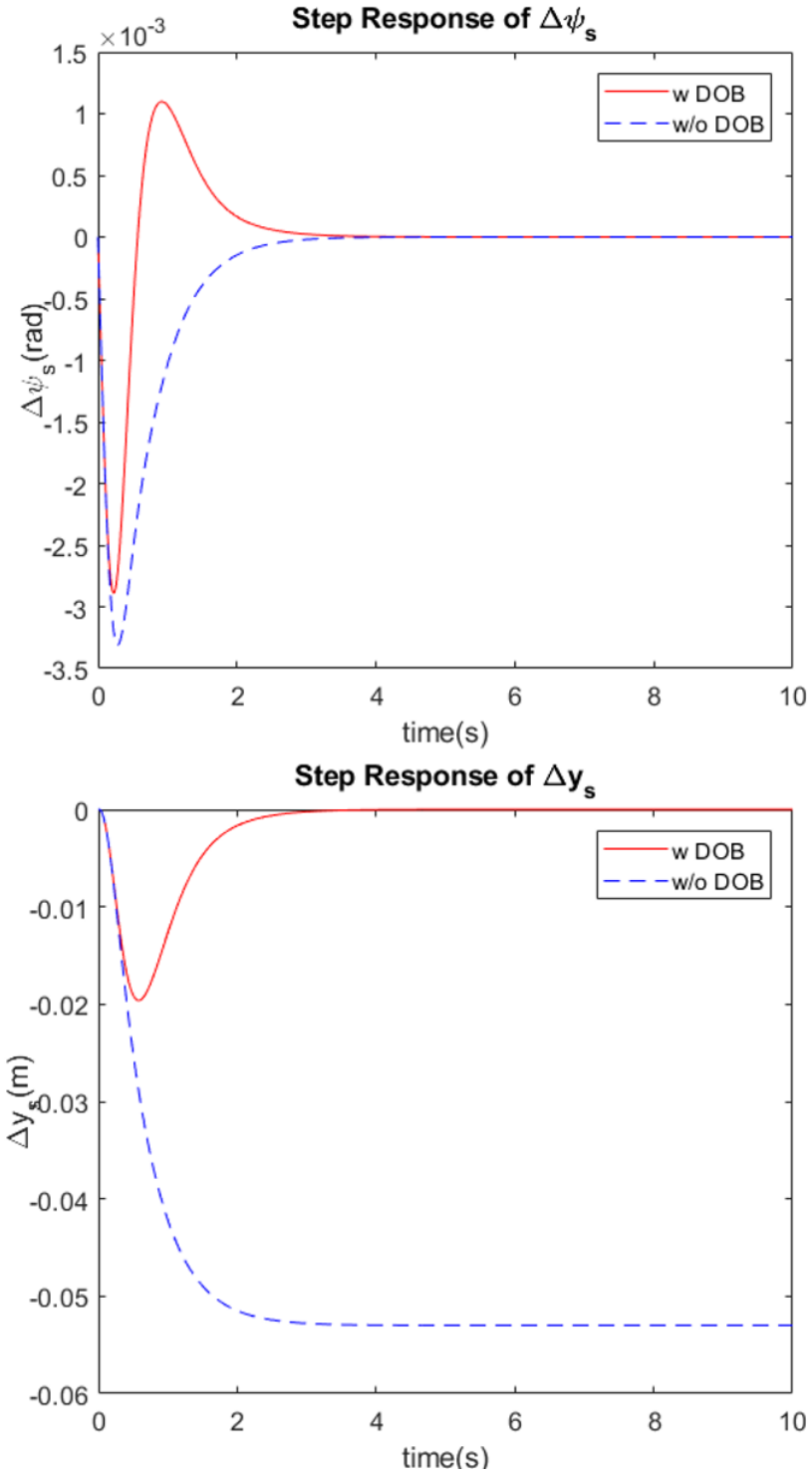


Figure 7.8: Step responses of $\Delta y_s$ and $\Delta\psi_s$.

Table 7.4: Comparison of the performances of baseline RL policies and RL-RC architecture.

| Task | Baseline RL policy | | |
|---|---|---|---|
| | Source | Target - model errors | Target - side force |
| LK | $1000 \pm 0$<br>$49418.0 \pm 270.7$ | $926.3 \pm 177.8$<br>$44644.5 \pm 9538.6$ | $1000 \pm 0$<br>$47102.9 \pm 542.2$ |
| LC | $1000 \pm 0$<br>$47209.7 \pm 213.3$ | $775.8 \pm 239.3$<br>$33990.2 \pm 12007.3$ | $963.0 \pm 111.0$<br>$42578.0 \pm 5800.9$ |
| OA | $1000 \pm 0$<br>$47348.1 \pm 318.4$ | $749.0 \pm 239.1$<br>$33283.7 \pm 11846.1$ | $735.3 \pm 256.5$<br>$31655.7 \pm 12017.1$ |
| Task | RL-RC architecture | | |
| | Source | Target - model errors | Target - side force |
| LK | $1000 \pm 0$<br>$48688.1 \pm 221.3$ | $1000 \pm 0$<br>$48956.0 \pm 362.8$ | $1000 \pm 0$<br>$48959.2 \pm 110.4$ |
| LC | $1000 \pm 0$<br>$47363.2 \pm 303.2$ | $1000 \pm 0$<br>$47150.9 \pm 361.1$ | $1000 \pm 0$<br>$47385.2 \pm 241.4$ |
| OA | $1000 \pm 0$<br>$47661.8 \pm 214.7$ | $1000 \pm 0$<br>$47521.2 \pm 319.1$ | $1000 \pm 0$<br>$47694.8 \pm 374.8$ |

[1] LK stands for lane keeping. LC stands for lane changing. OA stands for obstacle avoidance.

[2] Data is presented in the form of $mean \pm std$. In each cell, episodic length and total reward are listed from top to bottom.

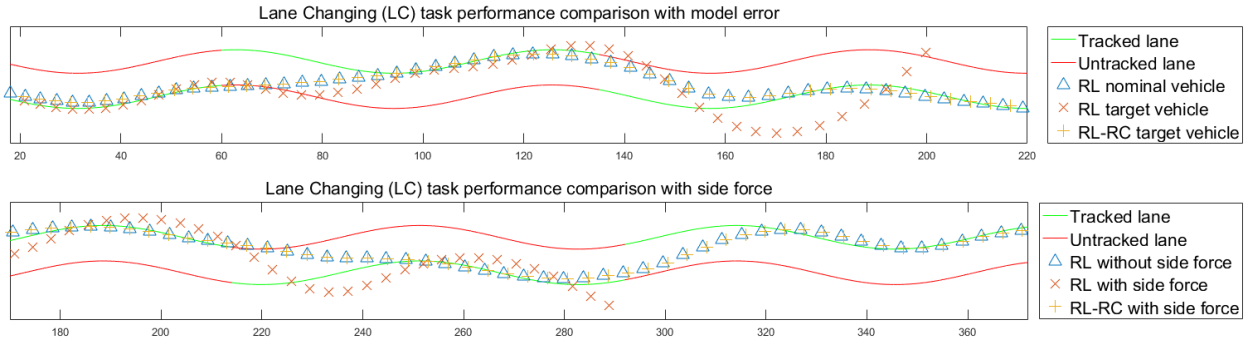


Figure 7.9: Comparison of the driving behaviors for the RL and the RL-RC with modeling gaps in the lane changing (LC) task.

We randomly generated test environments by adding zero-mean and uniformly distributed errors to the parameters of the vehicle model, including vehicle geometry, mass, rotational inertia, tire model parameters, and friction factors. We evaluated the performance on 10 different target vehicles. The second modeling gap was external side-force caused by the side slope of the road or wind. We added a constant external force along the $Y$ axis in the target environments. We conducted 10 tests for each given magnitude of the external force. We recorded the total discounted rewards and episodic lengths for each test episode and calculated their means and standard deviations for each testing category.

Table. 7.4 gives the overall performance comparison of the baseline RL policy and RL-RC

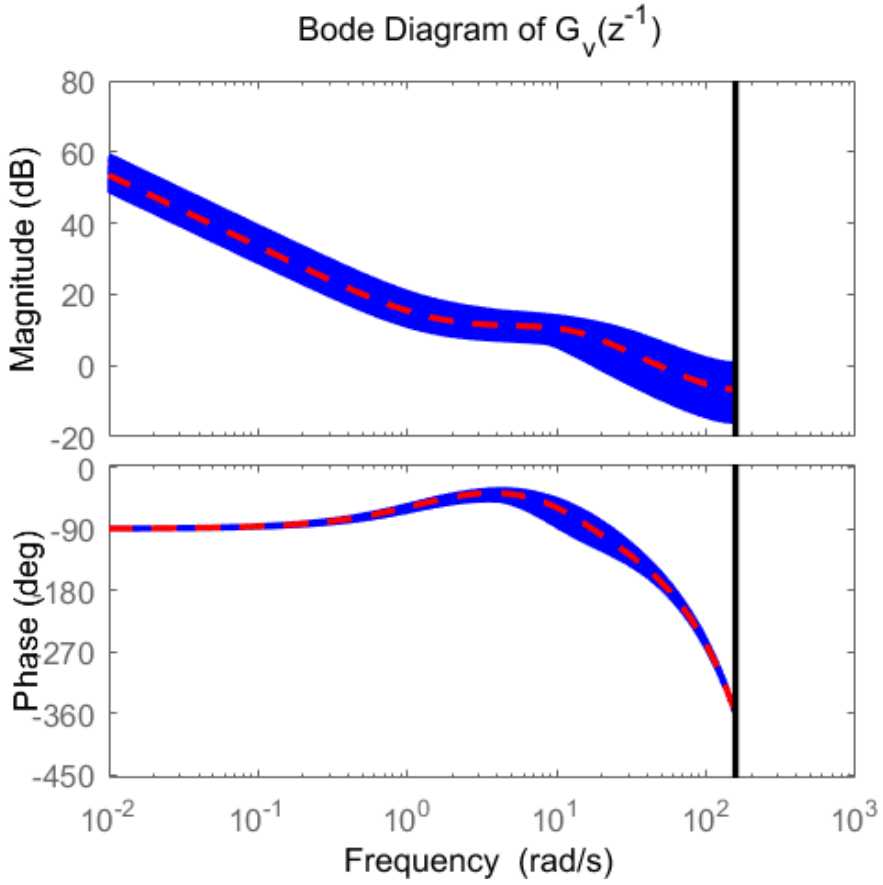


Figure 7.10: Bode plots of $G_v(z^{-1})$ with parameter variation bounded by 20%. The blue curves correspond to 100 samples of $G_v(z^{-1})$ with uniformly distributed parameter variation bounded by 20%. The red dash curve corresponds to the nominal model.

method. Fig. 7.9 shows two examples illustrating the driving behaviors of the two methods. The target settings have parameter variation bounded by 20% of the nominal values or side force of a magnitude of $5,000N$. A demonstration video under the same target settings is also provided online [1]. To give an estimate of the effects of such a modeling gap, Fig. 7.10 shows the bode plots of 100 samples of $G_v(z^{-1})$ with uniformly distributed parameter variation bounded by 20%. The $5,000N$ side force contributes a maximum lateral acceleration of $2.78m/s^2$ to the nominal vehicle.

Table 7.4 and Fig. 7.9 show the effectiveness of the RL-RC in three aspects: (1) The robustly satisfying performance of the deep RL policies in the source setting is validated, which gives us confidence in the transferred trajectory planner to generate reasonable ref-

[1] `https://berkeley.box.com/v/ITSC2018`

erence for the tracking controller; (2) The poor performance of the baseline RL policy in the target settings indicates the incapability of the direct policy transfer. When the baseline policy is applied to a different vehicle or when the target vehicle is affected by a side force, the baseline policy often fails and the target vehicle eventually loses control; (3) The consistent performance of the RL-RC in both the source and the target domain validates the proposed approach can zero-shoot the transfer tasks. This is thanks to the robust and satisfying tracking performance of the DOB-based tracking controller.

Fig. 7.11 further compares the performance of the baseline RL policy and RL-RC system under modeling gaps of different magnitude. The curves show the mean value of the reward, and the error-bar indicates the interval with a maximum deviation of one standard deviation from the mean.

The proposed RL-RC framework obtains consistent reward with parameter variation up to 20% or side force of magnitude up to $5,000N$, whereas the performance of the baseline RL policy keeps getting worse as the modeling gap increases. We see a decrease in reward for all tested disturbance magnitudes.

### 7.6.4 Sim-to-Real Transfer

We deploy our RL framework to a sim-to-real vehicle experiment on our Lincoln MKZ experimental vehicle. The experiment is carried out in the Richmond Field Station of the University of California, Berkeley. There are two experiment settings: one LK setting and one OA setting. In the OA setting, an imaginary (simulated only, for safety reasons) static obstacle vehicle is parked in front of the autonomous vehicle. The autonomous vehicle starts with a speed of $10m/s \approx 22mph$. The autonomous vehicle incorporates the GPS, IMU sensors that can measure its states, while the obstacle vehicle states are assumed known. The experimental vehicle is shown in Fig. 7.12.

The vehicle control pipeline operates in a robot operating system (ROS), as shown in Fig. 7.13. The major ROS nodes include the simulation node that generates the reference trajectory, the DOB node that generates lateral control signals, and the experimental vehicle node that controls the vehicle actuators and collects the onboard sensor data. In the simulation node, the policy and the simulator generate imaginary trajectories of 1.6 seconds in simulation time. The longitudinal control is based on a separate PID controller that is not shown in the figure.

In the experiments, the RL policy publishes the commands at a frequency of 50Hz, the multi-step reference trajectories are generated at a frequency of 5Hz, and the DOB node produces the steering commands at 50Hz. The controller parameters are set based on the nominal dynamics parameters of the experimental vehicle, which are obtained from system identification. We performed both the LK and the OA experiments 10 times with different starting positions, and both received 10/10 success rates. An experiment episode is considered successful if the tracking error is smaller than the threshold and there are no collisions between the ego and the obstacle vehicles. Fig. 7.14 shows a typical OA experimental trajectory including an onboard camera view and the tracking visualization.

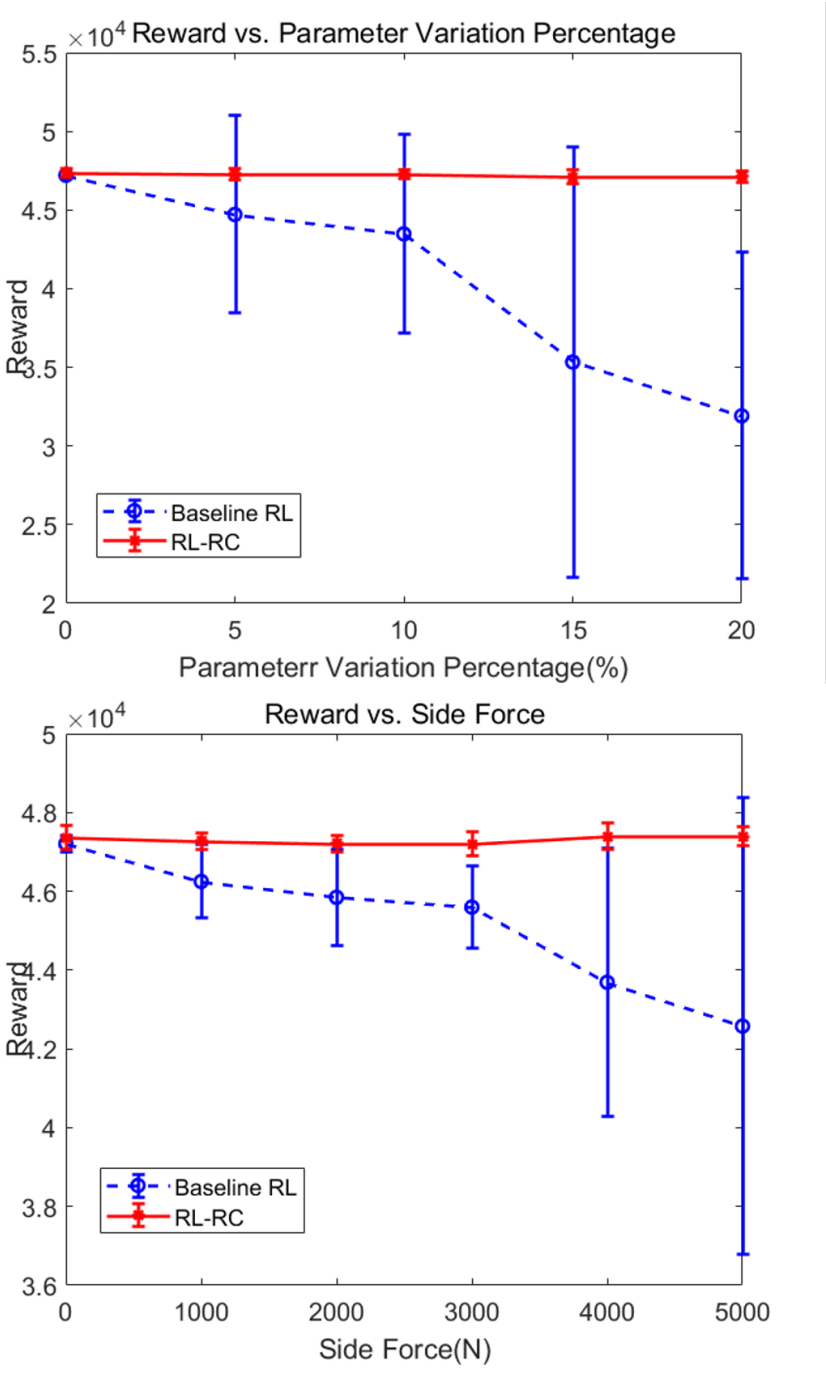


Figure 7.11: Performance (episodic return) of baseline RL policy and RL-RC architecture under increasing modeling gap in the lane changing (LC) task.

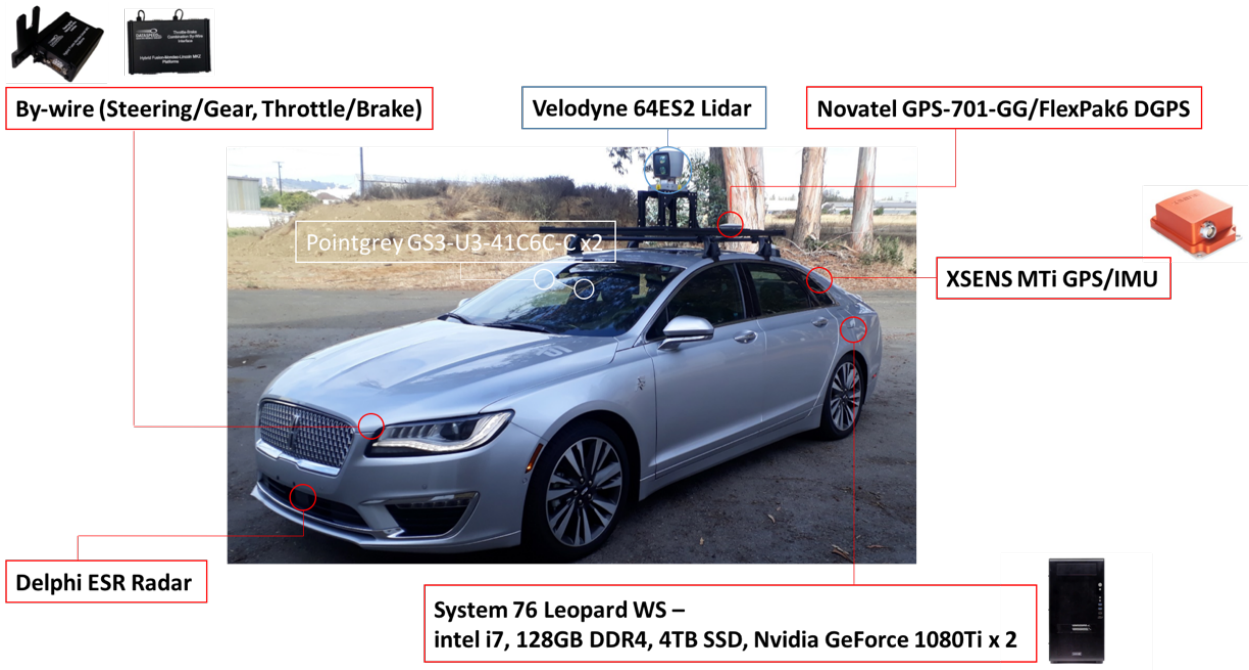


Figure 7.12: The Lincoln MKZ experimental vehicle and the onboard sensors.

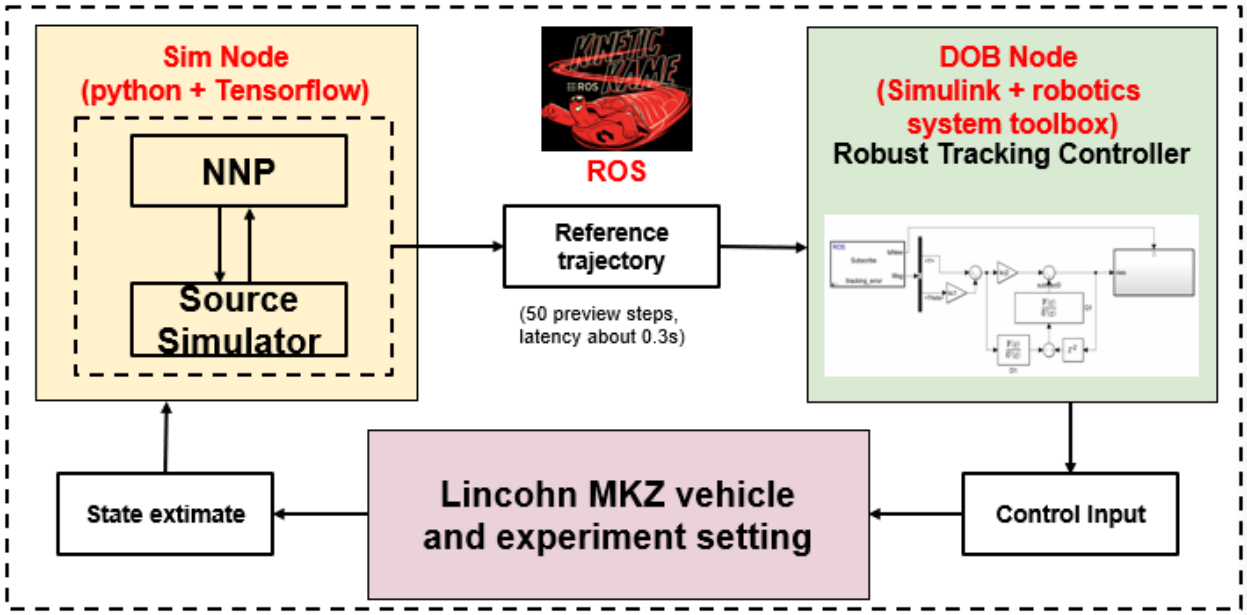


Figure 7.13: The vehicle control pipeline operates in a robot operating system (ROS).

Figure 7.14: A typical OA experimental trajectory including an onboard camera view and the tracking visualization.

## 7.7 Chapter Summary

We proposed a generic framework for driving PN transfer using RC techniques. In the policy transfer scheme, the PNs trained in the source domain are used to generate a finite-horizon previewed reference trajectory of interpretable kinematic features. In the target domain, a robust controller is designed to track the reference trajectory so that disturbances induced by the modeling gap can be rejected. We implemented the framework for three driving tasks in term of a hierarchical RL policy and a DOB-based robust tracking controller. Simulation and experiment results show that the proposed RL-RC architecture can achieve zero-shot consistent performance in the target domain with a certain level of parameter variation or external side force, while the performance of the baseline RL policy is hindered by the modeling gap.

# Chapter 8

# Final Words

## 8.1 Conclusions

While learning-based control (LbC) is one of the most promising solutions to general purpose robotics, it is technically challenging to obtain adaptive or robust LbC policies. This limits the wide application of LbC methodologies. In this dissertation, we explored the methods of representation reasoning for learning-based control, where the representations can reside inside the LbC policies, at the interface of the LbC policies, and outside the LbC policies. A comprehensive investigation of the selection and design of the representations was discussed for specific applications of model-based reinforcement learning (RL), model-free RL, and supervised learning in Chapter 3. We presented the design of a parametrized dynamics context representation inside LbC policies in Chapter 2. In order to transfer learned knowledge encoded in LbC policies, we proposed the attribute learning concept, which decomposes complicated control problems in terms of attributes and learns an attribute network for each attribute separately. Two policy learning frameworks—the cascade attribute networks and the parallel attribute networks—along with the corresponding LbC policy formulations, were presented in Chapter 4 and Chapter 5. Finally, we reasoned for representations outside the LbC policies in Chapter 6 and Chapter 7. Specifically, with kinematic representations bridging the source and target domains, we adopted meta learning (Chapter 6) and robust control (Chapter 7) for domain transfer of LbC policies, rejecting influences from dynamics variation and external disturbances.

## 8.2 Future Work

In order to achieve the goal of intelligent and general-purpose robotics, there are various directions we will explore in the future. These include, but are not limited to, representation reasoning for LbC. For example:

1. To analyze for system stability properties and derive policies with performance guar-

antee.
To get more reliable performance with LbC policies, more theoretical analysis shall be put on the stability of the LBC system. Combined with classic control theory, representations that reflect the reliability of the LbC policies shall be developed. The current works either lack guaranteed performance bounds [118, 46] or decrease the intelligence capabilities [67]. In the future, the combination of both flexible learning and precise control is a potential direction to produce intelligent and reliable policies.

2. To improve the policy learning scheme to narrow down the training-deployment gap.
The majority of the current simulation environments for LbC policy training are already based on high-fidelity physics engines [108, 29]. However, they still differ from the real world in aspects such as vision and dynamics. Recently, more advanced learning-based simulations tools [82] have further improved the simulation accuracy, and training in such environments can further improve the sim-to-real policy transfer capability. Also, new schemes such as improving the LbC policies and the simulation at the same time using evolution algorithms are interesting future research directions. This also requires a better understanding of the target domain, and to leverage the knowledge in the learning phase, so as to learn a policy that achieves satisfying performance in practice. To do so, more knowledge of physics [108, 29], electronics [123, 120], dynamic control systems [118, 105], shall be integrated.

# Bibliography


[1] Martın Abadi et al. "Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[2] Anurag Ajay et al. "Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 3066–3073.

[3] Haitham Bou Ammar et al. "Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015.

[4] Jacob Andreas, Dan Klein, and Sergey Levine. "Modular multitask reinforcement learning with policy sketches". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 166–175.

[5] Karl J Åström. *Introduction to stochastic control theory*. Courier Corporation, 2012.

[6] Karl J Åström and Björn Wittenmark. *Adaptive control*. Courier Corporation, 2013.

[7] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst". In: *arXiv preprint arXiv:1812.03079* (2018).

[8] Andre Barreto et al. "Transfer in deep reinforcement learning using successor features and generalised policy improvement". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 501–510.

[9] André Barreto et al. "Successor features for transfer in reinforcement learning". In: *arXiv preprint arXiv:1606.05312* (2016).

[10] Peter W Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[11] Maria Bauza and Alberto Rodriguez. "A probabilistic data-driven model for planar pushing". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3008–3015.

[12] Samy Bengio et al. "On the optimization of a synaptic learning rule". In: *Preprints Conf. Optimality in Artificial and Biological Neural Networks*. Vol. 2. 1992.

[13] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule*. Citeseer, 1990.

[14] Yoshua Bengio et al. "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 41–48.

[15] Cristian Bodnar et al. "Quantile qt-opt for risk-aware vision-based robotic grasping". In: *arXiv preprint arXiv:1910.02787* (2019).

[16] Joschka Boedecker et al. "Approximate real-time optimal control based on sparse gaussian process models". In: *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE. 2014, pp. 1–8.

[17] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[18] Francesco Borrelli et al. "MPC-based approach to active steering for autonomous vehicle systems". In: *International journal of vehicle autonomous systems* 3.2-4 (2005), pp. 265–291.

[19] Alexander Braylan et al. "Reuse of neural modules for general video game playing". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[20] Arunkumar Byravan and Dieter Fox. "Se3-nets: Learning rigid body motion using deep neural networks". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 173–180.

[21] Haonan Chang, Zhuo Xu, and Masayoshi Tomizuka. "Cascade Attribute Network: Decomposing Reinforcement Learning Control Policies using Hierarchical Neural Networks". In: *arXiv preprint arXiv:2005.04213* (2020).

[22] Yevgen Chebotar et al. "Closing the sim-to-real loop: Adapting simulation randomization with real world experience". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8973–8979.

[23] Jianyu Chen, Zhuo Xu, and Masayoshi Tomizuka. "End-to-end Autonomous Driving Perception with Sequential Latent Representation Learning". In: *arXiv preprint arXiv:2003.12464* (2020).

[24] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. "Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety". In: *arXiv preprint arXiv:1903.00640* (2019).

[25] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. "Model-free deep reinforcement learning for urban autonomous driving". In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2019, pp. 2765–2771.

[26] Wen-Hua Chen et al. "Disturbance-observer-based control and related methods—An overview". In: *IEEE Transactions on Industrial Electronics* 63.2 (2016), pp. 1083–1095.

[27] Xu Chen and Masayoshi Tomizuka. "Control methodologies for precision positioning systems". In: *American Control Conference (ACC), 2013*. IEEE. 2013, pp. 3704–3711.

[28] Paul Christiano et al. "Transfer from simulation to real world through learning deep inverse dynamics model". In: *arXiv preprint arXiv:1610.03518* (2016).

[29] Erwin Coumans and Yunfei Bai. "Pybullet, a python module for physics simulation for games, robotics and machine learning". In: (2016).

[30] Shreyansh Daftry, J Andrew Bagnell, and Martial Hebert. "Learning transferable policies for monocular reactive mav control". In: *International Symposium on Experimental Robotics*. Springer. 2016, pp. 3–11.

[31] Pieter-Tjerk De Boer et al. "A tutorial on the cross-entropy method". In: *Annals of operations research* 134.1 (2005), pp. 19–67.

[32] Marc Deisenroth and Carl E Rasmussen. "PILCO: A model-based and data-efficient approach to policy search". In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. Citeseer. 2011, pp. 465–472.

[33] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[34] Coline Devin et al. "Learning modular neural network policies for multi-task and multi-robot transfer". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 2169–2176.

[35] Alexey Dosovitskiy et al. "CARLA: An open urban driving simulator". In: *Conference on robot learning*. PMLR. 2017, pp. 1–16.

[36] Yan Duan et al. "One-shot imitation learning". In: *arXiv preprint arXiv:1703.07326* (2017).

[37] Yan Duan et al. "$RL^2$: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv preprint arXiv:1611.02779* (2016).

[38] Harrison Edwards and Amos Storkey. "Towards a neural statistician". In: *arXiv preprint arXiv:1606.02185* (2016).

[39] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[40] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *arXiv preprint arXiv:1703.03400* (2017).

[41] Carlos Florensa et al. "Reverse curriculum generation for reinforcement learning". In: *Conference on robot learning*. PMLR. 2017, pp. 482–495.

[42] Justin Fu, Sergey Levine, and Pieter Abbeel. "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors". In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE. 2016, pp. 4019–4026.

[43] Gianni Gilardi and Inna Sharf. "Literature survey of contact dynamics modelling". In: *Mechanism and machine theory* 37.10 (2002), pp. 1213–1239.

[44] Abhishek Gupta et al. "Learning invariant feature spaces to transfer skills with reinforcement learning". In: *arXiv preprint arXiv:1703.02949* (2017).

[45] Tuomas Haarnoja et al. "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (2018).

[46] James Harrison et al. "Adapt: zero-shot adaptive policy transfer for stochastic dynamical systems". In: *arXiv preprint arXiv:1707.04674* (2017).

[47] James Harrison et al. "Adapt: zero-shot adaptive policy transfer for stochastic dynamical systems". In: *Robotics Research*. Springer, 2020, pp. 437–453.

[48] Nicolas Heess et al. "Emergence of locomotion behaviours in rich environments". In: *arXiv preprint arXiv:1707.02286* (2017).

[49] Matteo Hessel et al. "Rainbow: Combining improvements in deep reinforcement learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[50] Daniel Ho et al. *RetinaGAN: An Object-aware Approach to Sim-to-Real Transfer*. 2020. URL: `https://retinagan.github.io`.

[51] Gregory Kahn et al. "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 5129–5136.

[52] Dmitry Kalashnikov et al. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation". In: *arXiv preprint arXiv:1806.10293* (2018).

[53] Alex Kendall et al. "Learning to drive in a day". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8248–8254.

[54] Arne Kesting, Martin Treiber, and Dirk Helbing. "Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 368.1928 (2010), pp. 4585–4605.

[55] Thomas Kipf et al. "Neural relational inference for interacting systems". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2688–2697.

[56] Marek Kopicki et al. "Learning modular and transferable forward models of the motions of push manipulated objects". In: *Autonomous Robots* 41.5 (2017), pp. 1061–1082.

[57] Solomon Kullback and Richard A Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[58] Phil LeBeau. "Waymo starts commercial ride-share service". In: *URL: https://www.cnbc.com/2018/12/05/waymo-starts-commercial-ride-share-service.html* (2018).

[59] Joonho Lee et al. "Learning quadrupedal locomotion over challenging terrain". In: *Science Robotics* 5.47 (2020).

[60] Michelle A Lee et al. "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8943–8950.

[61] Sergey Levine and Pieter Abbeel. "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics." In: *NIPS*. Vol. 27. Citeseer. 2014, pp. 1071–1079.

[62] Sergey Levine and Vladlen Koltun. "Guided policy search". In: *International Conference on Machine Learning*. 2013, pp. 1–9.

[63] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[64] Jesse Levinson et al. "Towards fully autonomous driving: Systems and algorithms". In: *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE. 2011, pp. 163–168.

[65] Jue Kun Li, Wee Sun Lee, and David Hsu. "Push-Net: Deep Planar Pushing for Objects with Unknown Physical Properties." In: *Robotics: Science and Systems*. Vol. 14. 2018, pp. 1–9.

[66] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[67] Changliu Liu and Masayoshi Tomizuka. "Algorithmic safety measures for intelligent industrial co-robots". In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 3095–3102.

[68] Changliu Liu and Masayoshi Tomizuka. "Enabling safe freeway driving for automated vehicles". In: *American Control Conference (ACC), 2016*. IEEE. 2016, pp. 3461–3467.

[69] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: `http://jmlr.org/papers/v9/vandermaaten08a.html`.

[70] Matthew T Mason. "Mechanics and planning of manipulator pushing operations". In: *The International Journal of Robotics Research* 5.3 (1986), pp. 53–71.

[71] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*. 2016, pp. 1928–1937.

[72] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[73] Igor Mordatch et al. "Combining model-based policy search with online model learning for control of physical humanoids". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 242–248.

[74] Urs Muller et al. "Off-road obstacle avoidance through end-to-end learning". In: *Advances in neural information processing systems*. 2006, pp. 739–746.

[75] Tsendsuren Munkhdalai and Hong Yu. "Meta networks". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2554–2563.

[76] Sanmit Narvekar and Peter Stone. "Learning curriculum policies for reinforcement learning". In: *arXiv preprint arXiv:1812.00285* (2018).

[77] Hans B Pacejka and Egbert Bakker. "The magic formula tyre model". In: *Vehicle system dynamics* 21.S1 (1992), pp. 1–18.

[78] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[79] Zherong Pan, Chonhyon Park, and Dinesh Manocha. "Robot motion planning for pouring liquids". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 26. 1. 2016.

[80] Xue Bin Peng et al. "Sim-to-real transfer of robotic control with dynamics randomization". In: *arXiv preprint arXiv:1710.06537* (2017).

[81] Xue Bin Peng et al. "Sim-to-real transfer of robotic control with dynamics randomization". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 1–8.

[82] Tobias Pfaff et al. "Learning Mesh-Based Simulation with Graph Networks". In: *arXiv preprint arXiv:2010.03409* (2020).

[83] Dean A Pomerleau. "Alvinn: An autonomous land vehicle in a neural network". In: *Advances in neural information processing systems*. 1989, pp. 305–313.

[84] Ali Punjani and Pieter Abbeel. "Deep learning helicopter dynamics models". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 3223–3230.

[85] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[86] Aravind Rajeswaran et al. "Epopt: Learning robust neural network policies using model ensembles". In: *arXiv preprint arXiv:1610.01283* (2016).

[87] Kate Rakelly et al. "Efficient off-policy meta-reinforcement learning via probabilistic context variables". In: *International conference on machine learning*. 2019, pp. 5331–5340.

[88] Christian Rathgeber et al. "Lateral trajectory tracking control for autonomous vehicles". In: *Control Conference (ECC), 2014 European*. IEEE. 2014, pp. 1024–1029.

[89] Sachin Ravi and Hugo Larochelle. "Optimization as a model for few-shot learning". In: (2016).

[90] Danilo Rezende et al. "One-shot generalization in deep generative models". In: *International Conference on Machine Learning*. PMLR. 2016, pp. 1521–1529.

[91] Barry Ridge et al. "Self-supervised online learning of basic object push affordances". In: *International Journal of Advanced Robotic Systems* 12.3 (2015), p. 24.

[92] Andrei A Rusu et al. "Progressive neural networks". In: *arXiv preprint arXiv:1606.04671* (2016).

[93] Andrei A Rusu et al. "Sim-to-real robot learning from pixels with progressive nets". In: *Conference on Robot Learning*. PMLR. 2017, pp. 262–270.

[94] Ahmad El Sallab et al. "End-to-End Deep Reinforcement Learning for Lane Keeping Assist". In: *arXiv preprint arXiv:1612.04340* (2016).

[95] Adam Santoro et al. "Meta-learning with memory-augmented neural networks". In: *International conference on machine learning*. PMLR. 2016, pp. 1842–1850.

[96] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[97] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[98] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[99] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[100] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[101] Jochen Stüber, Marek Kopicki, and Claudio Zito. "Feature-based transfer learning for robotic push manipulation". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–5.

[102] Jochen Stüber, Claudio Zito, and Rustam Stolkin. "Let's Push Things Forward: A Survey on Robot Pushing". In: *Frontiers in Robotics and AI* 7 (2020), p. 8.

[103] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998.

[104] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2011.

[105] Chen Tang, Zhuo Xu, and Masayoshi Tomizuka. "Disturbance-Observer-Based Tracking Controller for Neural Network Driving Policy Transfer". In: *IEEE Transactions on Intelligent Transportation Systems* (2019).

[106] Matthew E Taylor and Peter Stone. "Transfer learning for reinforcement learning domains: A survey." In: *Journal of Machine Learning Research* 10.7 (2009).

[107] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[108] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[109] Ricardo Vilalta and Youssef Drissi. "A perspective view and survey of meta-learning". In: *Artificial intelligence review* 18.2 (2002), pp. 77–95.

[110] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[111] Oriol Vinyals et al. "Matching networks for one shot learning". In: *arXiv preprint arXiv:1606.04080* (2016).

[112] Chen Wang et al. "SwingBot: Learning Physical Features from In-hand Tactile Exploration for Dynamic Swing-up Manipulation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2020, pp. 5633–5640.

[113] Peter Wolf et al. "Learning how to drive in a real world simulation with deep q-networks". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 244–250.

[114] Huazhe Xu et al. "End-to-end learning of driving models from large-scale video datasets". In: *arXiv preprint* (2016).

[115] Zhenjia Xu et al. "Densephysnet: Learning dense physical object representations via multi-step dynamic interactions". In: *arXiv preprint arXiv:1906.03853* (2019).

[116] Zhuo Xu, Haonan Chang, and Masayoshi Tomizuka. "Cascade Attribute Learning Network". In: *arXiv preprint arXiv:1711.09142* (2017).

[117] Zhuo Xu, Jiangyu Chen, and Masayoshi Tomizuka. "Guided Policy Search Model-based Reinforcement Learning for Urban Autonomous Driving". In: *arXiv preprint arXiv:2005.03076* (2020).

[118] Zhuo Xu, Chen Tang, and Masayoshi Tomizuka. "Zero-shot deep reinforcement learning driving policy transfer for autonomous vehicles based on robust control". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 2865–2871.

[119] Zhuo Xu and Masayoshi Tomizuka. "History Encoding Representation Design for Human Intention Inference". In: *arXiv preprint arXiv:2106.02222* (2021).

[120] Zhuo Xu, Zhengbao Yang, and Jean Zu. "Impedance matching circuit for synchronous switch harvesting on inductor interface". In: *2015 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE. 2015, pp. 341–345.

[121] Zhuo Xu et al. "COCOI: Contact-aware Online Context Inference for Generalizable Non-planar Pushing". In: *arXiv preprint arXiv:2011.11270* (2020).

[122] Zhuo Xu et al. "Toward Modularization of Neural Network Autonomous Driving Policy Using Parallel Attribute Networks". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 1400–1407.

[123] Zhengbao Yang, Jean Zu, and Zhuo Xu. "Reversible nonlinear energy harvester tuned by tilting and enhanced by nonlinear circuits". In: *IEEE/ASME Transactions on Mechatronics* 21.4 (2016), pp. 2174–2184.

[124] Michael C Yip and David B Camarillo. "Model-less feedback control of continuum manipulators in constrained environments". In: *IEEE Transactions on Robotics* 30.4 (2014), pp. 880–889.

[125] Kuan-Ting Yu et al. "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing". In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2016, pp. 30–37.

[126] Wenhao Yu et al. "Preparing for the unknown: Learning a universal policy with online system identification". In: *arXiv preprint arXiv:1702.02453* (2017).

[127] Tianhao Zhang et al. "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 528–535.

[128] Jiaji Zhou et al. "A convex polynomial force-motion model for planar sliding: Identification and application". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 372–377.

[129] Shaojun Zhu, Andrew Kimmel, and Abdeslam Boularias. "Information-theoretic model identification and policy search using physics engines with application to robotic manipulation". In: *arXiv preprint arXiv:1703.07822* (2017).