Title
Mobile Robot Motion Prediction and Control Under Uncertainties

Author
Gao, Huidong Mona

Publication Date
2022

Peer reviewed|Thesis/dissertation

Mobile Robot Motion Prediction and Control Under Uncertainties

by

Huidong Mona Gao

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 Anil Aswani
Professor Kameshwar Poolla

Fall 2022

Mobile Robot Motion Prediction and Control Under Uncertainties

Abstract

Mobile Robot Motion Prediction and Control Under Uncertainties

by

Huidong Mona Gao

Doctor of Philosophy in Engineering-Mechanical Engineering

University of California, Berkeley

Professor Masayoshi Tomizuka, Chair

Over the past years, wheeled mobile robots have attracted considerable attention in society. They are often used in civilian or military fields, such as agronomy, processing, environmental monitoring, and national defense. However, many uncertainties exist regarding the motion prediction and control of mobile robots. They often exhibit highly non-linear behaviors, which are difficult, or even impossible, to analytically model with sufficient accuracy. Furthermore, many existing prediction and control strategies suffer from parameter uncertainties and external disturbances.

This dissertation research mainly focuses on two specific tasks for mobile robots, namely pushing and trajectory tracking under slippery conditions. These tasks are highly uncertain in nature: First, in terms of pushing dynamics, it is difficult to determine all of the physical parameters (e.g., friction) and motion prediction and control are coupled with the pushed object; second, when tracking mobile robots' trajectory on slippery ground, their behavior cannot be modeled accurately and models are highly dependent on the ground condition.

This dissertation is divided into two parts. Part I focuses on improving the performance of mobile robot pushing. First, we explore the use of a combined prediction model and an online learning framework for planar push prediction. By adjusting low-dimensional analytical parameters in online situations, our robot is able to quickly adapt to new environments. Second, a different approach is discussed, which discards the analytical model all together and uses model-free reinforcement learning to perform robot pushing.

Part II focuses on mobile robot tracking under slippery conditions. The central methodology involves using reinforcement learning at a high level to adjust low-level model parameters, thus achieving robust performance. Specifically, we explore the adaptation of key parameters in a model predictive control optimization formulation, enabling the control scheme to be adjusted to be more aggressive or conservative. Then, online gain tuning in lateral and longitudinal controllers is explored.

To my family

Contents

List of Figures

List of Tables

Acknowledgments

The pursuit of my Ph.D. at Berkeley has been an amazing journey. I have been fortunate to receive a great deal of help and support from my advisor, committee members, internship companies, friends, and family. I cannot express my gratitude enough.

My deep and sincere gratitude goes to my Ph.D. advisor, Professor Masayoshi Tomizuka. His profound knowledge, timely suggestions, encouragement with kindness, and dedication to work have supported me throughout my entire Ph.D. journey. I will definitely bring what I have learned from him to my future endeavors.

I would also like to thank my qualifying exam committee members, namely Professor Kameshwar Poolla, Professor Anil Aswani, Professor Koushil Sreenath, and Professor Laurent El Ghaoui. Furthermore, I wish to again thank Professor Kameshwar Poolla and Professor Anil Aswani for serving as my dissertation committee. Without the knowledge that I acquired from them, I would not have been able to finish this dissertation.

In addition, being a member of the Mechanical System Control (MSC) laboratory over the past few years has been a wonderful experience. I want to especially thank Dr. Zhuo Xu for taking so much time to guide me and provide support for my research projects, and also Dr. Zhaoqi Leng for all of the hands-on help and project ideas. In addition, I wish to sincerely thank my collaborators for all of their inspiring discussions: Dr. Ouyang Yi, Dr. Liting Sun, Dr. Chen Tang, and Dr. Jessica Leu. I also wish to express my thanks to the following MSC lab members for their help during my graduate studies: Changhao Wang, Xinghao Zhu, Yiyang Zhou, Lingfeng Sun, Ge Zhang, Zheng Wu, Jinning Li, Catherine Faulkner, Wu-Te Yang, Xiang Zhang, Ting Xu, Chengfeng Xu, Chenran Li, Akio Kodaira, Ran Tian, Qiyang Qian, Keita Kobashi, Ce Hao, and Wei-Jer Chang.

Moreover, I was fortunate to have internship opportunities at Futurewei Technologies and Meta Platforms. I would like to thank my Futurewei internship collaborators Jiangsheng Yu, Yixin Shen, and Edward Tsien, as well as my Meta internship collaborator Haoyu Fu for his generous support.

Finally, I would like to thank my father Liangyu Gao, my mother Fang Jia, and my brother Endong Gao. You are my role models in life. Thank you all so much for putting your faith in me over the last few years. I would not be where I am today without you.

Chapter 1

Introduction

1.1 An Overview of Mobile Robot Control and Navigation

Ground robots can be divided into fixed-place robots and mobile robots. Fixed-place robots have a fixed base, and hence, their movements are limited by their kinematic structure and links. By contrast, mobile robots are free to move autonomously to perform tasks and achieve set goals. Today, mobile robots are found in numerous different areas, such as agronomy, processing, environmental monitoring, and national defense. They are able to work in cluttered environments, recognize surrounding objects, understand speech, and navigate. Mobile robots already substantially contribute to the welfare of society.

Mobile robot research has been conducted in the fields of perception and localization, prediction, trajectory planning, and trajectory tracking control. The present research mainly focuses on motion prediction and trajectory tracking control. In general, the control procedure of a wheeled mobile robot (WMR) involves two stages, namely kinematic tracking/stabilizing control and dynamic tracking control. The first stage uses a kinematic model of the robot and outputs linear and angular velocities $[v, \omega]$. The second stage uses a Lagrange model or the Newton dynamic model and models low-level dynamics responses to given forces or torques generated by the motor/gear box. The two stages are separately described further in the following subsections.

Mobile Robot Kinematic Modelling

Robot kinematics only deal with a robot's geometric structure, whereas robot dynamics are focused on the analysis of the robot's dynamic behavior, which involves the forces and torques experienced by the robot. Robotic kinematics are essential for describing each joint's position, orientation, and motion. They involve the configuration in the robot's workspace, relations between the robot's parameters, as well as constraints imposed by their built structure and given trajectories. The study of robot kinematics is essential for the study of

dynamics, path planning, stability analysis, and control strategies. Furthermore, mobile robots may have different degrees of freedom as well as different mechanical designs. The major nonholonomic WMRs are as follows: differential drive WMRs, tricycle model WMRs, and car-like steering drive WMRs. In this research, we exclusively focus on the first type, namely differential drive WMRs. The most basic kinematic model for a differential drive mobile robot is the unicycle model, which is described by the following equations:

$$\dot{x_r} = v_r \cdot cos\phi, \quad \dot{y_r} = v_r \cdot sin\phi, \quad \dot{\phi} = \omega_r \tag{1.1}$$

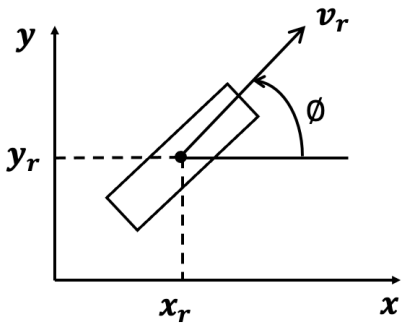


Figure 1.1: Unicycle model.

Mobile Robot Dynamic Modelling

While many researchers prefer to directly use kinematic models in mobile robot control and navigation due to their simplicity, many others aim for a deeper examination of the dynamic forces that impact mobile robots' behaviors. Robot dynamics can be divided into

- direct/forward dynamics, and
- inverse dynamics

Forward dynamics refer to how the robot will act dynamically given forces or torques exerted on the body links. Inverse dynamics provide the required force or torque if certain positions or motions of the robot links are required.

The dynamic model of a mobile robot is derived by using Newton–Euler and Lagrange equations. The most general model for a robot with many links is as follows:

$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}, \;\; \mathbf{q} = [q_1, q_2, ..., q_m]^T \tag{1.2}$$

where, $\mathbf{D}(\mathbf{q})$ is an n by n positive definite matrix for any $\dot{\mathbf{q}} \neq \mathbf{0}$.

In most cases, mobile robot dynamic models are derived from this general Lagrangian model. Notably, many different versions of mobile robot dynamic models exist. They attempt to achieve reliable tracking control by modeling the external forces with complex tire models;

thus, many parameters must be known. Once an effective dynamic model is derived and validated, it assists with tracking control at high loads or speeds, which requires adaptiveness to disturbances from external forces and ground contact.

For WMR prediction and control, many studies use the aforementioned kinematic and dynamic models. Many standard controllers have been studied over the years, such as (i) proportional integral and derivative controllers; (ii) Lyapunov function-based controllers; and (iii) computed torque controllers. There are also advanced controllers, such as (i) state feedback linearization-based controllers and (ii) invariant manifold-based controllers. Moreover, with recent advances in deep learning, data-driven learning methods have also gained popularity in mobile robot control and navigation. Some works use end-to-end learning to output motion commands for robots directly from perceptual inputs. Thus, the system directly bypasses the classical robot navigation hierarchical paradigm, and furthermore, it does not require any human knowledge or engineering design. However many scholars have highlighted that these methods are data-hungry and typically not explainable; thus, debugging might become a concern.

In addition, researchers have explored the combination of deep learning algorithms with classical control methods, such as the use of Q-learning to add correctional inputs to PID commands, or the use of reinforcement learning as an upper module to regulate lower control modules. These approaches use machine learning to alter or aid only in existing modules in traditional control; thus, they are more explainable than end-to-end methods.

In this research, we mainly focus on combining machine learning methods with traditional control/analytical models in prediction and tracking control tasks. The tasks that we consider are robot object pushing and trajectory tracking in slippery conditions, which suffer from parameter uncertainties and external disturbances. Moreover, these tasks are common for mobile robots as they are often required to work in environments where (1) pushing is more efficient than picking up obstacles, and where (2) slippery terrain conditions or rapid cornering are common. We do not consider the use of analytical dynamic models because building one that is accurate for our tasks would be difficult as they involve uncertainties. For the considered tasks, many parameters are unknown and system identification for different scenarios for many parameters would lead to poor generalizability of the model to varied situations.

1.2 Dissertation Contributions and Outline

This dissertation is divided into the following two parts, robot object pushing in Part I and robot trajectory tracking under slippery conditions in Part II. Each part discusses several approaches to the corresponding task. An outline of the two parts and their corresponding dissertation chapters is provided as follows:

Part I: Robot Object Pushing

Pushing is an important motion primitive in a robot's manipulative repertoire, with applications from daily life to extreme environments. Achieving set goals through pushing is challenging for robots and the task has very high uncertainty. This is due to the complexity that arises from unknown frictional forces, nonlinear dynamics, and the fact that the outcome of a push is difficult to predict. Many different approaches have been implemented for pushing under different assumptions and levels of computational power, including the purely analytic, hybrid, simulation, and data-driven methods. Part I of the dissertation contains Chapters 2 and 3, which are outlined as follows:

Chapter 2

In Chapter 2, we focus on 2D planar pushing for mobile robots. Specifically, we seek to predict the effect of pushes (i.e., where the robot and object's position and orientation will be after the push). A thorough literature review revealed that many physical prediction models already exist. However their assumptions usually do not hold in real-world cases. In addition, data-driven machine learning approaches can provide accurate predictions from offline data; however, they usually entail problems related to generalizability.

In this chapter, we introduce a combined prediction model and an online learning framework for planar push prediction. The model takes advantage of both analytical and data-driven models. The combined model has a neural network module and analytical components with a low-dimensional parameter to be learned online. By adjusting this low-dimensional parameter online, the robot is able to quickly adapt to different situations involving the pushing of different objects. We validate our combined model on real robot pushing experiments, and the experimental results demonstrate our model's adaptive capabilities.

Chapter 3

In Chapter 3, we continue to investigate mobile robot pushing in a 3D environment. Many existing works in mobile robot pushing either use analytical models to calculate pushing outcomes or model-based data-driven methods to learn a model to plan trajectories. However, pushing is generally difficult to model accurately due to frequent changes in contact modes and perturbations from the surrounding environment. Therefore, many model-free reinforcement learning approaches have been proposed to solve the pushing problem. Chapter 3 explores a novel soft-actor-critic (SAC)-based model-free framework to output control policies for pushing. By processing raw images to a robot-centric multicolored mask and adding LSTM modules into the critic network, the entire framework incorporates pushing dynamic learning and possesses high model generalizability.

Furthermore, with our framework, we provide a comparison in terms of exploration efficiency and pushing strategy for direct and hierarchical control policies. A direct control policy refers to the end-to-end policy that directly outputs desired wheel velocity commands to the robot. A hierarchical control policy, by contrast, outputs the robot's next time-step

positional command. Then, a low-level controller executes this high-level positional command by giving a series of corresponding wheel velocity commands. Finally, we test the generalizability for dealing with the novel objects of these two control policies with our framework.

Part II: Robot Trajectory Tracking Under Slippery Conditions

Today, WMRs have an increasing number of applications both indoors and outdoors. They are often required to function in different environments, such as on rough terrain, icy or wet roads, and routes with rapid cornering. Therefore, it is crucial to design a control scheme that delivers robust tracking performance on various terrains, on especially those that induce slipping easily. However, trajectory control on slippery terrain poses numerous uncertainties. Moreover, an accurate model is difficult to obtain, and although some works have focused on identifying terrain parameters within the model, they have heavily relied on test-time data and are not particularly robust. In Part II, we discuss two approaches that are aimed at improving the performance and robustness of mobile robot trajectory tracking under slipping.

Chapter 4

In this chapter, we focus on the challenge of mobile robot motion under slippery conditions and propose a hierarchical framework. Said framework learns to adapt the control parameters online for the robust model predictive tracking control of mobile robots. Concretely, the high-level module is based on reinforcement learning (RL) and actively adjusts the model predictive control (MPC) scheme to be more aggressive or conservative through adapting key parameters in the MPC optimization formulation. Our experiments demonstrate that this framework achieves adaptive and robust tracking performance, especially in rejecting slipping and reducing tracking errors when the mobile robot travels through various terrains. Our framework does not need to know specific dynamic parameters, nor does it require data fitting. The model trained in simulation is validated in a zero-shot manner with completely different real-world terrain conditions, thus demonstrating its adaptability.

Chapter 5

In Chapter 5, we follow a similar approach to that in Chapter 4, exploring with a hierarchical control structure. Instead of using a model predictive control scheme, we divide the control part into longitudinal and lateral control modules and propose the use of reinforcement learning to tune parameters in the modules simultaneously. Tuning gains directly within controllers provide a more straightforward approach for influencing with lateral and speed tracking errors.

Our experiments demonstrate the necessity of performing simultaneous gain tuning of our framework. We also demonstrate that our online framework outperforms the best base-

line model using fixed gains. By using the online gain adaptation scheme, our framework achieves robust tracking performance in terms of long- and short-term average and max lateral tracking errors, which are the most critical aspects when considering tracking under slipping.

Part I

Mobile Robot Object Pushing Task

Chapter 2

Pushing Prediction For Mobile Robots

2.1 Introduction

Pushing is a useful robotic capability for the positioning and reorientation of objects. Consider, for example, a household robot attempting to move a box to another room. Instead of picking up every object that obstructs the path to create space, the robot could simply push the box to the goal. Pushing larger obstacles out of the way is even more essential for mobile robots working in extreme environments, such in mines or on the moon. Furthermore, the robot may not be equipped with grippers to save costs or space, in which case pushing becomes an essential ability for the robot to achieve its goals. Figure 2.1 presents example scenarios where mobile robots are required to push objects.

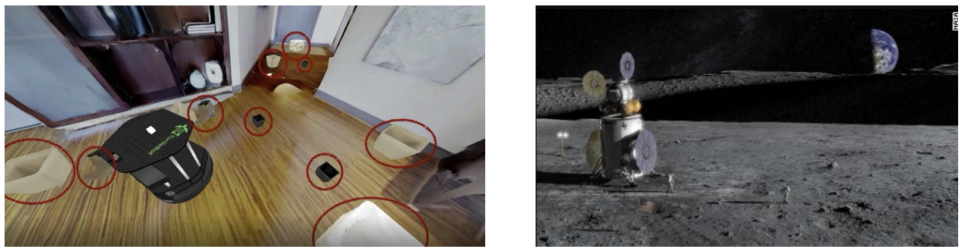


Figure 2.1: Example scenarios of pushing by mobile robots.

Due to increasing interest in the study of mobile robot pushing, deriving accurate pushing prediction models is crucial. Traditionally, robot pushing models are constructed from analytical methods using physical laws. However, these models usually fail to provide accurate predictions due to their strong assumptions and requirements for certain physical states and parameters to be known. In recent years, data-driven approaches have been growing in popu-

larity for building models that predict complex physical dynamics. However, although these models can perform amazingly accurate predictions, they often struggle when transferred to new and unseen situations online.

In this chapter, we investigate the online adaptation abilities of push prediction models under new or changing environments. The task that we consider is planar pushing. Although pushing is a primitive manipulation action, the dynamics are highly non-linear and involve multiple factors such as geometry and mass distribution, even in the simplest case of a linear planar push. Well established analytical models for planar pushing require information about object properties and frictional parameters,[1, 2] and moreover, their assumptions on frictional forces may not hold. Recent studies on the use of data-driven models for planar pushing have improved the push prediction accuracy over analytical models for offline data; however, they have not been designed for online adaptation [3, 6, 7, 8, 9, 10, 11, 12, 13, 4, 5].

To achieve quick online adaptation, we propose a combined prediction model and an online learning framework for planar push prediction [14]. The combined model consists of a neural network module and several analytical components. The neural network helps to improve the expressiveness of the prediction model, while the low-dimensional parameter in analytical components allows rapid online adjustment. First, we train the high-dimensional neural network parameter offline using a pre-collected dataset. Then we iteratively learn the low-dimensional analytical parameters online from data collected in the actual push trajectory. This online learning framework allows us to take advantage of the accuracy of data-driven offline training as well as the rapid online adaptation under new or changing conditions. We test our combined prediction model and online learning framework in the following two sets of experiments: (1) experiments generated from the MIT Push Dataset[15], and (2) real pushing data collected by a modified TurtleBot3 [16] pushing regular packing boxes. The experimental results indicate that our combined prediction model is able to adapt quickly to unseen situations, and it achieves a final prediction performance similar to offline training loss.

Our main contributions in this chapter are as follows: (1) We provide a novel combined push prediction model that consists of both analytical and data-driven components. (2) We propose a simple and effective online learning framework that can adapt quickly to online situations by adjusting a low-dimensional parameter. Moreover, this adjustment only requires very few data points. (3) We verify the capability of our learning framework with two sets of real robot pushing experiments.

The remainder of this chapter is organized as follows: Section 2.2 provides a brief overview of works related to mobile robot pushing models; Section 2.3 provides some preliminaries for building our model; Section 2.4 provides a detailed explanation of the proposed approach; Section 2.5 demonstrates the experiments conducted in two different scenarios and then provides a detailed discussion; and lastly, Section 2.6 concludes the chapter.

2.2 Related Works

Data-driven approaches are popular in pushing prediction, and multiple learning methods have been proposed to train different prediction models. Gaussian approximation is a basic learning tool which has been used to predict final object poses from initial pushing orientations [3]. Regular regression and density estimation methods have also been used to train push prediction models [6] which outperform analytical models. To further improve predictions on planar pushing, other types of models have been considered in the literature, including physics-based force-motion models [7], local Markov decision process (MDP) models [8], and heteroscedastic Gaussian process models [9].

More recently, advances in deep learning have drawn attention to using neural network models for physical interactions. SE3-Nets [12] use deep neural networks to directly predict rigid body motions from point cloud data. Image prediction based deep learning models [13, 4, 5] also show success in several manipulation tasks including planar pushing. These deep learning models generally improve the prediction accuracy on specific distributions of the given dataset, but they often have generalizability issues for unseen data distributions.

One approach to improve the transferability of neural networks is to use hybrid architectures that combine analytical and data-driven models [10, 11]. This approach benefits from the expressiveness of data-driven models and the generalizability of analytical methods. Our combined push prediction model is inspired by the generalization abilities of these hybrid models. We further push beyond generalization to online adaptation and design models that take advantages of both offline and online training.

There are other deep learning based online adaptation methods using either recurrent neural networks [17] or meta learning [18]. Although these methods seem appealing, they require complex training procedures which often can only be done in simulators. Works that focus on improving model transferability also exist [19], but they require point cloud as inputs in order to sample 1,000 surface features to feed into the model. In comparison, our combined model and online learning method provide a simple though effective online adaptation scheme. For a more complete overview on robot pushing, please refer to the survey [20].

2.3 Preliminaries

Planar Push Prediction

We consider the prediction problem of the pushing behavior between a robot and an object on a surface. At each time t, we observe the (center) position $\mathbf{p}_o(t) = (\mathbf{p}_{o,x}(t), \mathbf{p}_{o,y}(t)) \in \mathbb{R}^2$ and orientation $\omega_o(t) \in [0, 2\pi)$ of the object, the position $\mathbf{p}_r(t) = (\mathbf{p}_{r,x}(t), \mathbf{p}_{r,y}(t)) \in \mathbb{R}^2$ of the robot, and the robot's motion command $\mathbf{u}_r(t) = (\mathbf{u}_{r,x}(t), \mathbf{u}_{r,y}(t)) \in \mathbb{R}^2$. The robot will then move according to $\mathbf{u}_r(t)$ to $\mathbf{p}_r(t+1) = \mathbf{p}_r(t) + \mathbf{u}_r(t)$ and push the object along its

way. Given the observation, our goal is to predict the object's next position $\mathbf{p}_o(t+1)$ and orientation $\omega_o(t+1)$ at time $t+1$ after pushed by the robot.

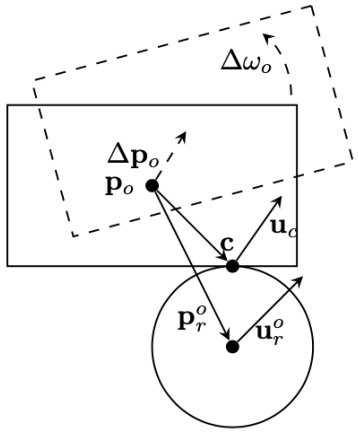


Figure 2.2: Planar Pushing.

Since the goal is to predict the object's movement, we transform the observations and predictions into the object's current frame. Specifically, let $\mathbf{p}_r^o(t)$ and $\mathbf{u}_r^o(t)$ denote the relative position and motion of the robot, and $\Delta\mathbf{p}_o(t)$ and $\Delta\omega_o(t)$ be the relative change in position and orientation of the object. They satisfy the following equations:

$$\mathbf{p}_r^o(t) = \mathbf{R}_{(-\omega_o(t))}(\mathbf{p}_r(t) - \mathbf{p}_o(t)) \tag{2.1}$$
$$\mathbf{u}_r^o(t) = \mathbf{R}_{(-\omega_o(t))}\mathbf{u}_r(t) \tag{2.2}$$
$$\Delta\mathbf{p}_o(t) = \mathbf{R}_{(-\omega_o(t))}(\mathbf{p}_o(t+1) - \mathbf{p}_o(t)) \tag{2.3}$$
$$\Delta\omega_o(t) = \omega_o(t+1) - \omega_o(t) \tag{2.4}$$

where $\mathbf{R}_\omega$ denotes the rotation matrix by the angle ω.

Figure 2.2 illustrates the planar pushing interaction between the robot and the object. For notation simplicity, we will drop the time index t if it's clear in the context in the rest of the paper. Then the goal of push prediction is to find a prediction function f_θ, parameterized by θ, that predicts $f_\theta(\mathbf{p}_r^o, \mathbf{u}_r^o) = (\hat{\Delta}\mathbf{p}_o, \hat{\Delta}\omega_o)$.

Performance Metric

To evaluate the prediction performance, we use the standard metric of normalized mean square error (NMSE) for both positional and rotational losses. The same metric was also used in data-driven pushing models [9].

To compute NMSE, we define positional loss functions $\ell_{\text{pos},x}$, $\ell_{\text{pos},y}$, and a rotational loss function ℓ_{rot} by

$$\ell_{\text{pos},i}(\hat{\Delta}\mathbf{p}_o, \Delta\mathbf{p}_o) = \frac{1}{\sigma^2_{\Delta\mathbf{p}_o}}(\hat{\Delta}\mathbf{p}_{o,i} - \Delta\mathbf{p}_{o,i})^2,\ i = x, y \tag{2.5}$$

$$\ell_{\text{rot}}(\hat{\Delta}\omega_o, \Delta\omega_o) = \frac{1}{\sigma^2_{\Delta\omega_o}}(\hat{\Delta}\omega_o - \Delta\omega_o)^2 \tag{2.6}$$

where $\sigma_{\Delta\mathbf{p}_o}$ and $\sigma_{\Delta\omega_o}$ are the standard deviations for $\Delta\mathbf{p}_o$ and $\Delta\omega_o$. Positional and rotational NMSEs are given by

$$\text{NMSE}_{\text{pos}} = \frac{1}{2}\mathbb{E}\big[\ell_{\text{pos},x}(\hat{\Delta}\mathbf{p}_o, \Delta\mathbf{p}_o) + \ell_{\text{pos},y}(\hat{\Delta}\mathbf{p}_o, \Delta\mathbf{p}_o)\big] \tag{2.7}$$

$$\text{NMSE}_{\text{rot}} = \mathbb{E}\big[\ell_{\text{rot}}(\hat{\Delta}\omega_o, \Delta\omega_o)\big] \tag{2.8}$$

We also define the overall loss function ℓ as

$$\begin{aligned}\ell((\hat{\Delta}\mathbf{p}_o, \hat{\Delta}\omega_o),\ (\Delta\mathbf{p}_o, \Delta\omega_o)) &= \ell_{\text{pos},x}(\hat{\Delta}\mathbf{p}_o, \Delta\mathbf{p}_o) \\ &+ \ell_{\text{pos},y}(\hat{\Delta}\mathbf{p}_o, \Delta\mathbf{p}_o) + \ell_{\text{rot}}(\hat{\Delta}\omega_o, \Delta\omega_o)\end{aligned} \tag{2.9}$$

Physical Model

Suppose the center of mass (COM) of the object is at the object center such that $\mathbf{p}_o = \text{COM} = (\text{COM}_x, \text{COM}_y)$. Let $\mathbf{c} = (\mathbf{c}_x, \mathbf{c}_y)$ be the pushing contact point in the object's frame, and $\mathbf{u}_c = (\mathbf{u}_{c,x}, \mathbf{u}_{c,y})$ is the motion (in the object's frame) of the contact point being pushed by the robot. When $\mathbf{c}$, $\mathbf{u}_c$ and a friction-related parameter h are available, the physical model of pushing dynamics [2] gives

$$\Delta\text{COM}_x = \frac{(h^2 + \mathbf{c}_x^2)\mathbf{u}_{c,x} + \mathbf{c}_x\mathbf{c}_y\mathbf{u}_{c,y}}{h^2 + \mathbf{c}_x^2 + \mathbf{c}_y^2} \tag{2.10}$$

$$\Delta\text{COM}_y = \frac{(h^2 + \mathbf{c}_y^2)\mathbf{u}_{c,y} + \mathbf{c}_x\mathbf{c}_y\mathbf{u}_{c,x}}{h^2 + \mathbf{c}_x^2 + \mathbf{c}_y^2} \tag{2.11}$$

$$\Delta\omega_o = \frac{\mathbf{c}_x\Delta\text{COM}_y - \mathbf{c}_y\Delta\text{COM}_x}{h^2} \tag{2.12}$$

We use F_{physical} to denote this physical model (2.10)-(2.12) as $F_{\text{physical}}(\mathbf{c}, \mathbf{u}_c) = (\Delta\text{COM}, \Delta\omega_o)$. One may attempt to directly use this model as a prediction function, but the physical model is subjected to several limitations: (1) COM of the object may not be at its center $\mathbf{p}_o$. For example, when the object is a box containing items with different weights, its COM is usually different from its geometric center. (2) It's difficult to determine the exact contact point $\mathbf{c}$ between the robot and the object from their positions. Moreover, a push can be either sticking or slipping. For a sticking push we can simply get $\mathbf{u}_c = \mathbf{u}_r$. However, in the slipping case, $\mathbf{u}_c$ will depend on the complex friction interaction between the robot and the object.

(3) The physical model requires the knowledge of a friction-related parameter h, but this parameter varies for different contact surfaces and is usually unknown for unseen objects.

Therefore, directly applying the physical model may result in inaccurate predictions due to above issues.

2.4 Learning Method and Prediction Model

Online Learning for Planar Push Prediction

When a pushing dataset is available, data-driven approaches can learn prediction functions offline. Offline trained functions may perform well for situations similar to the collected dataset, but they usually have generalizability issues with unseen cases.

In planar pushing, many important factors can vary in different scenarios. For example, different objects have different weights, friction coefficients, and different COM positions. Furthermore, these pushing-related factors are often not available to the robot before it actually pushes the object. In most cases, the only way to infer these properties is to observe the online pushing results. Therefore, online learning is essential to perform accurate push predictions.

Taking advantages of both online and offline training, we consider an online learning setting with offline pre-training. In particular, we split the prediction function parameter into $\theta = (\theta_{\text{offline}}, \theta_{\text{online}})$. The offline component θ_{offline} is trained offline with a pre-collected dataset while the online component θ_{online} will be learned online to adapt to new pushing trajectories. The idea is that the offline component θ_{offline} can be high-dimensional to improve the expressiveness of the prediction model. On the other hand, the online component θ_{online} can be designed to be low-dimensional for fast online adaptation.

Suppose we have a dataset D consisting of data points of the form (x, y) where $x = (\mathbf{p}_r^o, \mathbf{u}^o)$ and $y = (\Delta\mathbf{p}_o, \Delta\omega_o)$. Then the goal of the offline pre-training phase is to find optimal offline parameter

$$\theta^*_{\text{offline}} = \theta_{\text{offline}} \frac{1}{|D|} \sum_{(x,y)\in D} \ell(f_{(\theta_{\text{offline}}, \theta_{\text{online}}(0))}(x), y) \tag{2.13}$$

Note that we fix the online parameter to an initial value $\theta_{\text{online}}(0)$ in offline training. An optional scheme is to also train the online parameter θ_{online} offline, but this may require additional hyperparameter tuning.

When it comes to the online situation, we have a pre-trained prediction function $f_{(\theta^*_{\text{offline}}, \theta_{\text{online}}(0))}$ at time 0. The main idea of online learning is to adjust the online parameter $\theta_{\text{online}}(t)$ adapting to the pushing outcomes at each time t.

Let $x(t) = (\mathbf{p}_r^o(t), \mathbf{u}^o(t))$ and $y(t) = (\Delta\mathbf{p}_o(t), \Delta\omega_o(t))$ be the pushing outcome to be predicted. Then the push prediction at this time is $f_{(\theta^*_{\text{offline}}, \theta_{\text{online}})}(x(t))$ with a prediction loss

$$\ell(f_{(\theta^*_{\text{offline}}, \theta_{\text{online}}(t))}(x(t)), y(t)). \tag{2.14}$$

After the pushing outcome $y(t)$ is observed at time $t+1$, we can update the online parameter by

$$\theta_{\text{online}}(t) =_{\theta_{\text{online}}} \ell(f_{(\theta^*_{\text{offline}},\theta_{\text{online}})}(x(t)), y(t)) \tag{2.15}$$

The overall online learning algorithm is described below.

Algorithm 1 Online Learning for Push Prediction

Inputs:
Dataset D of pushing trajectories
Prediction model f_θ
Initial parameter θ_1
Loss function $NMSE$
Train NN parameter θ_2
for $t = 0, 1, 2, \ldots$ **do**
Observe $x(t)$ and $y(t)$ online
Update low-dimensional parameter $\theta_1(t+1)$
end for

To apply this online learning framework, we need to have a prediction model $f_{(\theta_{\text{offline}},\theta_{\text{online}})}$ with a high-dimensional offline parameter θ_{offline} and a low-dimensional online parameter θ_{online}. To do this, we will make use of analytical models as low-dimensional online components, and a data-driven model for the high-dimensional offline component. These components will be constructed in the next subsections.

Center of Mass Corrections

One challenge in push prediction is to know the center of mass (COM) of the object. The object center $\mathbf{p}_o$ can be an approximation of COM, but it usually has an offset from COM in many situations. Therefore, we propose an analytical procedure to correct COM as shown in Figure 2.3.

Consider $\omega_o = 0$ without loss of generality. Let $\mathbf{v} = \mathbf{p}_o - \text{COM} \in \mathbb{R}^2$ be the offset vector from the true COM to the object center in the object's frame. Then the robot's relative position to the object should be corrected to its relative position to the COM by

$$\mathbf{p}_r^{o,\text{corrected}} = \mathbf{p}_r - \text{COM} = \mathbf{p}_r^o + \mathbf{v} \tag{2.16}$$

For the push prediction, note that the offset between the object center and COM after pushing is the rotated vector $\mathbf{R}_{\Delta\omega_o}\mathbf{v}$. Suppose the COM motion is ΔCOM after being pushed, then the motion of the object center is given by

$$\begin{aligned}\Delta\mathbf{p}_o =& \mathbf{p}_o(t+1) - \mathbf{p}_o(t) \\ =& \text{COM}(t+1) + R_{\Delta\omega_o}\mathbf{v} - \mathbf{p}_o(t) \\ =& \Delta\text{COM} + (R_{\Delta\omega_o} - \mathbf{I})\mathbf{v}\end{aligned} \tag{2.17}$$

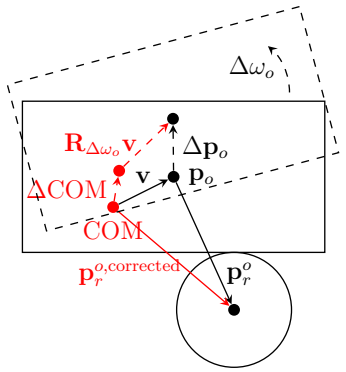


Figure 2.3: Center of Mass Corrections.

Contact Point Prediction for Physical Model

Since we want to apply the physical model F_{physical}, we need to predict the contact point $\mathbf{c}$ and the contact motion $\mathbf{u}_c$. As discussed earlier, predicting the contact point and motion is one of the key challenges in analyzing pushing behaviors. Therefore, we take advantage of data-driven ideas to train a model for the complex contact interactions. We achieve this by using a feedforward neural network $\phi_{\theta_{\text{offline}}}$ parameterized by an offline parameter θ_{offline}. In particular, this neural network will take the corrected relative robot position and motion as input, and output the predicted contact point and motion $\phi_{\theta_{\text{offline}}}(\mathbf{p}_r^{o,\text{corrected}}, \mathbf{u}_r^o) = (\mathbf{c}, \mathbf{u}_c)$.

Combined Push Prediction Model

As shown in Figure 2.4, putting together the neural network $\phi_{\theta_{\text{offline}}}$, physical model F_{physical}, and COM corrections (2.16)-(2.17) we get the combined push prediction model:

$$f_{(\theta_{\text{offline}},\theta_{\text{online}})}(\mathbf{p}_r^o, \mathbf{u}_r^o) = (\Delta\mathbf{p}_o, \Delta\omega_o) \tag{2.18}$$

In the prediction model, we have a high-dimensional offline parameter θ_{offline} from the neural network which can improve the expressive power of the model. Form the analytical components F_{physical} and COM corrections (2.16)-(2.17), we have a low-dimensional online parameter $\theta_{\text{online}} = (\mathbf{v}, h) \in \mathbb{R}^3$. Since the online parameter θ_{online} has a low dimension, it can be quickly trained to adapt to online data.

One important features of the model is that all the components are differentiable. This allows us to perform both offline and online training in an end-to-end manner. Note that it's commonly observed in deep models that end-to-end training can further exploit the expressive power by letting the network to determine its own state representation. This also implies that each intermediate variable may be trained to behave differently than its analytical role.

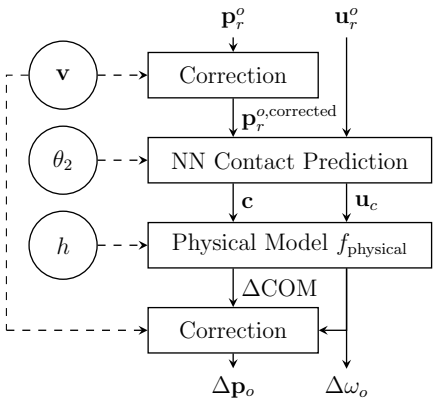


Figure 2.4: Combined Push Prediction Model.
$f_{(\theta_{\text{offline}},\theta_{\text{online}})}$, $\theta_{\text{online}} = (\mathbf{v}, h) \in \mathbb{R}^3$.

2.5 Experiments

To test our online learning algorithm and the combined push prediction model, we consider two different sets of experiments. The first set of experiments are based on data from the MIT push dataset [15]. The MIT dataset is collected by a robot arm with a stiff cylindrical steel pusher (9.5 mm diameter) pushing a variety of small objects (about 100 mm wide) on four different surfaces. To simulate the online situation, we create a simulator that outputs one data point at a time from a complete trajectory. The model is then updated online with the simulated pushing trajectory.

In the second set of experiments, we collected real data using a TurtleBot3 [16]. Since our focus is on predicting positional and rotational changes by planar point pushing, we modify the shape of the robot to a disc by covering it with a round basin (radius 350 mm). The experiments include pushing boxes of two different sizes, different COM positions, and with different contact surfaces with the floor.

The two sets of experiments vary in data collection frequency, pusher size and object type, in order to show our method works under a variety of settings. We consider only rectangle objects in both experiments, as it is the most common shape in real world scenarios. Though in principle our method can also be applied to other object shapes when the training and testing objects have the same shape.

Since our goal is to demonstrate the online learning ability of the combined push prediction model, in all the experiments we choose a simple neural network architecture consisting of three fully connected layers, with each layer having 16 units followed by the ReLU activation. The entire combined model, including the analytical components and the neural network, is implemented in the deep learning framework Chainer [21]. Before model training, each of the input and output variable to the prediction model is normalized to zero mean and

unit variance. In the offline training phase, we use the Adam optimizer [22] with learning rate 0.005 and batch size 32. In online learning, at every time step the online optimization problem (2.15) is solved by gradient descent with 5 gradient steps with learning rate 0.005. The initial v is set to zero because there is no COM offset during offline training. The initial h is set to 0.03. Note that the learning result is not sensitive to this initial h value because the neural network will learn a proper output layer scale to compensate for the mismatch from the true h value.

For each experiment, we consider four different results: [Offline NN], [Online NN], [Offline], and [Online]. [Offline NN] refers to the offline final loss of the pure Neural Network (NN) model, which is documented in Table 2.1, and not shown in graphs. [Online NN] refers to the online loss obtained by directly using offline trained NN model online. [Offline] refers to the offline final loss of the combined model, and [Online] refers to online loss calculated using our combined model with the online learning framework. Notice that [Online NN] is refered to as [NN] in the graphs for simplicity.

Following our online learning algorithm, we first train the offline component using data collected in the offline setting. Then in online testing, the online parameter adapts to the pushing outcomes observed in the online setting. The online learning results of each experiment is shown by the orange colored curve of the time-varying online prediction loss from equation (2.14) and is denoted by [Online].

To illustrate the significance of online learning with the combined model in unseen situations, we first compare the online learning results with the online prediction loss obtained from a pure neural network (NN) prediction model. The NN model is only trained for push prediction offline and is directly used to predict online pushes. The results are represented by the blue curve in each figure and is denoted by [NN]. Note the NN architecture consists of three fully connected layers, with each layer having 30 units followed by the ReLU activation. We further compare our online results with the offline training loss of the combined model, which is shown by the flat black line in each figure, and is denoted by [Offline]. Ideally, the online loss should decrease to a level similar to the offline loss in a short amount of time.

In all the experiment figures, the x-axis represents the number of time steps, and y-axis represents losses at each step. The caption: setting1/setting2, setting3/setting4, setting5/setting6 (if there is a third row for the figure) means that the first row's offline setting is setting1, online setting is setting2, and similarly for the second and the third rows. The captions only show the difference between two settings. The complete setting including object shape, surface material, online COM offsets, and pushing sides, please refer to sections A and B below. Subfigures on the left column show the total loss with mean and shaded one standard deviation area. Subfigures on the right column show the total loss break down of the left graph. The total loss is divided into positional and rotational losses, and their mean values are shown. We show the curves with a moving average of 10 time steps for better visualization.

Table 2.1 provides a summary of the results. Total loss is divided into positional and rotational losses for a detailed examination. The online losses [Online NN] and [Online] are the average losses. [Offline NN] is provided here to compare with [Offline], since the two

models should exhibit similar predictive abilities being provided with the same offline data.

MIT Push Dataset

The experiments are conducted as following. For a certain object, material, COM and pushing side setting, a total of around 20 straight push trajectories are used to test online learning. The 20 trajectories consist of different pushing points and pushing angles. For offline training, each setting has around 50 straight push trajectories. Each offline and online trajectory contains around 500 data points, collected at 250 Hz. The cases we consider include: Three objects: rect1, rect2, and rect3. Four materials: abs, delrin, plywood, pu. Pushing sides: front, left. Two COM positions: center = $(0.00m, 0.00m)$, and upper right(UR) = $(0.01m, 0.01m)$ which is done by adding the offset to the object positions in the dataset. See [15] for more details about the dataset.

Exp M1 (Fig. 2.5): different materials

We consider two experiments with rec2 object and do offline and online training with different materials. Pushing surface is front and there is no COM offset. The goal is to verify if our model with online adaption is able to adjust for different friction coefficients online.

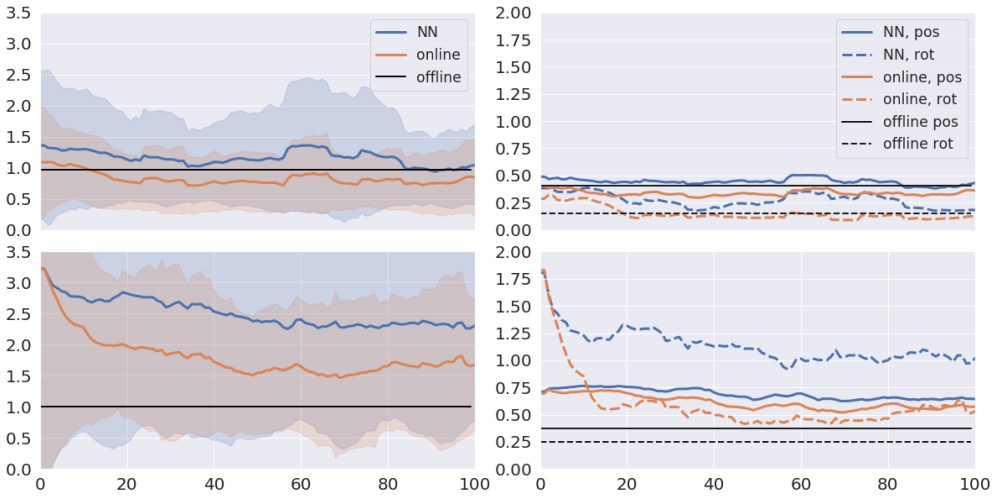


Figure 2.5: Exp M1.
delrin/abs, plywood/pu

Exp M2 (Fig. 2.6): manual online COM offsets

We consider two experiments with the rec2 object and add a manual COM offset of $(0.01m, 0.01m)$ to the object position online. Pushing surface is front for both cases. The goal is to verify

if our model with online adaption can adjust to different COM offsets.

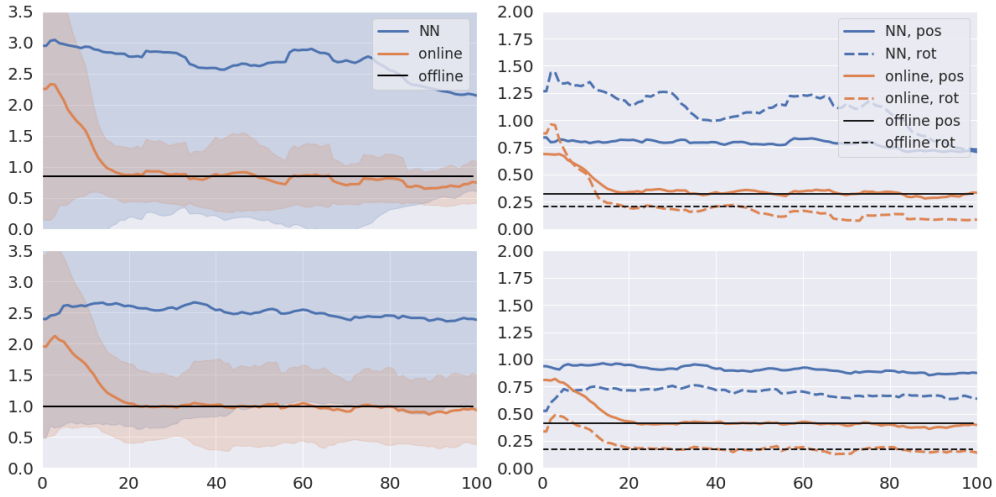


Figure 2.6: Exp M2.
abs center/abs UR, delrin center/delrin UR

Exp M3 (Fig. 2.7): different objects

We consider two experiments with different rectangular objects for online and offline training. All are on abs material. Pushing surface is front and there is no COM offset. The goal is to verify if our model can learn online to predict pushing of an unseen rectangular object.

Exp M4 (Fig. 2.8): different pushing sides

We consider two experiments with different pushing sides for online and offline training. Object is rect3 and materials are plywood and abs, respectively. There is no COM offset. The goal is to see if our model can learn online to predict pushing from a side different from the pushing side in the offline data.

Exp M5 (Fig. 2.9): different objects, materials, pushing sides, COM offsets

We consider two experiments with different objects, materials, pushing sides, COM offsets for offline and online. The goal is to see if our model with online adaption can predict pushing in a setting very different from the offline training data.

TurtleBot3 Experiments

See Figure 2.10 for our TurtleBot3 setting. The experiments are conducted as following. For a certain object, material, COM and pushing side setting, a total of 9 straight push

Figure 2.7: Exp M3.
rect1/rect3, rect2/rect3

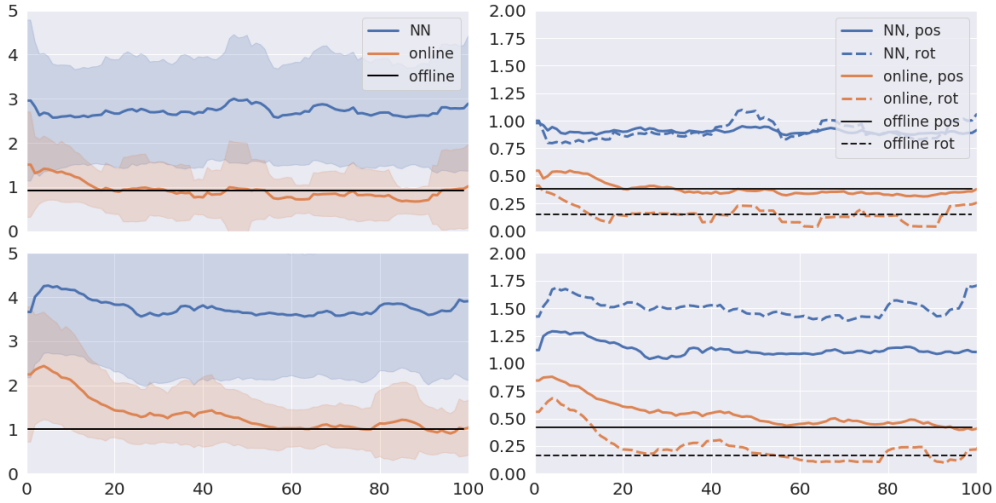


Figure 2.8: Exp M4.
plywood front/plywood left, abs front/abs left

trajectories are used to test online learning. These trajectories are a combination of different pushing points and pushing angles. For offline training, each setting has 27 straight push trajectories. Each offline and online trajectory contains around 100 data points, with a data collection frequency of 4 Hz. In order to make a reasonable prediction, we set the prediction horizon to be 2 seconds, predicting the object's position and rotation 2 seconds from the current time frame.

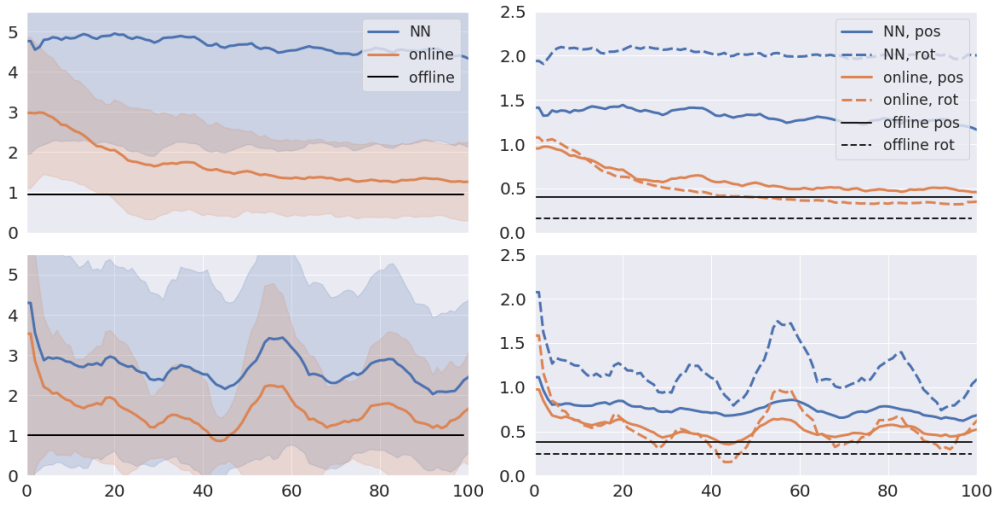


Figure 2.9: Exp M5.
delrin rect2 front center/abs rect3 left UR,
plywood rect2 front center/delrin rect3 left UR

Exps	Positional Losses				Rotational Losses			
	Offline NN	Offline	Online NN	Online	Offline NN	Offline	Online NN	Online
Exp M1	0.390	0.406	0.421	0.347	0.150	0.153	0.244	0.160
	0.382	0.376	0.614	0.554	0.269	0.247	0.914	0.492
Exp M2	0.303	0.321	0.650	0.320	0.155	0.202	0.746	0.145
	0.395	0.410	0.812	0.398	0.151	0.172	0.668	0.212
Exp M3	0.388	0.388	0.854	0.562	0.255	0.209	0.356	0.176
	0.307	0.322	0.992	0.380	0.161	0.144	0.564	0.088
Exp M4	0.383	0.381	0.938	0.364	0.148	0.148	1.022	0.176
	0.425	0.419	1.130	0.477	0.167	0.164	1.552	0.212
Exp M5	0.391	0.400	1.167	0.506	0.149	0.156	1.906	0.353
	0.378	0.381	0.652	0.476	0.265	0.246	0.948	0.462

Table 2.1: Experiments of the MIT push dataset.

Exp R1 (Fig. 2.11): different materials

We consider two experiments with box1 and do offline and online training with different materials. Pushing surface is front and there is no COM offset. As in Exp M1, the goal is to verify the ability of our model to adapt to different friction coefficients online.

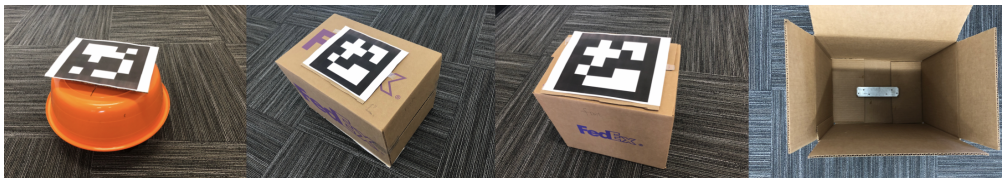


Figure 2.10: TurtleBot Experiment Setup.
From left to right: TurtleBot3, box1, box2, and an opened box with a weight. The objects we used are standard boxes from FedEx, box1 dimensions are 18" by 12" by 12" and box2 dimensions are 12" by 9" by 6". The weight we used to create different COM offsets is about 1 pound. We track the robot and object positions and orientations by their ArUco markers using a ceiling camera. We consider two different surface materials, paper (original) and plastic for the bottom of the boxes, and three COM positions, center, upper right corner (UR), lower right corner (LR), done by moving the weight to the associated positions in the box.

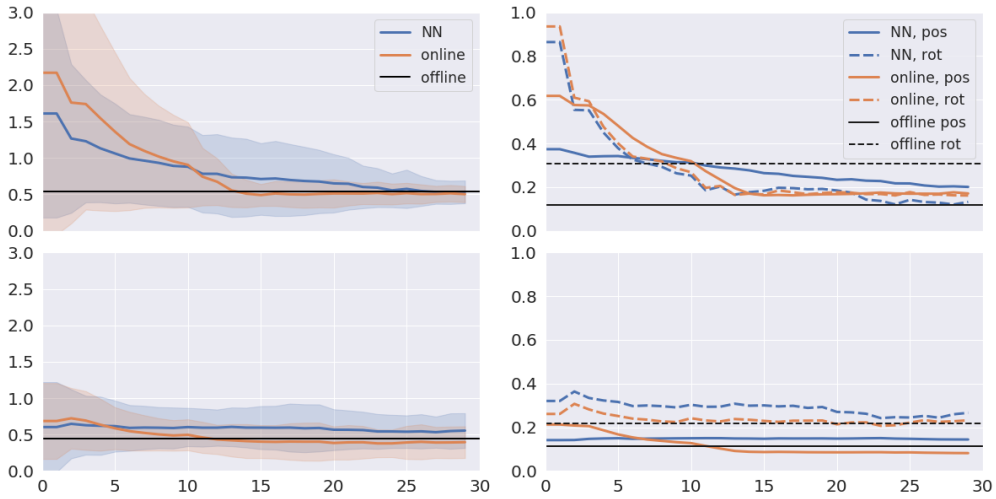


Figure 2.11: Exp R1.
paper/plastic, plastic/paper

Exp R2 (Fig. 2.12): different COMs

We consider two experiments with different COM settings online. We use box1 with paper surface in all cases and pushing surface is front. As in Exp M2, the goal is to verify if our model with online adaption can adjust to different COM offsets.

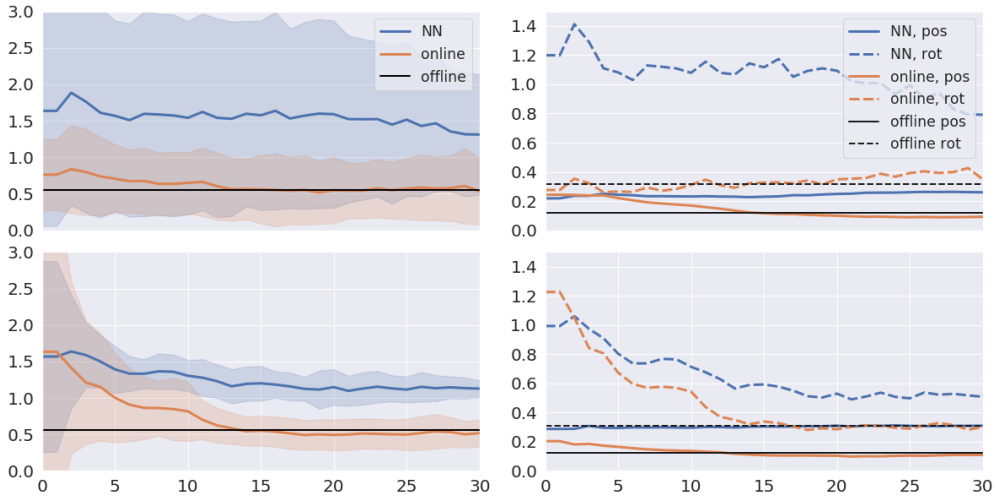


Figure 2.12: Exp R2.
paper center/paper LR, paper center/paper UR

Exp R3 (Fig. 2.13): different objects

We consider two experiments with different boxes for online and offline training on either paper or plastic. Pushing surface is front and there is no COM offset. As in Exp M3, the goal is to verify if our model with online adaption can predict pushing of an unseen rectangular object.

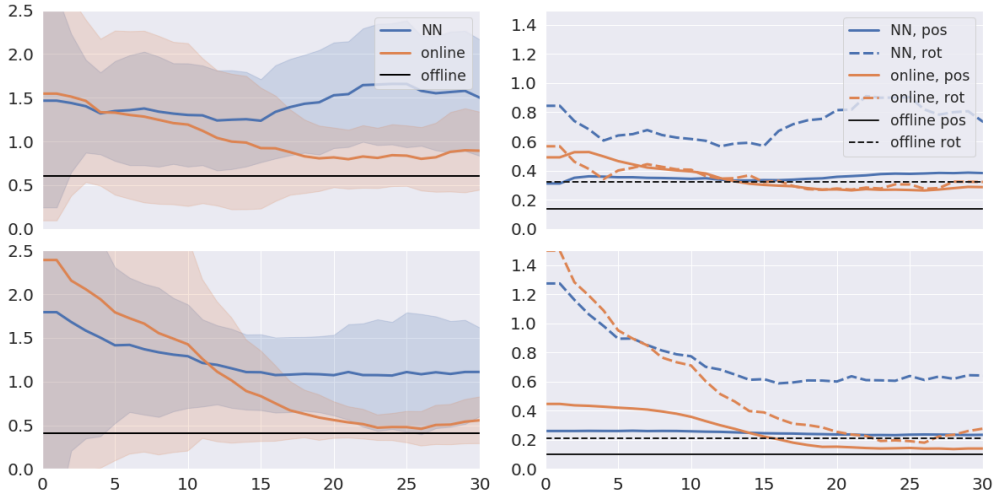


Figure 2.13: Exp R3.
paper box1/box2, Exp R3. plastic box1/box2

Exp R4 (Fig. 2.14): different pushing sides

We consider two experiments with different pushing sides for online and offline training with box1. There is no COM offset. As in Exp M4, the goal is to see if our model can learn online to predict pushing from a side different from the pushing side in the offline data.

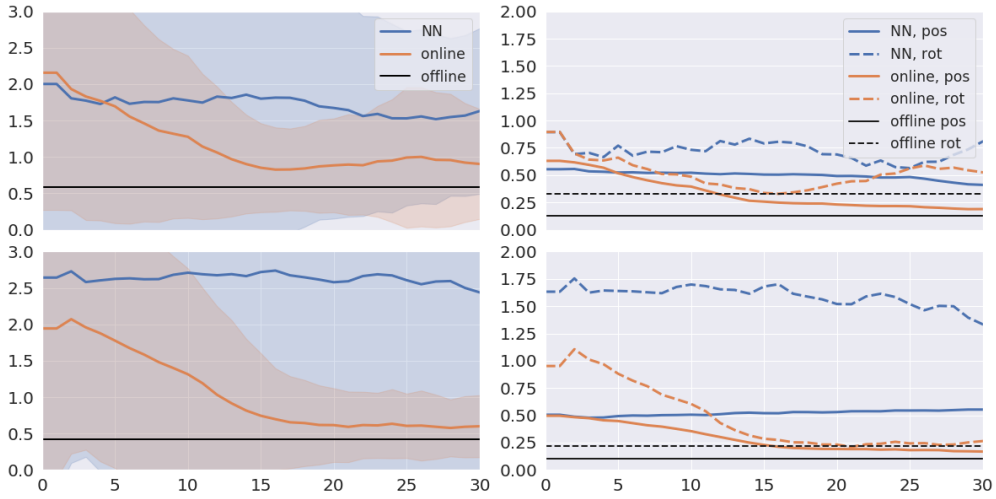


Figure 2.14: Exp R4.
paper front/paper right, plastic front/plastic right

Exp R5 (Fig. 2.15): different objects, materials, pushing sides, and COM offsets

We consider two experiments with different objects, materials, pushing sides and COM offsets for offline and online. As in Exp M5, the goal is to see if our model with online adaption can predict pushing in a setting very different from the offline training data.

Discussion

Despite the fact that the two sets of experiments vary in data collection frequency, pusher size and object type, the results exhibit similar behavior. From Table 2.1, the offline training loss of our combined model [Offline] is of similar magnitude as that of the baseline offline NN model [Offline NN] as well as existing data-driven models [9]. In all the experiments, the combined model with online learning [Online] is able to adjust to the online situation quickly and achieve a loss similar to the offline combined model training loss [Offline]. On the other hand, even though the NN baseline has the accurate offline training score, its online prediction [Online NN] is poor due to the lack of online adaptation.

Exp M1 and R1 show that the combined model can efficiently learn to predict pushing outcomes when faced with a different surface material online for the two different robots

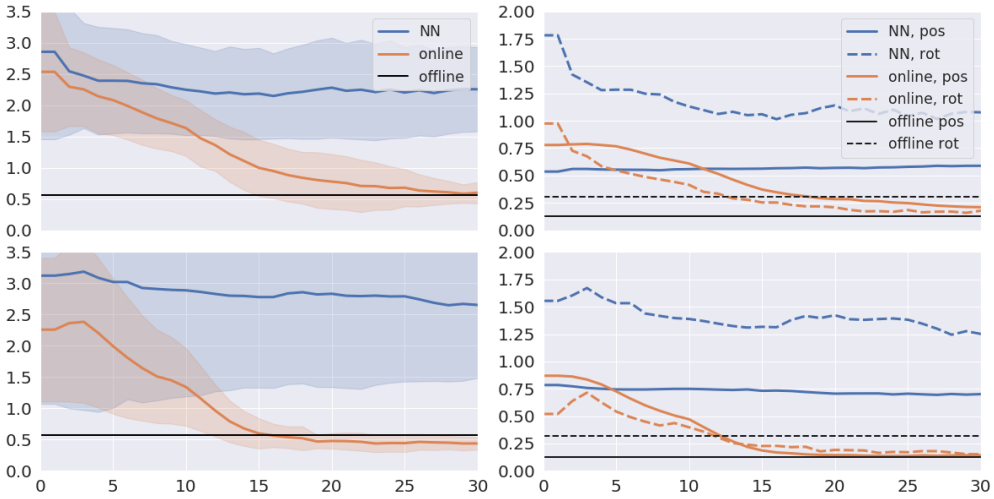


Figure 2.15: Exp R5.
paper box1 front center/plastic box2 right LR, paper box1 front center/plastic box2 right UR

Exps	Positional Losses				Rotational Losses			
	Offline NN	Offline	Online NN	Online	Offline NN	Offline	Online NN	Online
Exp R1	0.116	0.117	0.277	0.287	0.319	0.307	0.273	0.292
	0.105	0.112	0.147	0.118	0.183	0.219	0.290	0.236
Exp R2	0.160	0.120	0.243	0.123	0.320	0.316	0.880	0.321
	0.138	0.126	0.310	0.121	0.282	0.307	0.616	0.435
Exp R3	0.118	0.138	0.365	0.316	0.281	0.325	0.628	0.312
	0.098	0.099	0.242	0.213	0.178	0.212	0.677	0.455
Exp R4	0.123	0.128	0.439	0.269	0.291	0.327	0.744	0.498
	0.105	0.104	0.507	0.232	0.182	0.218	1.447	0.388
Exp R5	0.141	0.125	0.579	0.371	0.296	0.306	1.153	0.312
	0.129	0.127	0.729	0.281	0.293	0.316	1.394	0.296

Table 2.2: Experiments by TurtleBot3.

under different offline/online combinations. Note that in the first experiment of Exp R1 the online loss for the NN model also decreases. This actually is not due to learning, but because the magnitude of rotational movement on the plastic surface decreases rapidly along the trajectory so pushing prediction becomes easier.

Exp M2 and R2 show that our method is also very cable to quickly online learn the unknown COM offset under various situations. In these experiments, the inaccurate and high variance online loss of the NN baseline is expected because the NN model assumes an incorrect COM position.

In Exp M3 and R3, even though the robot pushes a different object in the online setting

than the one seen in offline data, the two objects are of the same rectangular shape. Pushing two different rectangular objects is basically similar to pushing the same one with different COM positions. Therefore, the results are similar to those in Exp M2 and R2 besides the first experiment of Exp 3. In the case the initial positional prediction loss is very large which might be due to the very different lengths of the two objects. Despite the initial loss, our method still can learn to quickly provide much improved predictions.

Since pushing different sides can be viewed as pushing different rectangular objects, the results of Exp M4 and R4 are indeed similar to the ones of Exp M3 and R3. Exp M5 and R5 further demonstrate the robustness of our method that with all the attributes being different, the online learning algorithm is still able to adapt to the new situation within 30 time steps in all the cases.

2.6 Conclusion

In this chapter, we described the problem of mobile robot planar pushing, and proposed a combined prediction model and an online framework for planar push prediction. Our method takes advantages of both offline and online learning by utilizing a neural network module and several analytical components. We conducted two sets of real-robot pushing experiments to verify the prediction accuracy and online adaptation ability of our model. The experiments showed that our model can achieve similar offline performance as a pure neural network model. In situations different from offline data, with online learning our combined prediction model is able to reduce the loss to a level similar to the offline loss, as well as reducing the uncertainty in both positional and rotational predictions.

Chapter 3

Mobile Robot Pushing Control Policy using Reinforcement Learning

3.1 Introduction

As discussed in Chapter 2, pushing is an essential motion primitive in a robot's manipulative repertoire with applications from daily life to extreme environments [20]. In the previous chapter, we also discussed a novel approach for building a planar pushing prediction model. Said model can be further used in trajectory planning and control for mobile robots to push objects. Similar to our work in Chapter 2, many existing studies on mobile robot pushing have attempted to derive models for calculating pushing outcomes, or they have used model-based data-driven methods to learn a model to plan for trajectories. However, pushing is generally difficult to model accurately due to frequent changes in contact modes and perturbations from the surrounding environment. Therefore, many model-free reinforcement learning approaches have been proposed to solve the pushing problem. In this chapter, we continue to study mobile robot pushing in a 3D environment by exploring a novel soft actor-critic (SAC)-based model-free framework to output control policies for pushing.

For robots, achieving goals through pushing is challenging due to the complexity that arises from unknown frictional forces, nonlinear dynamics, and the fact that the outcome of a push is difficult to predict as well as highly uncertain. Numerous types of approach have been implemented for pushing under different assumptions and levels of computational power, including purely analytic, hybrid, simulation, and data-driven methods. Among these methods, it is data-driven methods that have drawn increasing interest, especially those based on deep learning. This is because pushing models are extremely difficult to analytically model and analyze; however, although many data-driven and deep-learning approaches exist for learning pushing models, relatively little work has been conducted on the application of reinforcement learning for learning a model specifically for pushing. This is despite the fact that deep reinforcement learning has numerous successful applications in the robotics field [23, 24]. Our work in this chapter aims to fill this research gap.

In general, there are two kinds of learning structures in reinforcement learning, which we refer to as direct and hierarchical control in this chapter. Direct control policy is an end-to-end policy that directly outputs desired action commands to the robot, whereas hierarchical control policy outputs high- and low-level policies separately. Hierarchical reinforcement learning [25, 26, 27, 28] networks have served as valuable tools for solving tasks with large time horizons and/or sparse rewards. These models achieve superior performance by temporally abstracting the rate at which actions from various levels of the hierarchy are issued, thereby allowing for higher-level policies to more easily perceive the effects of actions over larger time intervals. The actions of higher-level policies are passed to the levels below them as a goal; then, the low-level policies are trained to achieve these goals through a goal-conditioned reward function, or they are simply designed using model-based control methods. This two-level hierarchical model often serves as a more robust and efficient method than an end-to-end direct learning structure. Most existing hierarchical reinforcement learning approaches focus on subgoals that are in the same space as the final goal, such as intermediate locations toward a final location [26] or intermediate joint configurations towards a final one [28]. However, the subgoals in our problem setup are heterogeneous; for example, while the final goal is to design a two-dimensional robot wheel velocity command, the intermediate subgoals is to identify the robot's desired next position relative to the current position.

In this chapter, we investigate the use of reinforcement learning to perform mobile robot pushing. Our main contributions are as follows: (1) We propose a novel SAC-based framework that incorporates the learning of pushing dynamics; (2) we provide a comparison in terms of exploration efficiency and pushing strategy for direct and hierarchical control policies using our framework; and (3) we verify the generalization abilities of our framework in a simulation.

The remainder of this chapter is organized as follows: Section 3.2 provides a brief overview of works related to mobile robot pushing as well as direct versus hierarchical control strategies; Section 3.3 provides a detailed explanation of the proposed approach; Section 3.4 includes experiments that provide a comparison of the two control strategies as well as some generalization results; and Section 3.5 concludes the chapter.

3.2 Related Work

Pushing

Pushing is a complex mechanical system that is difficult to model accurately due to unknown system parameters such as coefficients of friction and pressure distributions, which restricts the application of the purely analytical approach [29, 1, 30] and the hybrid approach [31, 32]. While physics engines, such as Bullet Physics, the Dynamic Animation and Robotics Toolkit (DART) and MuJoCo [33], offer great value for robotic applications, e.g. by taking into consideration dynamic interaction and 3D objects, they nevertheless require explicit object modelling and extensive parameter. There are many data-driven and deep learning

approaches to learn the pushing control policy from examples. This approach can be mainly summarized as the model-based methods and model-free methods:

Model-free methods

One reinforcement learning work in pushing is [34]. The authors propose a spatial action map as a new action representation for mobile manipulation. By using ConvNets to infer spatial action maps from state images, action predictions are spatially located at local visual features in the scene. Then, the predicted action can be used to enable faster learning of complex behaviors for robot pushing with reinforcement learning.

However, there are some limitations for this work. The first one is that this method is limited to a discrete action space due to the application of Q-learning based method. The second one is the gap between high-level control and low-level control. In the paper, only high-level motion primitives is used by the spatial action maps, which assumes that low-level control can be handled separately. However, as it is hard for the low-level controller to exactly track the trajectory planned by the high-level controller, this method inherently has its limitations. Besides, in the paper, the mobile robot is assumed to conduct line contact with the simple cubic objects, which is a quite simple scenario. Chances are high that the method might not be able to generalize well to scenarios with irregular shaped objects.

Model-based methods

There are several studies in robot pushing that focus on estimating the physical parameters, predicting the outcome of pushing actions, and learning a dynamic model. In [35], the authors studied the learning of physical properties such as mass and cohesion of the objects. Deep reinforcement learning is applied for the robot to learn strategies that balance the cost of gathering information and the cost of inaccurate estimation. Another approach is to learn a dynamics model. Paper [36] presented a deep learning framework for learning simple physics simulators. They utilize an object-based representation of the environment and decompose dynamics into pairwise interaction between objects. However, their evaluation is limited to simple 2D cases. Authors in [17] proposed Push-net, a deep recurrent neural network to deal with the problem of quasi-static planar pushing to re-orient and re-position objects. Their approach uses a long short-term memory (LSTM) module to remember the pushing interactions and estimate the COM of the object by observing the behavior of objects under random interactions. The results indicate that Push-Net is able to generalize to novel objects. It is worth mentioning that their application is on the robot arm, which is different from our work.

Direct vs. Hierarchical

Pushing tasks are usually long-horizon and composed of heterogeneous phases of pure navigation, pure manipulation, and their combination. Using the wrong part of the embodiment

is inefficient and hinders progress. The structure mentioned above is well suited for a hierarchical reinforcement learning (HRL) solution, where the high-level learns the phase type and sets a subgoal, and the low-level learns to achieve it.

There already exists some hierarchical reinforcement learning works in robot pushing. In [37], the author uses RelMoGen to do interactive navigation. ReLMoGen is a hierarchical framework that integrates classical motion generation with reinforcement learning to solve mobile manipulation tasks. ReLMoGen takes advantages of both worlds: learning complex subgoal prediction from high dimensional observations via Reinforcement Learning and precise low-level action execution via Motion Generator. It is worth mentioning that the framework generates different subgoals for the robot arm and the base respectively, which are basically positions and orientations for arm and base. As a result, the proposed RelMoGen framework succeeded in completing the task for the robot to plan a push to the box to clear out the way. The authors have demonstrated better task completion and higher training efficiency compared to other learning based approaches including HRL4IN and SAC [38, 39, 24]. The learned policies with ReLMoGen are also robust and can transfer to different motion planners after training.

HRL4IN proposed in [38] is another paper using the hierarchical RL architecture for Interactive Navigation. In this work, a high-level policy is trained to output a subgoal, and a low-level policy is trained to execute the goal, which is different from RelMoGen, where the subgoal is executed by some common low-level controllers. The performance of HRL4IN has been evaluated against the flat PPO and HAC. The results indicate that HRL4IN achieves higher reward and solve the tasks more successfully and efficiently.

However, both papers listed above only focus on interactive navigation, not on the mobile robot pushing objects to a desired location. The behavior of the object under pushing is highly influenced by physical properties, such as mass and cohesion of objects, so it is very important to improve the framework with pushing dynamics considered.

3.3 Methodology

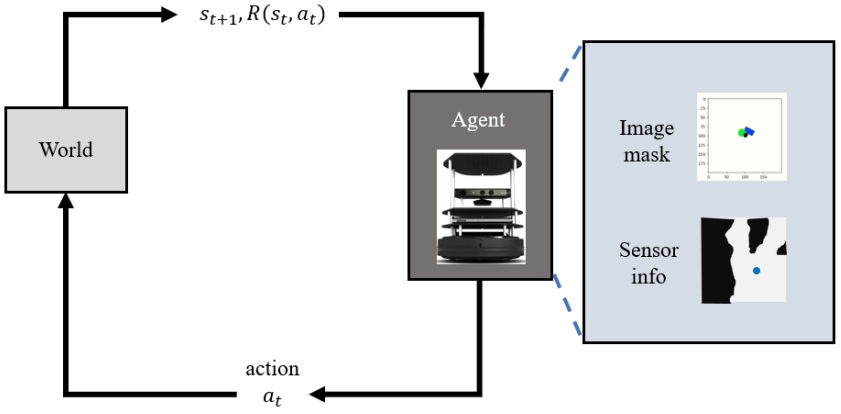
Problem formulation and training schemes for direct and hierarchical control

We first describe direct control setup. We formulate a robot pushing task as a time-discrete Partially Observable Markov Decision Process (POMDP) defined by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{R}, \gamma)$. Here, $\mathcal{S}$ is the state space; $\mathcal{A}$ is the action space; $\mathcal{O}$ is the observation space; $\mathcal{T}(s^{'}|s, a)$ is the state transition model; $\mathcal{R}(s, a)$ is the reward function; $\gamma \in [0, 1)$ is the discount factor. We assume that the state is not directly observable and learn a policy $\pi(a|o)$ conditioned on observations $o \in \mathcal{O}$. Therefore, the agent following the policy π obtains an observation o_t at time t and performs an action a_t, receiving from the environment an immediate reward R_t and a new observation o_{t+1}. The goal of RL is to learn an action selection policy π that maximizes the discounted sum of rewards.

The reward for each step is defined in Eq. (5.3). Here we penalize the distance between object $\mathbf{O}_{x,y}$ and target $\mathbf{T}_{x,y}$, the distance between robot $\mathbf{R}_{x,y}$ and object $\mathbf{O}_{x,y}$, where $\mathbf{O}_{x,y}, \mathbf{T}_{x,y}, \mathbf{R}_{x,y}$ denote the coordinate position of the object, target and robot. In order to encourage the robot to be in contact with the object, we add a large reward if the distance between robot and object is small at each step. We further give a hige reward for success, when the object arrives at the target location within a $0.2m$ tolerance.

$$R_t(s_t, a_t) = [-2 \cdot \ell_2(\mathbf{O}_{x,y}, \mathbf{T}_{x,y})] + [-0.5 \cdot \ell_2(\mathbf{R}_{x,y}, \mathbf{O}_{x,y})] + [10 \cdot I_{\ell_2(\mathbf{R}_{x,y}, \mathbf{O}_{x,y}) < 0.05}] + [50 \cdot I_{\text{success=true}}] \tag{3.1}$$

where $\ell_2(\cdot, \cdot)$ measures the euclidean distance. After receiving the observations o_t and the reward R_t at time t, the agent takes the state and outputs wheel velocity action a_t, and sends it to the environment, which then outputs the resulting state and rewards. A training scheme for the POMDP formulation of the direct control is illustrated in figure 3.1.



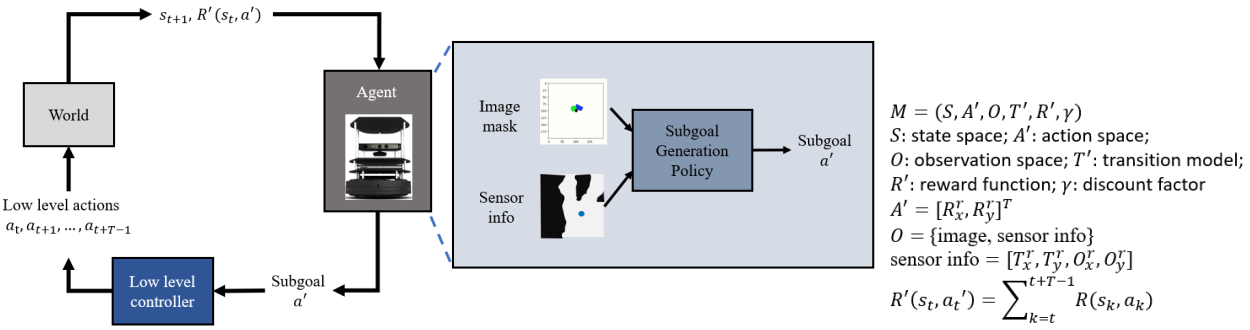
(a) Training scheme (b) Notations

Figure 3.1: Training scheme for the POMDP formulation of direct control.

Similar to [37], for hierarchical control, we redefine the problem and lift the action space. $\mathcal{A}'$ is the lifted action space, which is the robot's next position relative to the current position. We use relative information to only capture the relationships between robot, object and target, not the global information. Once we have the lifted action a' (also noted as a subgoal), a lower level controller is used to plan actions a_t to a_{t+T-1}, which are the original wheel velocity commands. The low level controller that we used is a simple open loop controller that turns and moves the robot to the desired location. The transition function now depends on a'. The lifted reward function is the accumulated reward obtained from executing the sequence of actions from the Motion Generator (MG).

We have the state and reward at time t, the agent takes the state and output a', and a low level controller executes a' and sends it to the environment, which then output the

resulting state and rewards. We argue that, by lifting the action space, we can efficiently use RL to solve more complex, long-horizon tasks that are difficult to solve with existing RL methods in the original action space. A training scheme for the lifted POMDP formulation of hierarchical control is showed in the figure 3.2.



(a) Training scheme (b) Notations

Figure 3.2: Training scheme for the lifted POMDP formulation of hierarchical control.

For both direct and hierarchical control, the observation space consists of an image mask and sensor information. The image mask contains information of robot, object and goal, and it is a transformed bird-eye view of the original image. The sensor information contains target's position relative to the robot's position: $[\mathbf{T}_x^r, \mathbf{T}_y^r]$, and object's position relative to the robot's position: $[\mathbf{O}_x^r, \mathbf{O}_y^r]$. A complete summary of observation space, action space, reward function and total cumulative reward for the two control policies is given as follows:

- Observation space $\mathcal{O}$ = [image,sensor] for both direct control and hierarchical control contains where
  (1) image is the image mask of size 200 by 200.
  (2) sensor is the sensor's information $[\mathbf{T}_x^r, \mathbf{T}_y^r, \mathbf{O}_x^r, \mathbf{O}_y^r]$.
- Action space:
  (1) $[v_{\text{left\_wheel}}, v_{\text{right\_wheel}}]$ for direct control.
  (2) $[R_x^r, R_y^r]$ relative to previous robot position $[x, y]$ for hierarchical control.
- Reward for each step:

  $$R_t(s_t, a_t) = [-2{\cdot}\ell_2(\mathbf{O}_{x,y}, \mathbf{T}_{x,y})]+[-0.5{\cdot}\ell_2(\mathbf{R}_{x,y}, \mathbf{O}_{x,y})]+[10{\cdot}I_{\ell_2(\mathbf{R}_{x,y},\mathbf{O}_{x,y})<0.05}]+[50{\cdot}I_{\text{success=true}}].$$

- Total cumulative reward:
  (1) $\sum_{i=1}^{\infty} \gamma^{i-1} R_i$ for direct control.
  (2) $\sum_{k=t}^{t+T-1} \sum_{i=1}^{t} \gamma^{i-1} R_i$ for hierarchical control.

Training methodology

A Soft Actor Critic (SAC) is used for training. For the actor network, the image mask and sensor information go through preprocessing layers separately and are concatenated later. Then it goes through fully connected layers as well as a projection network to output the policy. For the critic network, the states go through the same encoder structure, and is combined with processed action information. The combined information goes through LSTM layers and a fully connected layer to output Q values. The architectures for the actor network and the critic network are given in the figure 3.3 and 3.4, respectively.

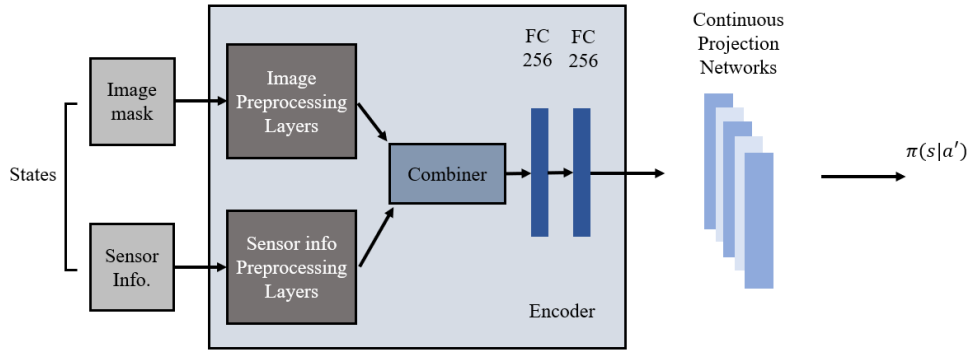


Figure 3.3: An architecture for the actor network.

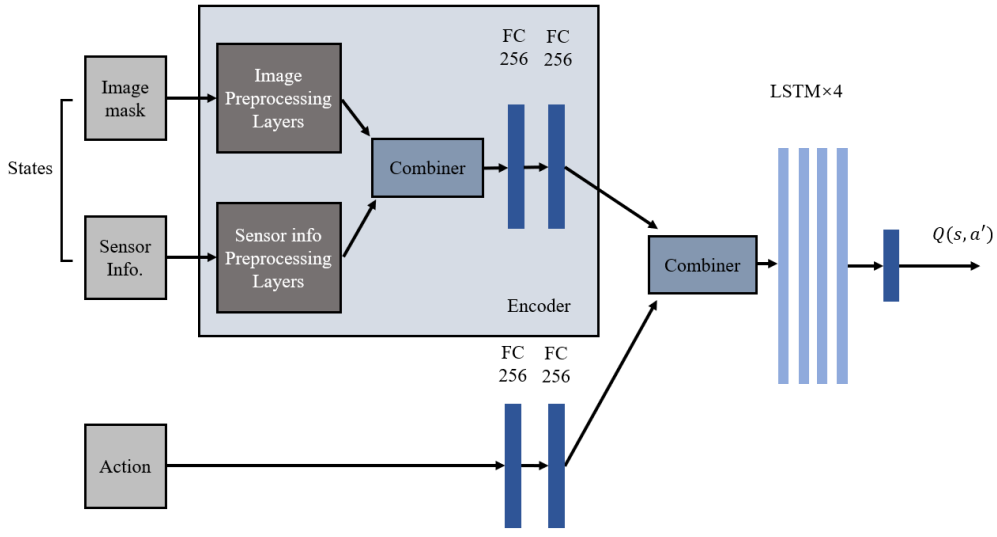


Figure 3.4: An architecture for the critic network.

Image masking

The raw image data collected by the environment is processed to a colored mask as shown in Figure 3.5. The image is cropped to a 200 by 200 sized mask oriented based on robot's coordinate, so the robot is always centered in the image and facing front. This step ensures no global information is captured, only the relationships between robot, object and target are recorded so training takes less data. We further put a blue mask for the object, a green mask for the target, a black mask for the robot, and a white mask for background. The target cylinder's radius indicates the $0.2m$ tolerance for success condition. The mask helps with the model's generalizability as it standardizes the input.

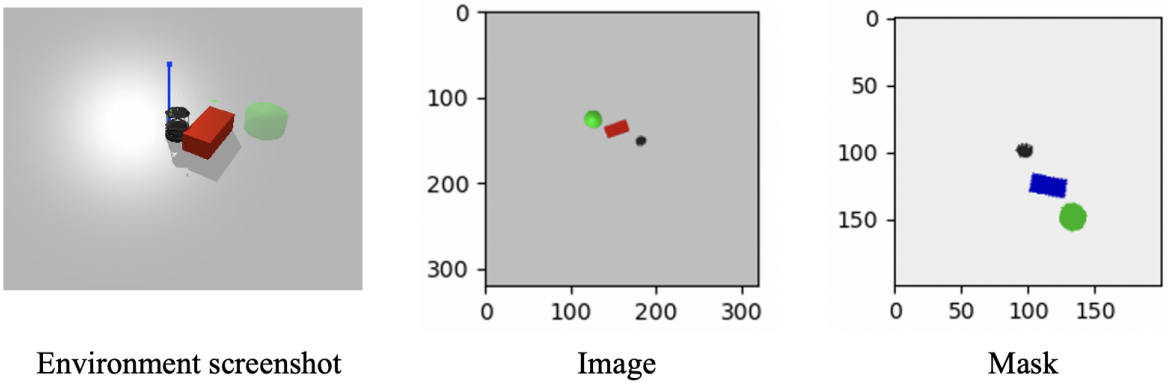


Figure 3.5: Image masking method.

LSTM

We include four LSTM layers in the critic network only, as the input is a combination of states and actions. The ability of LSTM layers to learn push dynamics with time-sequenced state-action inputs is validated in Experiment's second section.

3.4 Experiments

To validate our framework as well as compare hierarchical control and direct control policies, we design three sets of experiments. The experiments are conducted in PyBullet with a Turtlebot2. The first part presents a detailed comparison between hierarchical and direct control. The second part focuses on validating the benefits of our network design. The last part presents generalization results of our framework.

Hierarchical vs. Direct Control

In this section, we aim to compare the push strategy and exploration efficiency of hierarchical control and direct control policies. For all the experiments in this section, we train the agent to push a red rectangle of size 0.5 by 0.8 by 0.3 m. Mass of the object is set to be 0.3kg and uniformly distributed. After every episode, the positions of the robot, object and target are randomly reset. The reset strategy is as follows: first, the robot is randomly placed within a ($R_0 = 0.5m$) circle around the origin, with uniform distribution. Then the object is randomly placed around the robot within in a ($R_0 = 0.4m, R_1 = 1.0m$) disk area, with robot-object angle set to differ from robot's own orientation no more than 45 degrees. Finally, the target position is set to be within a ($R_0 = 0.5m$) circle area centered around the object, with object-target angle set to differ from robot-object angle no more than 45 degrees. The angle constraints are to make sure that object is approximately between the robot and the target at each run.

Training curves for two control policies are shown in Figure 3.6. Notice for all the training curves presented in all the experiment sections, the lines represent the average of three parallel environments, the standard deviation is not shown since the total number of environment is too small to show any meaningful variance results. For all the training curve plots, we use two criteria. The first criteria is average episode length, which measures how many steps the agent take to finish the episode. The other criteria is average episode return, which measures the average return for one episode. Hierarchical control takes less steps to plan for the trajectory since its action space is robot's position. The average return for hierarchical control is higher because its return is calculated based on less action steps. However, the trajectory for hierarchical control is not smooth due to our simple lower level controller design.

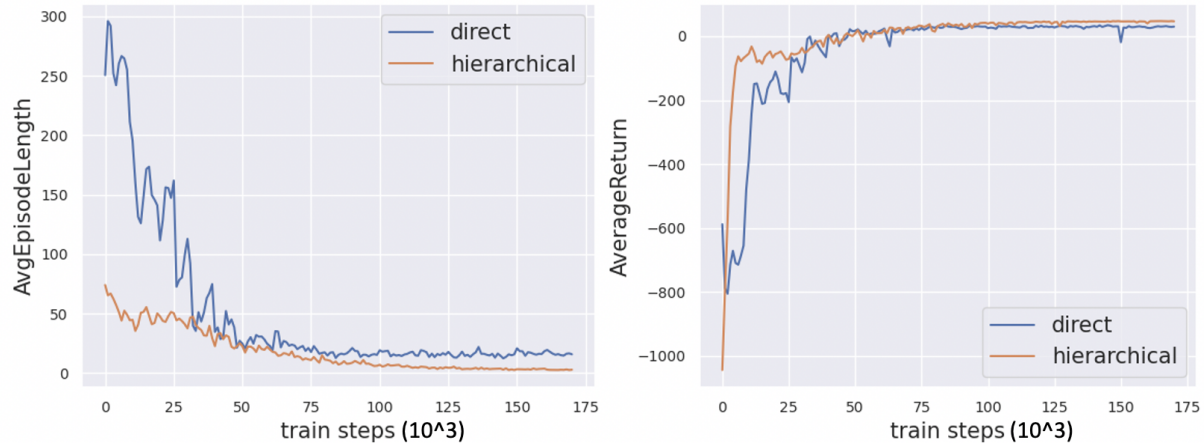


Figure 3.6: Comparison of hierarchical and direct control training processes for training a single object.

In order to compare the exploration strategies, we conduct another set of experiments with fixed position reset: after each episode, the robot, object and target positions are set to the same locations as the initial settings for the last episode respectively, so the RL agent is being trained to push the object from the same starting location to the same target location every time.

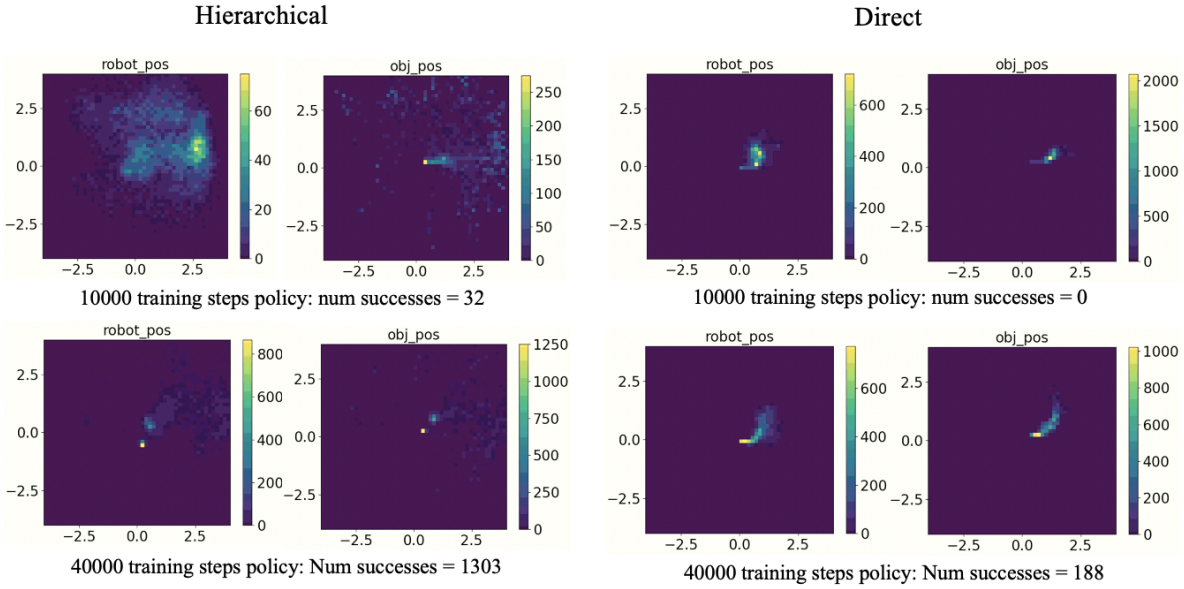


Figure 3.7: Comparison of hierarchical and direct control exploration efficiencies. The first row shows hierarchical and direct control exploration efficiencies when the policies are trained after 10000 steps. The second row shows the two control policies' exploration efficiencies when the policies are trained after 40000 steps. The object initial position is set to be $[0.4, 0.2]m$, without z-axis orientation offset. The target position is set to be $[1.0, 1.0]m$. The goal is to push the object to within 0.2 m of target position.

We save the policy at different stages of the training process, and use the corresponding policy at that stage to run it for 10000 steps in the environment. During the policy execution, we collect the robot and object positions for these 10000 environment steps. We also record number of successes. As shown in Figure 3.7, both policies converge after 40000 training steps as desired, and notice when the training just started at 10000 training steps, both robot and object state spaces were explored very efficiently for hierarchical control, as their locations vary significantly compared to that of direct control. We also consider a harder task that includes desired target orientation, as described in Figure 3.8. For direct control, the robot is not able to explore well when training just started, and due to reward engineering and lack of success cases, the robot gradually ceased to explore when trained further. Thus we hypothesize that hierarchical control may be less sensitive to reward engineering.

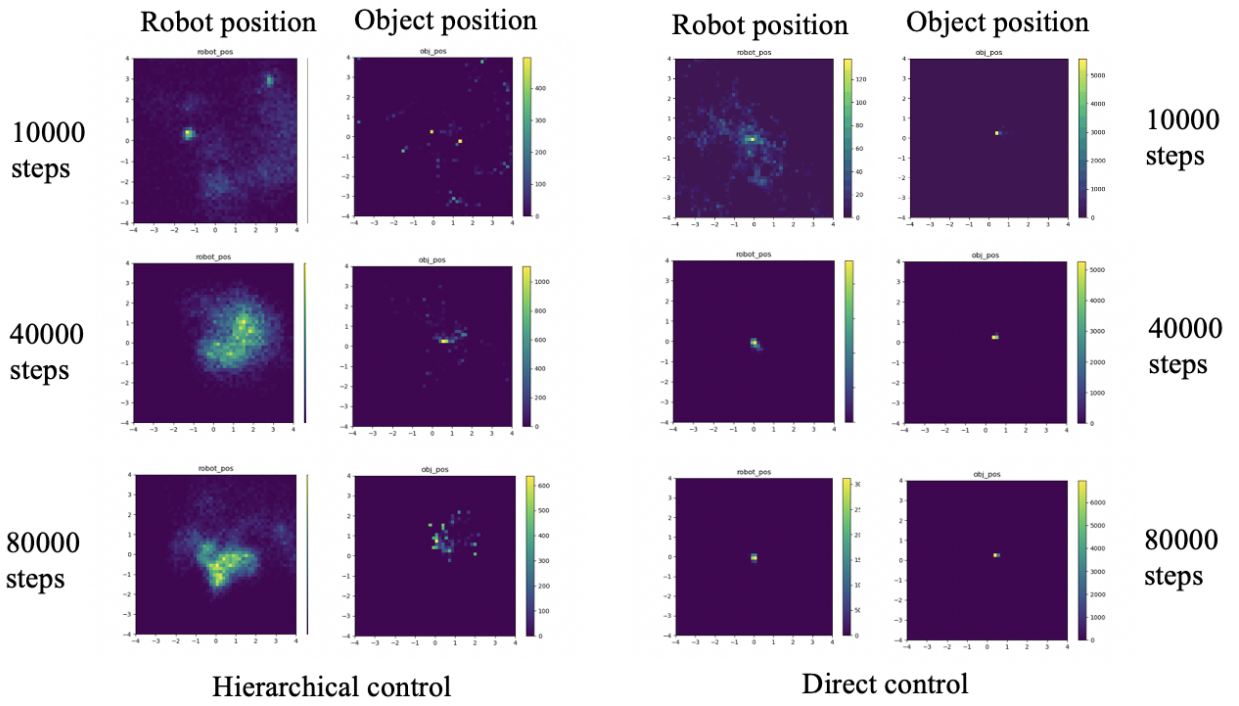


Figure 3.8: Comparison of hierarchical and direct control exploration efficiencies with a harder task.

Task here includes a desired target orientation for the object. Another penalty term $-abs((O)_{yaw}, (T)_{yaw})]$ is added to the reward function to penalize the difference between current object and target orientations. The rows show hierarchical and direct control exploration efficiencies when the policies are trained after 10000, 40000, 80000 steps, respectively. The object initial position is set to be $[0.4, 0.2]m$, without z-axis orientation offset. The target position is set to be $[1.0, 1.0]m$.

Benefits of Our SAC Network Design

Benefits of adding sensor information

Our input consists of an image mask and sensor information. The sensor information $[\mathbf{T}_x^r, \mathbf{T}_y^r, \mathbf{O}_x^r, \mathbf{O}_y^r]$, can be obtained by learning from the mask in theory. However, learning sensor information from the mask can be slow and noisy, also in many real world applications such sensor information are sometimes available so learning them would not be necesseary. As shown in Figure 3.9, adding sensor information helps with the training for both hierarchical and direct control policies. For hierarchical control, average episode length decreases significantly when sensor information is included. For direct control, the algorithm starts training with much better return with the addition of sensor information.

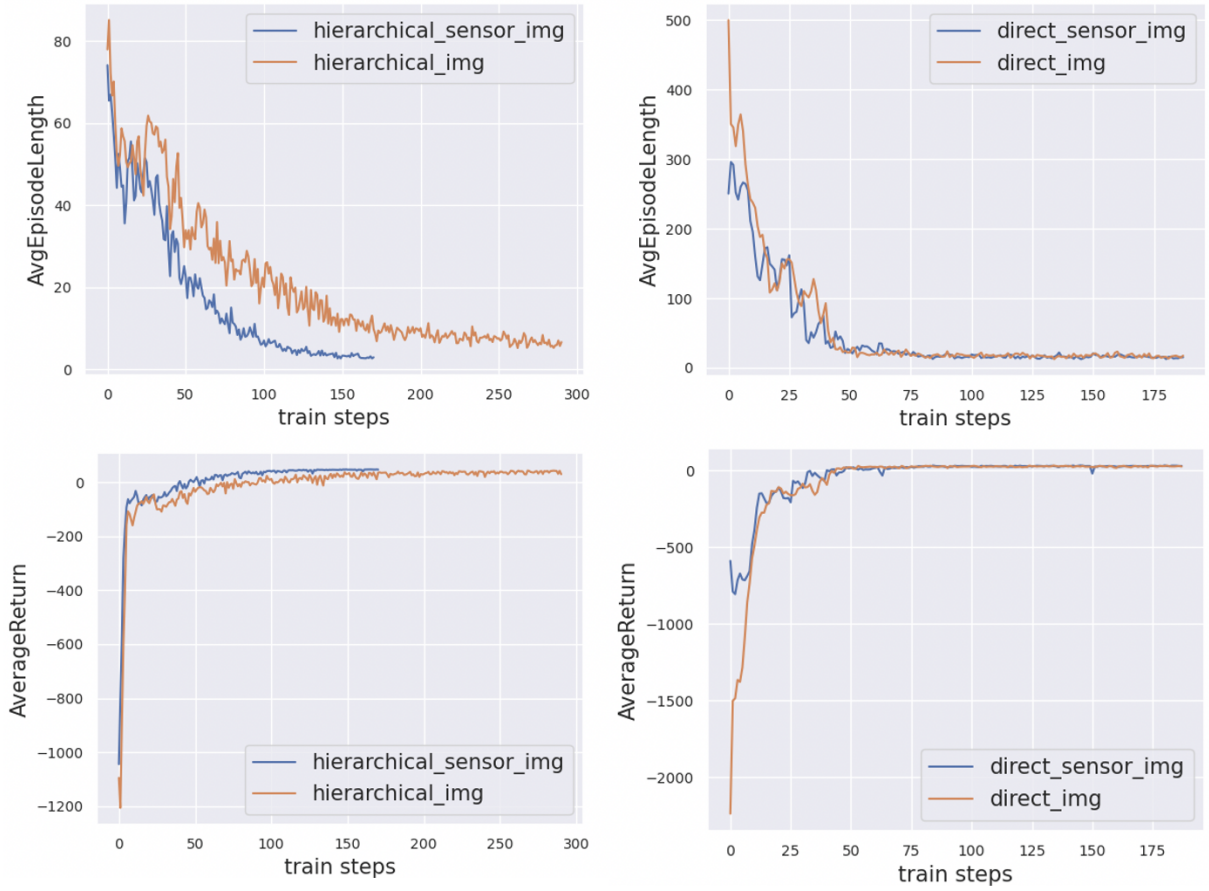


Figure 3.9: Training curves for hierarchical and direct agents, showing the benefits of adding sensor information.

sensor_img means the agent is trained with both sensor and imaging mask inputs, *_img* means the agent is trained with only mask input.

Benefits of LSTM module

In this section we validate the ability of LSTM module to capture pushing dynamics. For direct control, we first train two RL agents offline: one with LSTM module added to the critic network, the other framework is trained with regular SAC. Both agents utilize the masking techniques, and both agents are trained in the same environment with random target and object position resets for each episode.

After the training converged, we take the two agents and run them directly with an object that has the same appearance and shape as training, but with different center of masses (COM). During training, the training object to be pushed is a 0.5 by 0.8 by 0.3 meter red rectangle, with center of mass at the object center. During testing, the object

is a rectangle with the same visual appearance, but center of mass is 0.1 and 0.2 meters away from the object center, in positive x and y axis, respectively. During testing we change the object and target positions manually to validate the performance. Since the training and testing objects have different center of masses, we expect their pushing dynamics to be different. If LSTM captures the dynamics, the agent trained with LSTM should be able to output a better strategy than the agent trained without LSTM, since it knows the testing object has a different pushing dynamic through the LSTM module.

Figure 3.10 indicates LSTM helps with the training as both the average episode length and average return improves rapidly after training started. Figure 3.11 clearly shows that when the critic in SAC is incorporated with LSTM module, the agent becomes aware and more conservative when faced with an object that has different pushing dynamics (different COM), and breaks contact more often to adjust for the position of the objects. The resulting trajectory for the object then becomes smoother for LSTM agent, and the robot will not overpush the object beyond the target, as shown in setups 2 and 3, compared to the regular agent. Table 3.1 shows the time each agent used to finish the episode. The time results are comparable, meaning that the more conservative LSTM agent doesn't necessarily spend more time pushing the object.

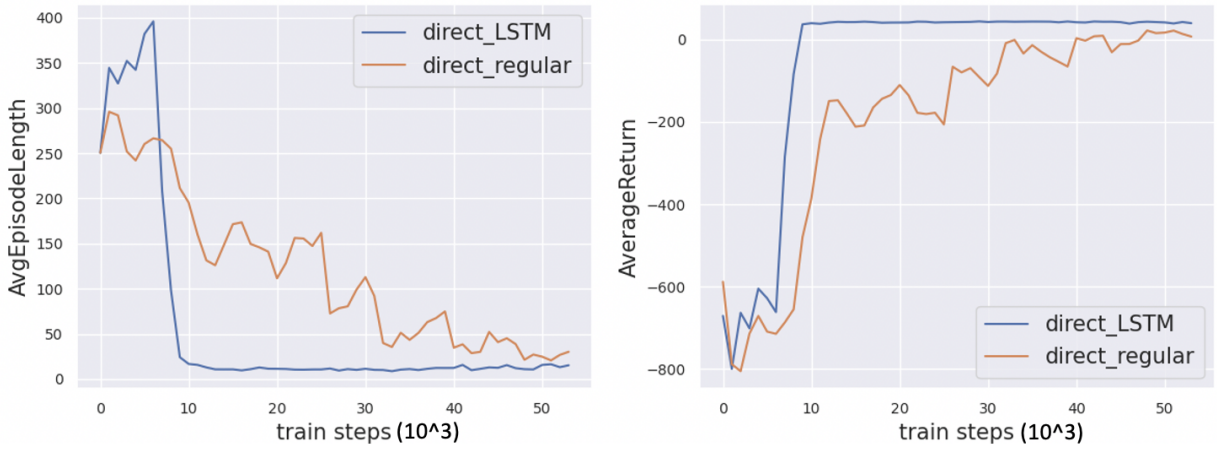


Figure 3.10: Comparison of training results for direct agent: with and without LSTM.

We also tested adding LSTM module in the critic for hierarchical control policy. We did a hyperparameter sweep that tested: LSTM_layer_number $= [2, 3, 4]$, target_update_τ $= [0.0005, 0.005, 0.01, 0.05]$, train_sequence_length $= [2, 3, 5]$, activation_function $= [\text{reLU}, \tanh]$. The hyperparameter sweeping did not find a suitable set of hyperparameters that are able to lead to training success convergence. The reason could be that for hierarchical control, there are only weak correlation between the steps since each positional control command is executed by many low level wheel velocity commands. Thus LSTM module is not able to capture the pushing dynamics well. For the section below, the hierarchical control policy

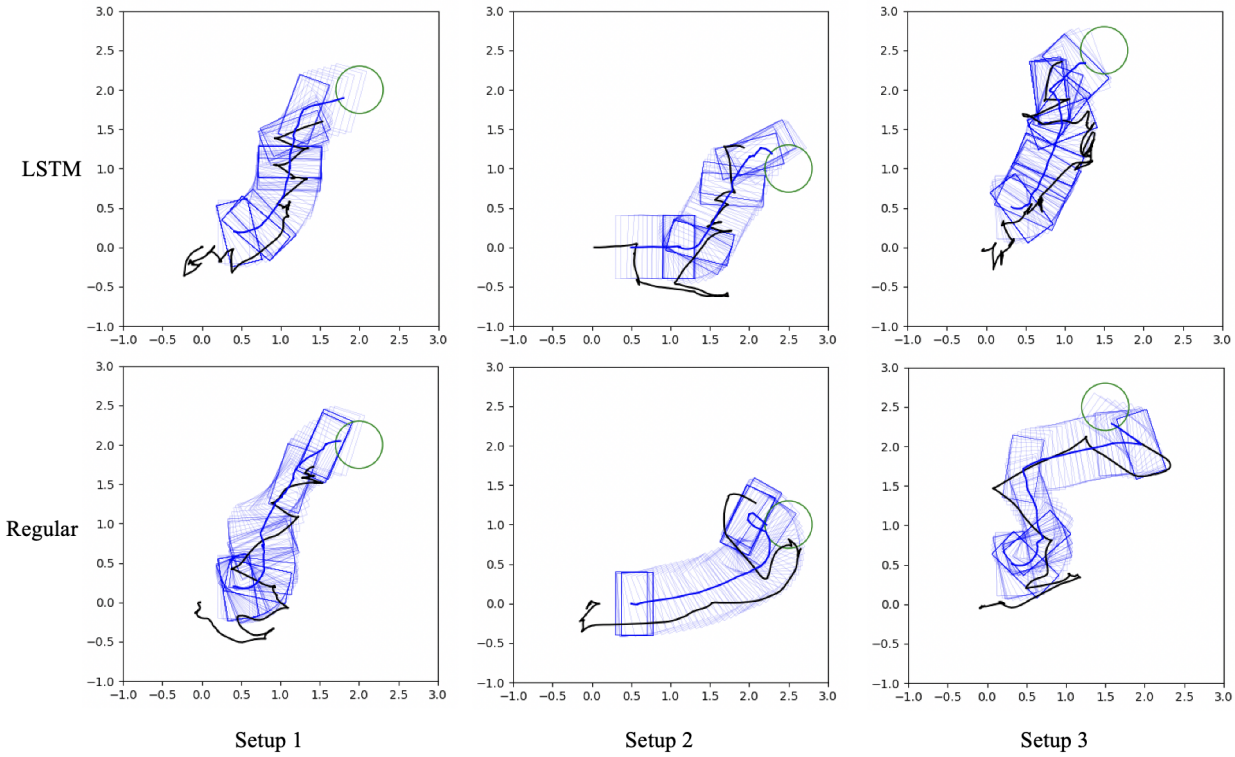


Figure 3.11: Plots of the pushing trajectories for the two agents when validated on an object with center of mass offset.

The first row shows the results of the agent trained with LSTM module, and the second row shows the results of the agent trained without LSTM. The columns corresponds to Setup 1, 2, 3 respectively.

does not utilize the LSTM module.

Generalizability Results

In this section we test the generalizability of both hierarchical and direct control policies. We train both agents with a variety of objects shown in Figure 3.12. The training curves are shown in Figure 3.14. Then we test the agent with unknown objects shown in Figure 3.13. The unknown objects are placed randomly for each episode. The push success rates for the two policies running 10000 episodes are shown in table 3.2. As can be seen in table 3.2, both hierarchical and direct control are able to generalize to unseen objects with a very high push success rate. However, for hierarchical control policy, when trained and tested on a variety of objects, episode length significantly increases, indicating it is taking more steps to plan pushing.

	Setup 1	Setup 2	Setup 3
obj start x,y position (m)	$[0.4, 0.2]$	$[0.5, 0.0]$	$[0.3, 0.5]$
target x, y position (m)	$[2.0, 2.0]$	$[2.5, 1.0]$	$[1.5, 2.5]$
Episode Time-LSTM (seconds)	11.99	9.12	13.12
Episode Time-Regular (seconds)	12.01	8.84	11.21

Table 3.1: Specifications for setups 1,2, and 3, and times each agent used to finish an episode. The table corresponds to Figure 3.11 above.

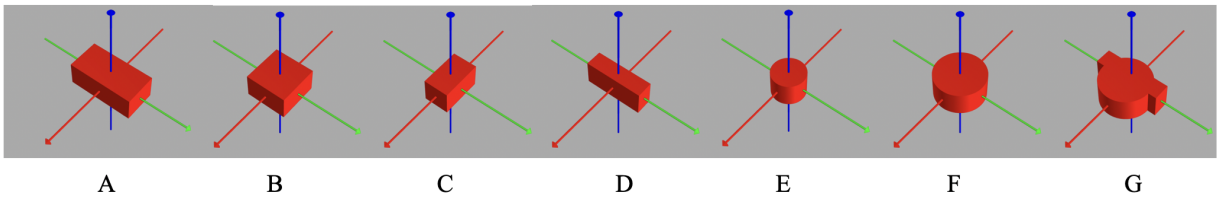


Figure 3.12: Objects A, B, C, D, E, F, G used in training. Dimensions for A to D are (L by W by H in meters): $[0.4, 0.8, 0.3], [0.5, 0.5, 0.3], [0.5, 0.3, 0.3], [0.2, 0.7, 0.3]$. Dimensions for E and F are (height and radius in meters): $[0.3, 0.2], [0.3, 0.3]$. Object G is a composite of a rectangle of size $[0.2, 0.8, 0.3]$, and a cylinder of $[0.3, 0.3]$ having an offset of $[0.1, 0.0, 0.0]$.

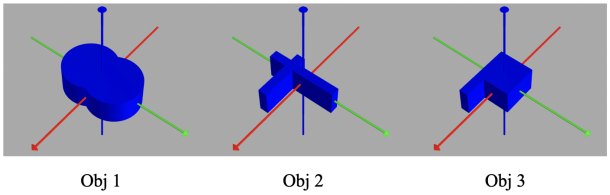


Figure 3.13: $Obj1, Obj2, Obj3$ used in testing.

3.5 Conclusion and Future Work

In this chapter, we proposed a novel soft-actor-critic(SAC) based framework to do mobile robot pushing. Our method utilizes image mask and LSTM modules to improve the model's generalizability and the ability to learn pushing dynamics. We further provided a comparison for direct and hierarchical control policies using our framework. It has been shown that hierarchical control policy explores better than the direct control. The direct control structure explores between adjust joint states, while the hierarchical control is able to jump between distant states, as long as these states are connected by the low level motion generator. In the end, we verified the generalization abilities of our control policies in simulation, reaching a result that both policies can be generalized to unseen objects with a very high success rate.

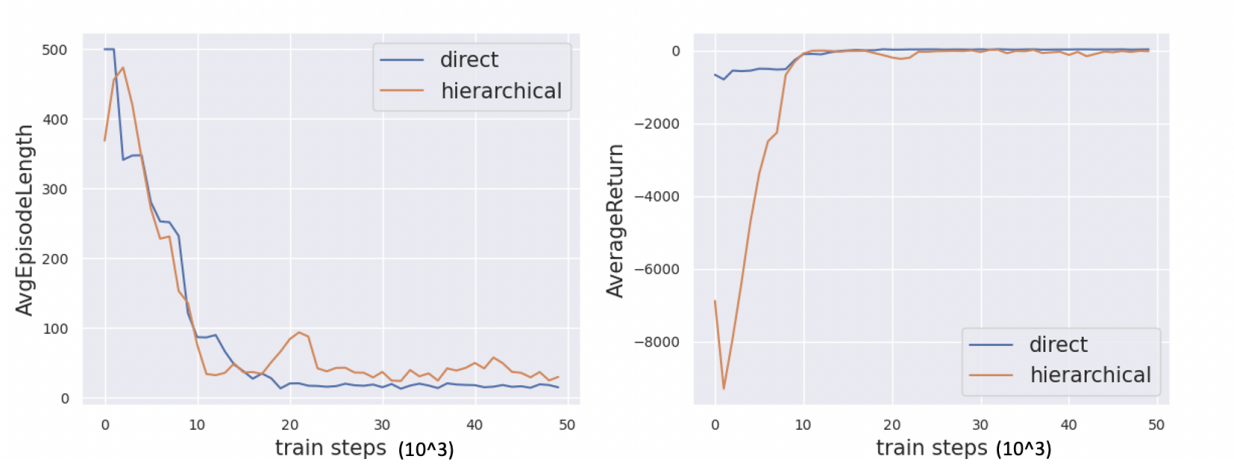


Figure 3.14: Training curves for hierarchical and direct control policies with random objects. Hierarchical control is experiencing a large initial negative return because it is exploring the robot position space.

		Single obj training	Random obj training	Test Obj 1	Test Obj 2	Test Obj 3
Hierarchical	Success Rate	-	-	0.98	0.98	0.98
	Avg. successful episode len.	3.56	30.48	22.09	35.78	24.68
Direct	Success Rate	-	-	1.00	1.00	1.00
	Avg. successful episode len.	17.01	18.03	22.53	34.02	19.19

Table 3.2: Results for hierarchical and two level control.
'Single object Training' refer to our results in section 1, where the policies are training pushing a single object. 'Random objects Training' refers to the training results in this section, where the policies are training pushing random objects. Testing objects Obj 1, Obj 2, and Obj 3 are shown in 3.13. Success rate measures the ratio of successful pushes, Avg. successful episode length measure how many steps the agent took to finish the task. Here we only measure the successful episodes. The average successful episode length for training is measured after training has converged.

Part II

Robot Trajectory Tracking Under Slippery Conditions

Chapter 4

Mobile Robot Slip Rejection: Model Predictive Tracking Control

4.1 Introduction

Over the past years, many researchers have focused on the problem of controlling nonholonomic systems. As a representative nonholonomic system, WMRs have attracted much attention. Many control methods for mobile robots are designed based on the assumption that their wheels roll without slipping, but this is often not the case. Today, mobile robots are used extensively in service areas, for planetary exploration, and for strategic applications, among other applications. Many of these applications require velocity maneuvering control in unstructured environments, which can easily induce wheel slipping. Ignoring the effect of wheel slip can cause numerous problems. The mobile robot may deviate from the desired path and the tracking control system could in some cases become unstable. Thus, accounting for slipping when controlling a mobile robot is critical.

Some studies that have investigated the problem of slip modeling and rejection have focused on estimating the exact amount of slip [40, 41, 42]. However, such works have failed to adequately improve the accuracy of trajectory tracking, mainly due to the lack of sophisticated control laws. More recent works have modeled relationships between observations and terrain/slip parameters to compensate for unmodeled dynamic effects in the kinematic model [43, 44], and one study further updated terrain fitting parameters online [45]. However, these studies have relied heavily on test-time data and the accuracy of analytical mapping found from estimated parameters to control inputs. Several other works have modeled wheel dynamics and focused on traction forces [46, 47, 48]. These models use detailed dynamic models over varying ground conditions; however, they require the real-time estimation of terrain-specific parameters and force, which are often not readily available. Finally, several works based on RL have aimed to mitigate slipping by directly outputting velocity commands [49], and modeling the dynamics with neural networks [50]. These end-to-end learning methods are often not adaptive to other scenarios or not interpretable in terms of model formulation.

Our work in this chapter [51] utilizes a hierarchical structure to conduct mobile robot trajectory tracking under slipping conditions. The upper level is an RL module while the lower level is an MPC optimization module for trajectory tracking. The high-level RL module actively adjusts the conservative level of the control scheme to mitigate the slipping problem. Our framework does not attempt to estimate terrain parameters and fit an analytical mapping to control commands as in previous works. Furthermore, it does not require test-time data to perform model fitting and achieves zero-shot adaptation to different scenarios. The control commands in our framework are calculated using the optimization problem in the MPC module. We use the RL module to decide how to adjust the parameters in the optimization problem based on the robot's current state, current slipping conditions, and trajectory information. The adjusted optimization module outputs commands that alleviate slippery conditions, while the MPC structure helps to mitigate the model–plant mismatch. Our framework does not involve wheel dynamics and only uses RL to regulate the optimization problem instead of directly outputting commands as in end-to-end RL. Therefore, it is more explainable than end-to-end RL, more adaptive to different terrain conditions, and faster to train.

The contributions of our work in this chapter are as follows: (1) We propose an online hierarchical MPC framework that actively adapts the control scheme to slippery conditions through RL; (2) we demonstrate that our framework outperforms the manually tuned best baselines by 17.5% and 12.7% for average minimum displacement error and average yaw error in simulation, respectively; and (3) we demonstrate the adaptiveness of our framework using the trained framework in various real-world scenarios without further tuning, thus closing the sim-to-real gap.

The remainder of this chapter is organized as follows: Section 4.2 provides a brief overview of works related to mobile robot trajectory tracking and control under slip conditions; Section 4.3 provides a detailed explanation of the proposed approach; Section 4.4 presents the experiments performed to validate the effectiveness and adaptiveness of our proposed framework; Section 4.5 discusses the experimental results; and Section 4.6 concludes the chapter.

4.2 Related Work

Trajectory tracking and control under slippery conditions

Methods using kinematic models

To account for slipping conditions for mobile robots, many works focus on identifying and incorporating slipping situations or terrain states into kinematic models. Some early works aim to estimate the exact amount of slip. Burke [40] proposed an adaptive approach to estimate slip solely based on angular velocity. Other works [41, 42] utilize optical flow sensors, GPS, and multiple sensing equipment fusions to accurately estimate slip. Even though these works can achieve high slip estimation accuracy, they design simple control

laws and ignore model-plant mismatch, resulting in inadequate trajectory tracking accuracy improvement.

More recent works focus on using additional parameters in the kinematic model to account for slipping, such as the effective wheelbase model [52] and the general kinematic slip model [53]. [43] also proposed a fitting method to find relationships between track length and yaw rate, which could be used to predict and compensate the unmodelled dynamic effects of the kinematic model. Drawbacks of these methods include high dependency on experimental data and no online tuning ability. In [44], a motion planning algorithm is developed to plan the trajectories that avoid slipping conditions in advance. However, they solve a nonlinear optimization problem to find four parameters in order to fit the wheel to terrain at each timestep, which could be slow and unstable. To adapt to dynamic and evolving situations, [45] developed online parameter estimation methods to aid their terramechanics equations, and [54, 55] proposed an instantaneous center of rotation (ICR)-based tracked robot model. However, the first line of work requires torque and sinkage parameters to be known and the second suffers from ICR saturation problems with small longitudinal slip. Some other works choose to model slip as disturbance by using observers and augmenting state space representation [56, 57, 58]. However, this line of work contains state vectors of high orders and massive matrix inverse calculations, also their control law is still limited to fixed forms.

Methods using dynamic models

In order to improve robots' maneuverability in real environments, many works choose to model wheel dynamics and focus on traction forces. [46] was one of the earliest works where slip was considered in the WMR dynamic model. The authors consider traction force being linearly dependent to small values of slip ratios. Other works use dynamic models for slipping prevention as in [47, 48, 59, 60, 61]. In [47], lateral slip dynamics are directly included in dynamics modeling using a Lagrangian approach. [48] considers both longitudinal and lateral slipping and designs a time-invariant discontinuous feedback law to stabilize the system. Both works utilize the Magic formula [62] to derive the relationship between traction force and slip ratio, which requires ground-specific coefficients to be determined by extensive experiments. [60] presents a traction and braking model for wheel slipping detection to calculate dynamics in an extended Kalman filter framework. In [59], a dynamic model is used for omnidirectional wheeled mobile robots. In [63], both kinematic and dynamic models are used and slipping prevention is achieved via disturbance rejection. Although dynamic models consider forces in addition to kinematics, using them for slipping rejection requires system identification for different scenarios and cannot generalize to varied situations.

Reinforcement Learning

With the recent advances in deep learning, RL methods have gained increased popularity in autonomous system control and trajectory tracking. Most of these methods implement an end-to-end learning pipeline. In [64], a high speed drift controller built on model-free deep RL

algorithm soft actor-critic is presented. [49], a RL algorithm is developed for position control of wheeled robots to allow automatic learning and reduced need for real robot learning. [65] uses a model-free deep RL method based on Deep Deterministic Policy Gradient (DDPG) for trajectory tracking control. In [50], a neural tracking algorithm based on RL with explicit neural networks approximating skidding and slipping items. In addition, there have been efforts in combining deep RL algorithms with classical control methods. For example, [66] uses Q-learning to directly add correctional inputs to PID commands. Although the end-to-end RL methods and methods combing RL and PID are yielding promising results, there are a lot of problems with them. The end-to-end RL approaches require considerable training time and are not adaptive to different situations. Combining PID and RL poses significant challenges in explainability and the PID module requires nontrivial tuning that profoundly impacts the performance.

Online MPC tuning

In recent years, Model Predictive Control (MPC) has become the prominent control method because of its ability to consider future horizons and state constraints. For example, [67] introduces a slip control method with safety constraints based on nonlinear MPC, and [68] presents an MPC-based slip ratio controller that considers road roughness. Both methods require dynamic model parameters to be known, have heavy computational time and no real-world experiments were presented. There also exist some works on tuning MPC with RL on robot tracking control [69, 70]. However, these works fail to consider the changes of ground conditions, and only parameters in the cost function are tuned offline. Our work, on the other hand, tunes the MPC online through RL to account for slip conditions.

4.3 Methodology

Problem Overview

Fig.4.1 illustrates our trajectory tracking problem setup. The mobile robot is required to travel from position $\mathbf{x}_{start}$, following a predefined trajectory (shown in red) to reach the end of trajectory point $\mathbf{x}_{end}$. Here we define a series of terms to describe the robot motion and the tracking task status, also shown in Fig. 4.1. First a look ahead point P is defined, which is d_p distance apart in robot x direction from the center of gravity (CG) O of the robot. We use minimum displacement errors, e and e_p, and yaw errors, $\Delta\theta$ and $\Delta\theta_p$, to represent the tracking errors relative to the reference trajectory. e and $\Delta\theta$ are defined with respect to O, and e_p, and $\Delta\theta_p$ are defined at the look ahead point P, which gives the robot preview abilities of the trajectory. Minimum displacement error e is the shortest distance between O and the trajectory, yaw error $\Delta\theta$ is the difference between the robot's current yaw θ and the corresponding closest reference trajectory point's yaw θ_{ref}. e_p and $\Delta\theta_p$ are defined similarly, with $\theta_p = \theta$. $v_{\parallel}$ and $v_{\perp}$ are robot's speed along and perpendicular to the reference curve:

$v_{\parallel} = v \cdot cos(\Delta\theta)$ and $v_{\perp} = v \cdot sin(\Delta\theta)$. Then $v_{\parallel diff}$ is defined to be the difference between reference speed v_{ref} and $v_{\parallel}$.

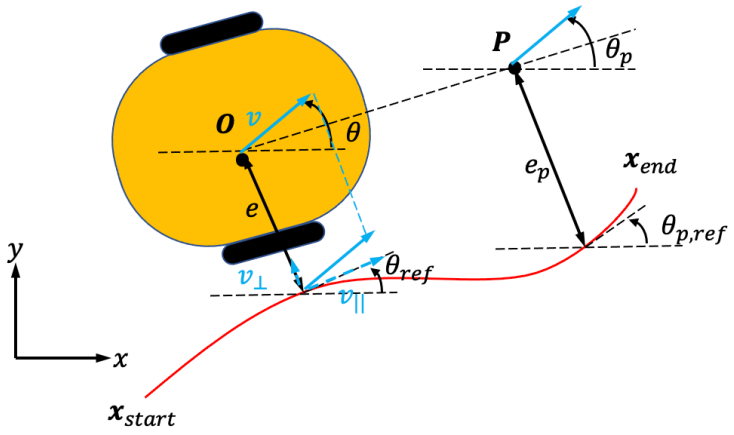


Figure 4.1: Schematic diagram of the problem setup.

The tracking task can be formulated as a time-discrete Markov Decision Process (MDP) defined by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space; $\mathcal{A}$ is the action space; $\mathcal{T}(s^{'}|s, a)$ is the state transition model; $\mathcal{R}(s, a)$ is the reward function; $\gamma \in [0, 1)$ is the discount factor. We learn a policy $\pi(a|S)$ conditioned on state $s \in \mathcal{S}$. Therefore, the agent following the policy π obtains an observation s_t at time t and performs an action a_t, receiving from the environment an immediate reward R_t and a new observation s_{t+1}. The goal of RL is to learn an action selection policy π that maximizes the expectation of discounted sum of rewards $E[\sum_{t=1}^{T} \gamma^{t-1} R_t]$.

The state S consists of 10 variables: e, $\Delta\theta$, e_p, $\Delta\theta_p$, $v_{\parallel diff}$, $v_{\perp}$, v_{actual}, ω_{actual}, v_{input}, and ω_{input}. The first six variables are as described before. v_{actual} and ω_{actual} are the robot's actual linear and angular velocities, and v_{input} and ω_{input} are the linear and angular velocity commands that are executed by the robot in the previous step. Considering the slipping conditions, the actual and input velocities can differ much, since the velocity commands are polluted by sliding between the wheel and the ground, and cannot transfer to actual velocities properly. In general, The objective of the problem is to come up with a control policy that takes as input the observable states and guides the robot to decide when and how much to adjust the acceleration bound, based on slipping, local trajectory information, and tracking errors. The states are selected as such because they are good local representations of the scene, and that they can be observed directly using the robot's onboard perception modules.

We define action A to be $\mathbf{u} = [v, \omega]$, which are linear and angular velocity commands to the robot. The low level PD controller takes in these kinematic level commands and map them to the dynamic level wheel torque through regulating the current input. The reward for each step is defined in Eqn. 5.3. Here we penalize e, $\Delta\theta$ and number of steps taken, with

coefficients R_{dist}, R_{ang} and R_{step}. The cumulative reward is defined as $\sum_{t=1}^{T} \gamma^{t-1} R_t$, where R_t is the step reward.

$$R_t(s_t, a_t) = R_{dist} \cdot e^2 + R_{ang} \cdot \Delta\theta^2 + R_{step} \cdot n_{steps} \tag{4.1}$$

Proposed Framework

The proposed framework consists of a deep RL-based high-level module, an MPC optimization module, a low-level tracking controller, and the robot. The framework is visualized in Fig. 4.2. The RL module takes observed robot states **obs**; reference trajectory $\mathbf{x_{ref}}$; and previous command **u** to formulate state S, and it outputs linear velocity acceleration bound **a**. The MPC optimization module takes **a** along with **obs**, $\mathbf{x_{ref}}$, and previous command **u**, and calculates the next step command **u**. The low level controller then executes the command on the robot and feeds the observed states back into the RL module to calculate for step rewards, and the policy is updated accordingly. The loop stops when the positional error of the robot and final goal position is within a threshold. The entire framework realizes our MDP formulation, and is trained end-to-end using an RL algorithm. Notice the MPC optimization module alone with a fixed acceleration bound is ideal for trajectory tracking without slipping. However, when slipping happens, sudden changes in velocity commands could lead to large forces on the wheels($F = m \cdot a$), which could not mitigate the slipping condition and could lead to further trajectory deviations. Our framework, on the other hand, detects the slip conditions, and is able to make adjustments to the robot's velocity commands **u** through updating acceleration bounds online, thus reducing tracking errors.

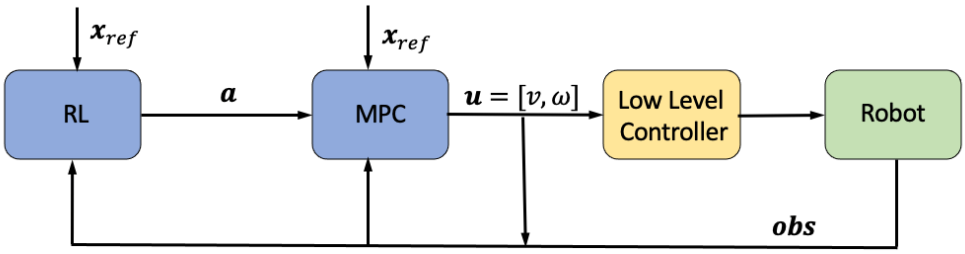


Figure 4.2: Proposed Framework.
The RL module adjusts linear acceleration bound **a** of the MPC optimization problem online at every timestep. $\mathbf{obs} = [x, y, \theta, v_x, v_y, \omega]$, which are robot's position, linear and angular velocities; $\mathbf{x_{ref}}$ is the reference trajectory; $\mathbf{u} = [v, \omega]$ are linear and angular velocity commands.

Reinforcement Learning Module

The RL module consists of actor and critic neural networks, and is used to output acceleration bounds. During training, the entire framework in Fig. 4.2 is trained with Soft Actor Critic

(SAC), an off-policy deep RL algorithm [71].

MPC Optimization Module

We describe the linear MPC optimization problem below. Here $\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x_r}$ represents the error with respect to the reference trajectory, with $\mathbf{x} = [x, y, \theta]$ being robot's position and yaw angle. $\tilde{\mathbf{u}} = \mathbf{u} - \mathbf{u_r}$ is the associated error control input, with $\mathbf{u} = [v, \omega]$ being robot's linear and angular velocities. The problem is defined at every timestep k, with a horizon H. The objective penalizes the state and input errors, as well as sudden change in input. The constraints include the discrete-time linear system error model (Eqn. 4.3), with A and B defined in Eqns. 4.5 and 4.6 according to local system linearization. Here T is the sampling period, $v_{r,k}$ and $\theta_{r,k}$ are reference speed and yaw at time k. The problem is also subject to acceleration constraints as shown in Eqn. 4.4. Notice v_{ref} is a fixed number throughout the trajectory. Thus the linear acceleration of the robot from time k to $k+H$ is bounded by a_l and a_u.

Here all the parameters are fixed except lower and upper linear acceleration bounds a_l and a_u. In our framework, the bounds are acquired from the RL module and updated at every timestep. Because although the control input is on the kinematic level, we had the assumption that the kinematic commands could influence the low level dynamics command.

$$\begin{aligned}\min_{\tilde{\mathbf{x}},\tilde{\mathbf{u}}} \quad & (\tilde{\mathbf{x}}_{k+H})^T Q_f(\tilde{\mathbf{x}}_{k+H}) + \\ & \sum_{k}^{k+H-1} [(\tilde{\mathbf{x}}_t)^T Q(\tilde{\mathbf{x}}_t) + (\tilde{\mathbf{u}}_t)^T R(\tilde{\mathbf{u}}_t)] + \\ & (\tilde{\mathbf{u}}_{t+1} - \tilde{\mathbf{u}}_t)^T R_d(\tilde{\mathbf{u}}_{t+1} - \tilde{\mathbf{u}}_t)\end{aligned} \tag{4.2}$$

$$\text{s.t.} \quad \tilde{\mathbf{x}}_{k+i+1} = A_{k+i}\tilde{\mathbf{x}}_{k+i} + B_{k+i}\tilde{\mathbf{u}}_{k+i},\ i = 0...H-1 \tag{4.3}$$

$$\begin{bmatrix} a_l \\ -1.82 \end{bmatrix} \leq (\tilde{\mathbf{u}}_{k+i+1} - \tilde{\mathbf{u}}_{k+i})/T \leq \begin{bmatrix} a_u \\ 1.82 \end{bmatrix},\ i = -1...H-1 \tag{4.4}$$

Eqn.(2)-(4). The optimization problem.

$$A_k = \begin{bmatrix} 1 & 0 & -v_{r,k}\ sin\theta_{r,k}\ T \\ 0 & 1 & v_{r,k}\ cos\theta_{r,k}\ T \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

$$B_k = \begin{bmatrix} cos\theta_{r,k}\ T & 0 \\ sin\theta_{r,k}\ T & 0 \\ 0 & T \end{bmatrix} \tag{4.6}$$

Eqn.(5)-(6). A, B matrix definition.

4.4 Experiments

Our experiments were designed and conducted in order to answer the following questions:

1. Could robust and efficient performance be achieved by manually tuning the parameters of the MPC module?
2. Is it possible to leverage RL to learn an adaptive control scheme that works better than fixed parameters?
3. How adaptive is our proposed online MPC tracking control framework under different conditions?

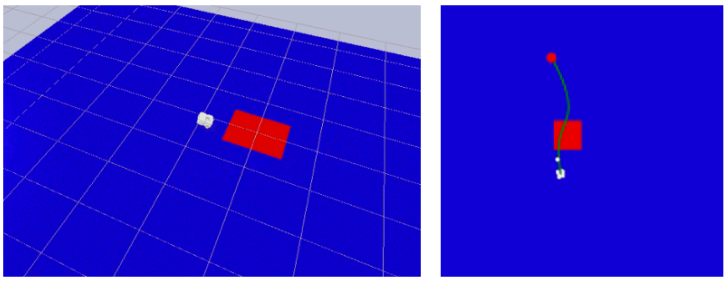
To answer these questions, we carried out simulated and real world experiments. The simulated experiments are conducted using PyBullet [72] with a TurtleBot waffle-pi model. The real-world experiments are conducted with a real TurtleBot waffle-pi. The framework is trained in simulation only. In order to show our framework's adaptiveness, the trained framework is directly applied to the two sets of experiments, which differ in command frequency and terrain conditions. The metrics we consider are the average of minimum displacement error over the entire trajectory: $\overline{e}$, and average yaw error over the entire trajectory: $\overline{\Delta\theta}$.

$$\overline{e} = \frac{1}{T_{traj}} \sum_{i=1}^{T_{traj}} e_i, \quad \overline{\Delta\theta} = \frac{1}{T_{traj}} \sum_{i=1}^{T_{traj}} \Delta\theta, \tag{4.7}$$

To validate our framework, we create two baselines with the same framework in Fig. 4.2, but without the RL module. Instead of varying the acceleration bounds online using RL, the two baselines just use fixed bounds throughout the timeframe. The first baseline is named **default** and uses $a_l = -2.0 m/s^2, a_u = 2.0\ m/s^2$, which are the default highest acceleration bounds. The second baseline is named **manual**, which uses the best acceleration bound found by parameter sweeping among the possible bound values. Our framework is denoted as **online**.

Simulation

The environment setup with a sample trajectory is shown in Fig. 4.3. Here blue indicates high frictional areas (frictional coefficient μ=0.9), and red indicates low frictional areas (μ=0.01). The red patch size is 1 m by 1 m, and is located at $[5, 5]$ in m. The trajectory is generated using a spline planner as described in [73]. The planner takes a given n number of 2D points and generates a smooth curve as shown in green in Fig. 4.3b. For all simulation experiments, the robot is run at 10 Hz. Reference speed is 0.4 m/s.



a. b.

Figure 4.3: Simulation rendering.
(a) An actual rendering of the simulation. b) bird-view image with annotations. The big white dot in 4.3b represents the robot, and the small white dot indicates the current look ahead point P. Here we set the x-axis distance of P to the robot's CG to be 0.5 m. The red dot indicates the final desired destination.

For MPC, the horizon T is set to be 3, $Q = Q_f = \text{diag}[50, 50, 1]$, $R = R_d = \text{diag}[1, 0.001]$. For training, random trajectories are generated for each episode. The five points that are given to the spline planner are generated from uniform distributions (Eqn. 4.8). We use SAC [71] algorithm to train the entire framework in Fig. 4.2. In SAC, both the policy network and the value network are two layer fully-connected neural networks of size 64. The learning rate is set to be 0.0003, γ is 0.99. The coefficients R_{dist}, R_{ang} and R_{step} are set to be 20, 1, 0.001, respectively. The entire framework is trained until convergence.

$$\begin{aligned}(x_1, y_1) &= [U(3.5, 4.0), U(5.0, 5.5)] \\ (x_2, y_2) &= [U(4.25, 5.25), U(4.5, 6.0)] \\ (x_3, y_3) &= [U(5.25, 6.25), U(4.5, 6.0)] \\ (x_4, y_4) &= [U(6.25, 7.25), U(4.5, 6.0)] \\ (x_5, y_5) &= [U(7.25, 8.25), U(4.5, 6.0)]\end{aligned} \tag{4.8}$$

Effects of varying acceleration bounds

To answer the first question, we design two sets of experiments to illustrate effects of fixing linear and angular bounds to different values. The setup is shown in Fig. 4.6 scenario a.

We swept over $a = [0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3, 1.5, 1.7, 1.9] m/s^2$, for linear acceleration bounds. a here is used for both lower and upper bound values for consistency. ($a_l = -a, a_u = a$). For each a, 10 trials were run. The mean and standard deviation values(std) for $\overline{e}$ and $\overline{\Delta\theta}$ for all trials are plotted. The line represents the mean and the shade represents the std. value in Fig. 4.4 and 4.5. Fig. 4.4 shows that an appropriate linear acceleration bound can be found to achieve minimum distance and yaw errors. For angular

acceleration bounds, we fix a to be 2.0 m/s^2, which is the maximum possible value, and swept over angular acceleration bound values $\alpha = [0.3, 0.6, 0.9, 1.2, 1.5, 1.8]$ rad/s^2. Notice the value α is again used for both lower and upper angular bounds. Fig. 4.5 indicates as angular bound increases, the robot has more freedom to adjust its angular position, thus reducing positional errors. Therefore, we choose to set the angular acceleration bound to be the largest possible number 1.82 rad/s^2, and choose to tune the linear acceleration bound online through a RL module, since a certain best bound exists at every time step.

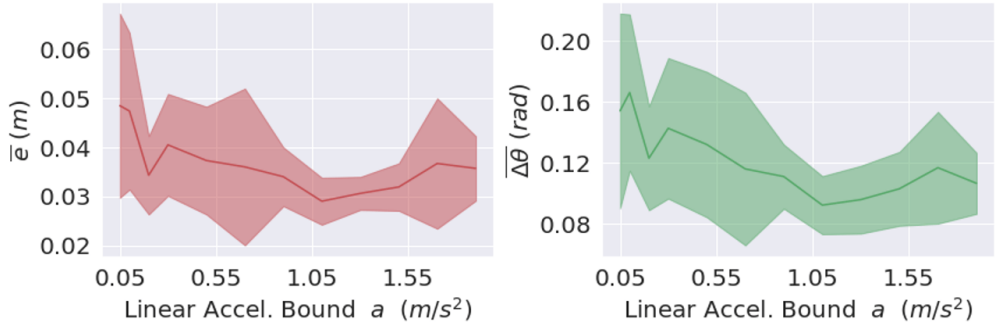


Figure 4.4: Effects of varying linear acceleration bound.
If the linear acceleration bound value is too small, $\overline{e}$ and $\overline{\Delta\theta}$ both increase, and increasing the bounds above 1.1 m/s^2 does not improve the performance. Thus, appropriate linear acceleration bound can be found to achieve minimum $\overline{e}$ and $\overline{\Delta\theta}$.

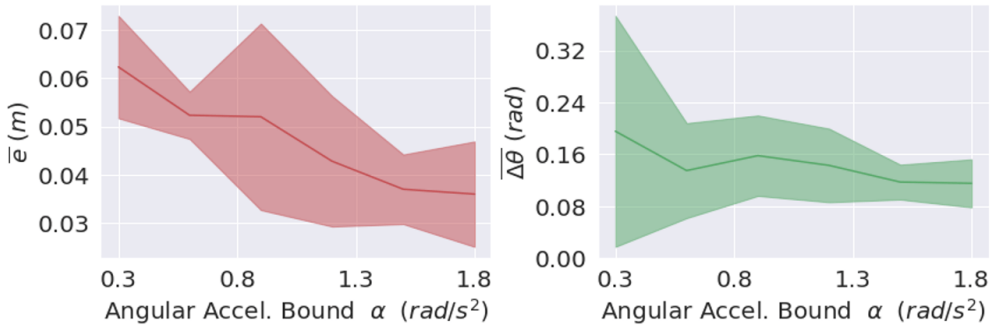


Figure 4.5: Effects of varying angular acceleration bound.
Increasing α helps to reduce $\overline{e}$ and $\overline{\Delta\theta}$.

Framework Validation in Simulation

To answer the second question, we verify our framework in simulation with three scenarios shown in Fig. 4.6. The points used in trajectory generation are listed in table 4.1.

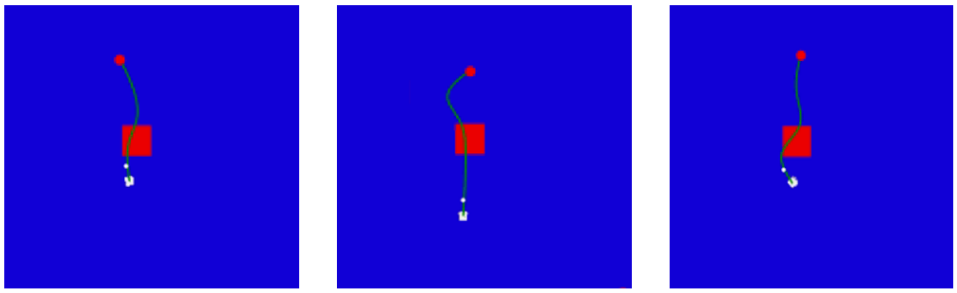


Figure 4.6: Three randomly generated simulation scenarios used for framework validation.

Scenario	Points used in trajectory generation
a	[3.76, 5.24], [4.99, 5.26], [5.81, 4.99], [6.64, 5.07], [7.75, 5.56]
b	[2.57, 5.22], [4.64, 5.15], [5.64, 5.33], [6.66, 5.74], [7.30, 5.02]
c	[3.74, 5.16], [4.66, 5.51], [5.64, 4.88], [6.66, 5.00], [7.90, 4.86]

Table 4.1: Points used in trajectory generation in Fig. 4.6

Improvement	Scenario a		Scenario b		Scenario c	
	$\overline{e}$	$\overline{\Delta\theta}$	$\overline{e}$	$\overline{\Delta\theta}$	$\overline{e}$	$\overline{\Delta\theta}$
To **default**(%)	29.4	21.2	20.7	28.6	19.2	23.5
To **manual**(%)	20.4	13.9	22.8	19.3	9.4	4.9

Table 4.2: Improvement of **online** compared to the two baselines.

For each scenario, 20 trials were run for our framework (**online**) and the two baselines (**default**, **manual**), respectively. **online** actively adjusts a at each timestep, **default** uses $a = 2.0\ m/s^2$, and **manual** uses $a = 1.1\ m/s^2$. Since the ground setup is the same as discussed in the previous section, $a = 1.1\ m/s^2$ is the best manually found bound value by parameter sweeping. The results are shown in Fig. 4.7. The black line indicates standard deviation. **online** achieves the best performance in all scenarios for both $\overline{e}$ and $\overline{\Delta\theta}$. Notice the std for **online** is also smaller compared to the other two. Quantitative results are presented in table 4.2.

To visualize how the bound value adapts online, we plot the bound value outputs versus

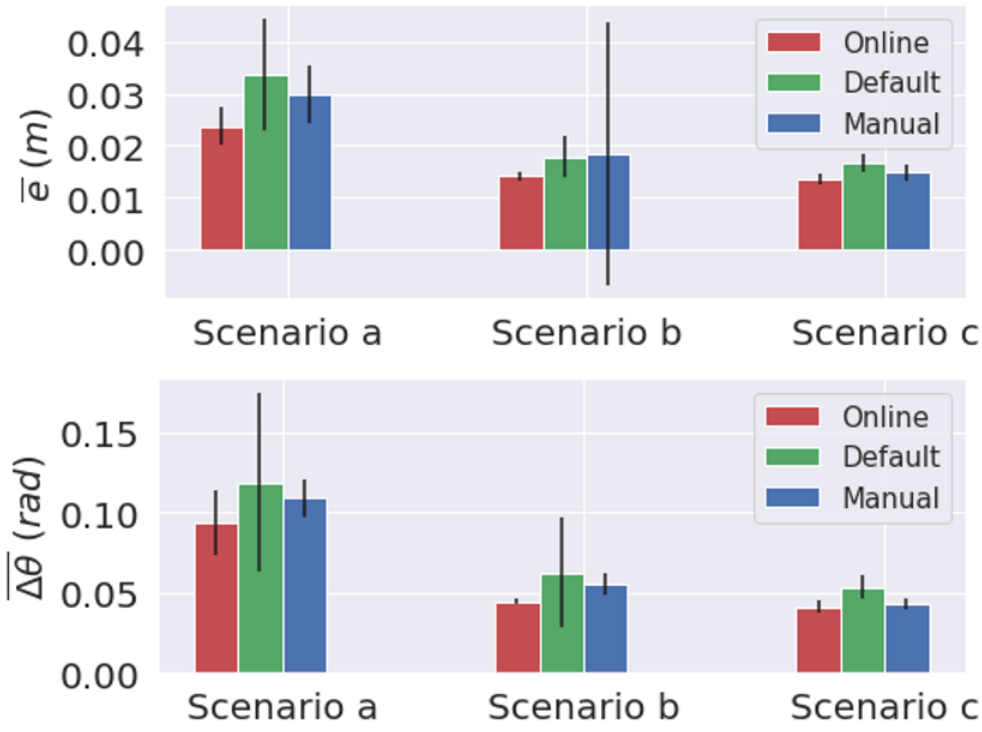


Figure 4.7: Comparison of three frameworks.
Our framework **online** achieves the best performance in all scenarios for both $\overline{e}$ and $\overline{\Delta\theta}$.

when the robot is on the low friction patch, for scenario a. shown in Fig. 4.8, when the robot enters the low friction patch, the upper acceleration bound a_u decreases to limit the robot from sudden acceleration, and the lower bound a_l increases to allow for quick speed decrease to prevent further deviation. We also plot the robot's actual linear and angular velocities versus its velocity inputs. We observe a clear discrepancy between the actual velocities and velocity input commands, which indicates the occurrence of slipping when the robot enters the low friction patch.

We visualize how e, $\Delta\theta$ and velocity inputs change throughout the timeframe for the three frameworks. We compare trials run on scenario a. Fig. 4.9 indicates that **online** has the lowest e and $\Delta\theta$ peaks throughout the trajectory. Also, Fig. 4.9c and d proves that actively updating bound values can regulate the velocity inputs, as the linear velocity inputs for **online** fluctuate less. This proves that our proposed online framework is actively adjusting the velocities through regulating acceleration bounds, so to achieve a more stable performance.

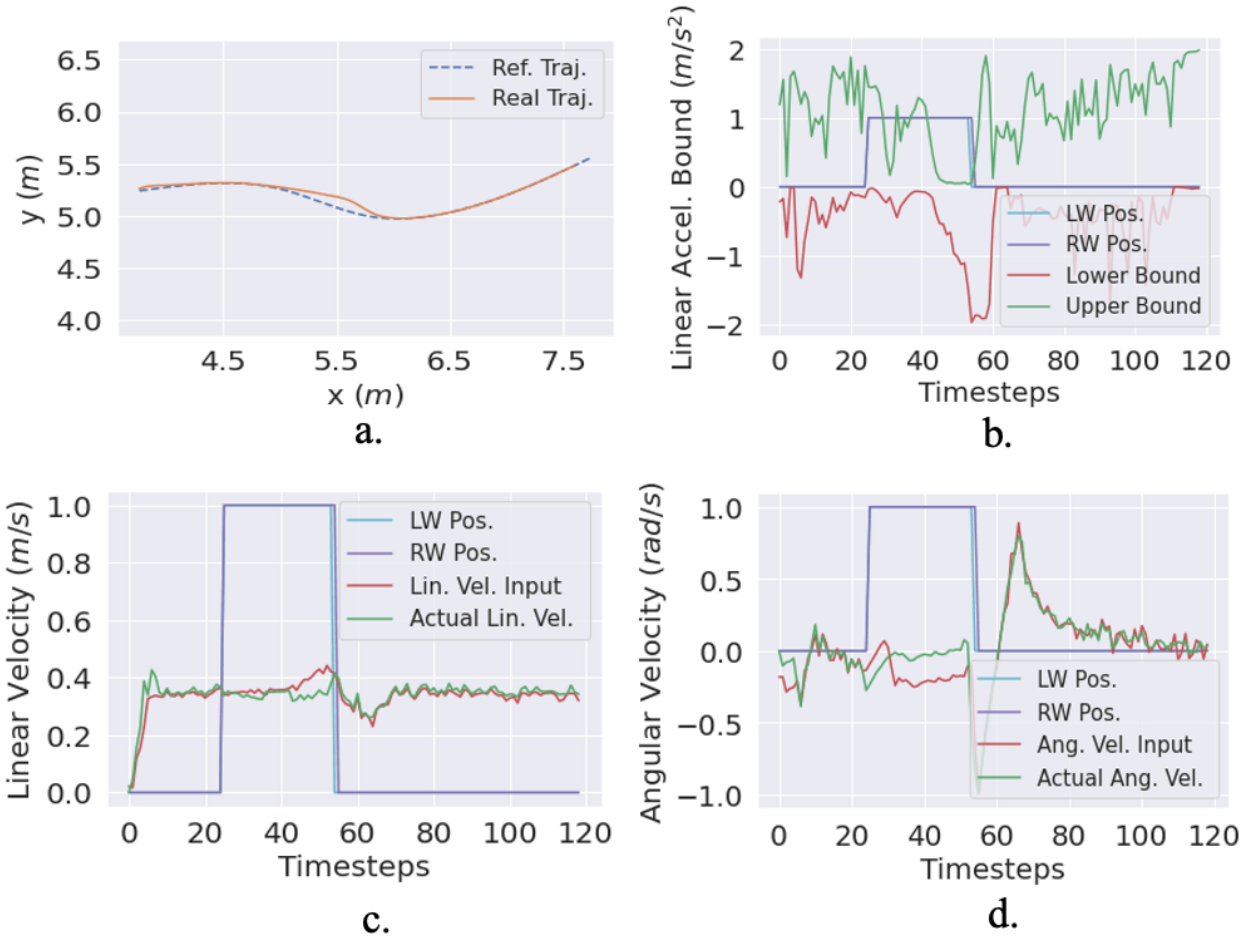


Figure 4.8: Visualization of bound output changes for scenario a, for our online framework. a) reference trajectory vs. actual trajectory. b) wheel positions vs. bound outputs. LW Pos. and RL Pos. mean left wheel and right wheel positions, respectively. Value 1 and 0 indicate the wheel is on/off the low friction patch. c) wheel positions vs. linear velocity output and actual linear velocity. d) wheel positions vs. angular velocity output and actual angular velocity. The RL module adjusts linear acceleration bounds accordingly when slipping and trajectory deviation are detected.

Real-world Experiments

To answer the third question, we directly apply a trained **online** framework onto a real-world TurtleBot, without fine tuning. The TurtleBot is run at 5 Hz, which is different from 10Hz in the previous section. Reference speed is changed from 0.4 m/s to 0.2 m/s. The ground friction coefficients are also different. According to [74], μ for wood is approximately 0.6.

The experiment setup is shown in Fig. 4.10. To induce slipping better, we choose to

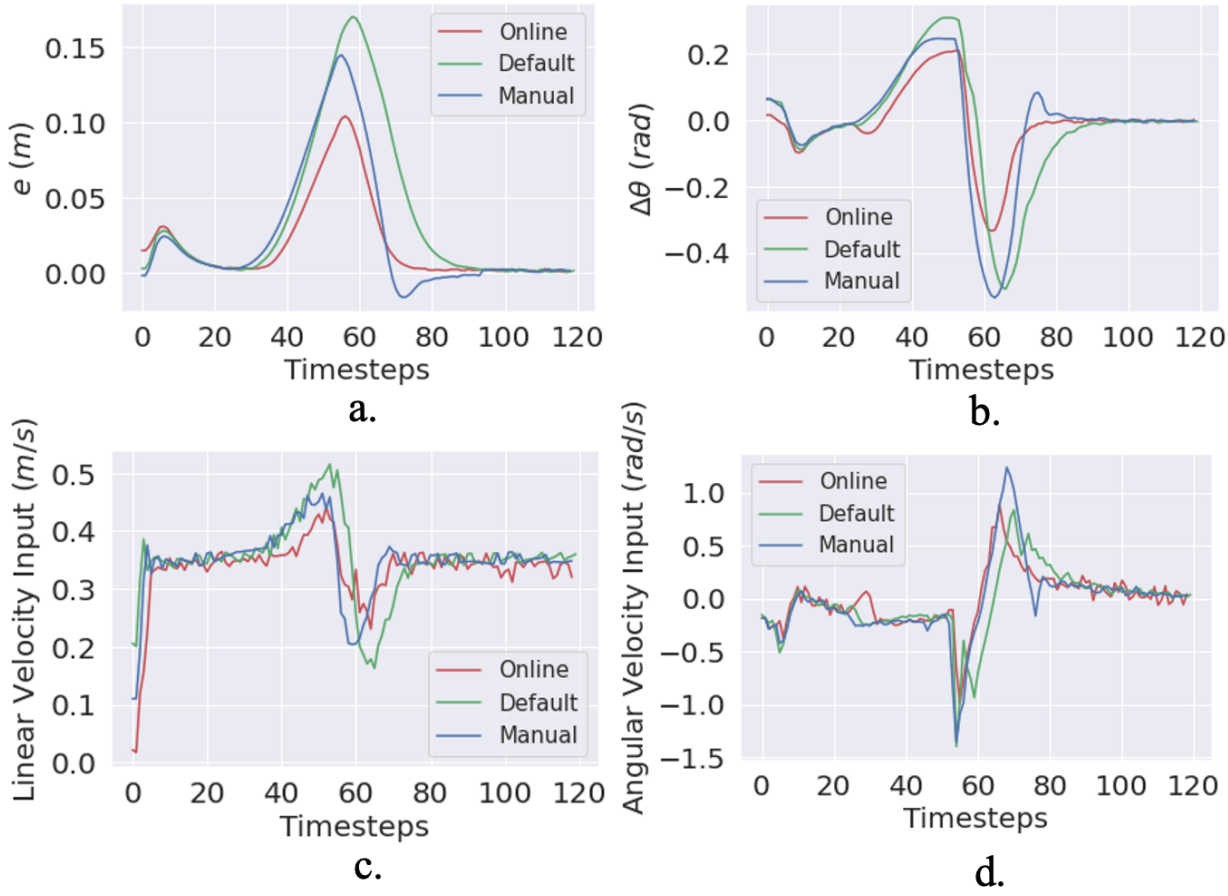


Figure 4.9: Performance comparison of online, manual, and default frameworks for scenario a.

online has the lowest e and $\Delta\theta$ peaks throughout the trajectory.

manually perturb the robot so to achieve a similar slipping effect. The manual perturbation is done by slowly moving the white cardboard sheet from position a to position b, as shown in Fig. 4.10b and c. The action is initiated when the robot's wheel first touches the sheet, and the entire action is finished within four seconds for each trial. The robot's initial position is shown in Fig. 4.10a, and the black curve indicates the reference trajectory.

In order to find the best linear acceleration bound for **manual** for this scenario, we did parameter sweeping again for $a = [0.05, 0.1, 0.3, 0.5, 0.7, 0.9]$. For each a, we run five trials and plot the mean and std for $\overline{e}$ and $\overline{\Delta\theta}$. As can be seen in Fig. 4.11, $a = 0.3\ m/s^2$ achieves the best performance, and this value is used for **manual** framework.

We again run five trials for each of the three frameworks and the results are shown in Fig. 4.12. **online** achieves the best performance for both $\overline{e}$ and $\overline{\Delta\theta}$. Also, **online** has the smallest std value among the three, indicating it has the most stable performance. To visualize how

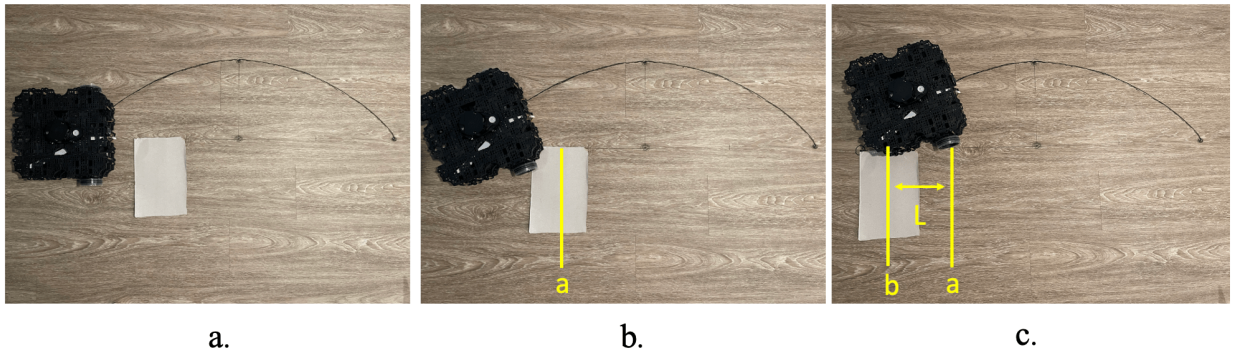


Figure 4.10: TurtleBot (black) driving on wood surface under slippery condition. The three points used in the spline planner to generate the trajectory are: $[0.0, 0.0][0.508, 0.254][1.016, 0.0]$. The cardboard size is 16 by 26 cm, and distance l is 16 cm.

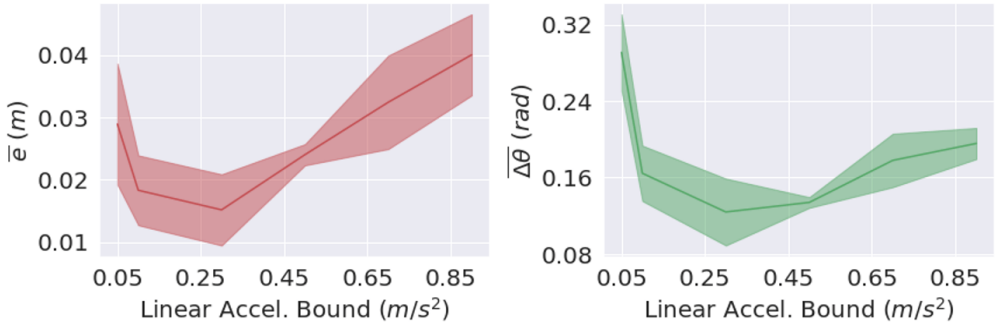


Figure 4.11: Effects of varying linear acceleration bound for real-world experiments. Best bound found is 0.3 m/s^2.

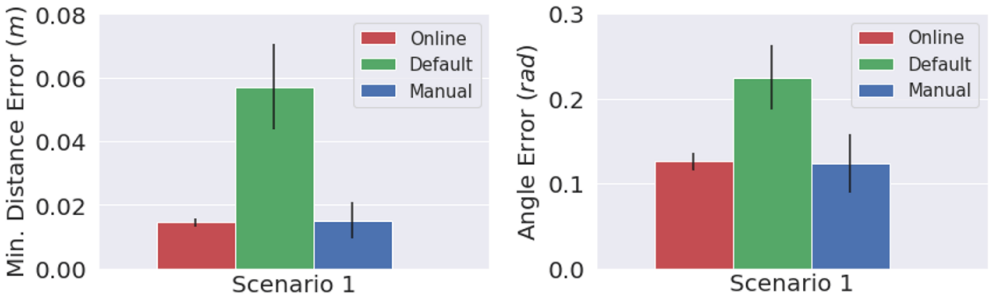


Figure 4.12: Comparison of three frameworks. Our framework **online** achieves the best performance for both $\overline{e}$ and $\overline{\Delta\theta}$.

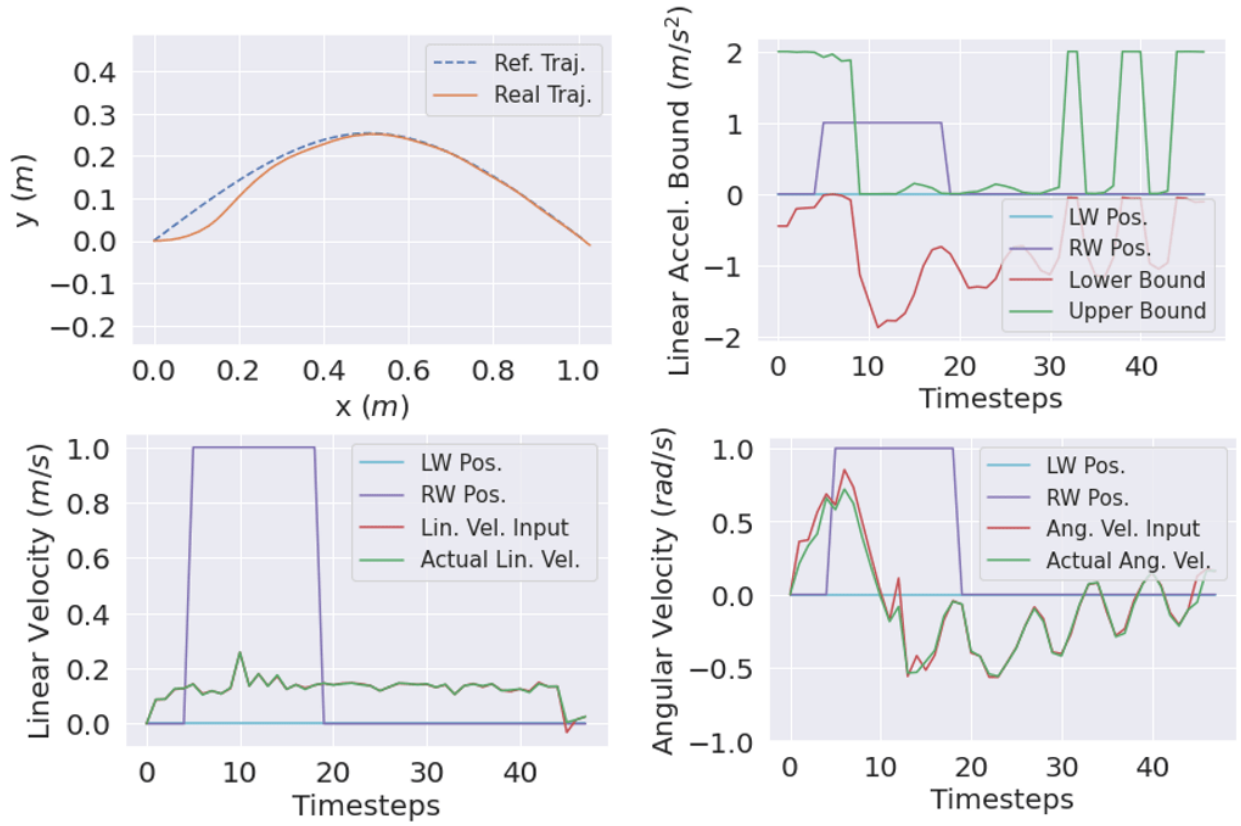


Figure 4.13: Visualization of bound output changes for real-world scenario, for our online framework.

The linear acceleration bounds are adjusted accordingly online.

the bound value changes online, we again plot the bound value outputs vs. when the robot is on the low friction patch. The results are similar to those in simulations. Fig. 4.13 shows the bound changes when the robot enters the low friction patch. Notice the change in the upper acceleration bound is steeper in the real-world case. Fig. 4.14 shows changes in e, $\Delta\theta$ and velocity inputs throughout the timeframe in the real-world case for the three frameworks. **online** and **manual** achieve similar performance for e and $\Delta\theta$, but **manual** requires offline tuning and our **online** framework does not.

4.5 Discussion

From the simulation and real-world experiments, we show the proposed online hierarchical MPC framework can adjust the bounds online based on the terrain and trajectory conditions, and the adjusted bounds lead to more stable linear velocity commands. Our framework also achieves the best performance compared to the two baselines both in simulation and real-

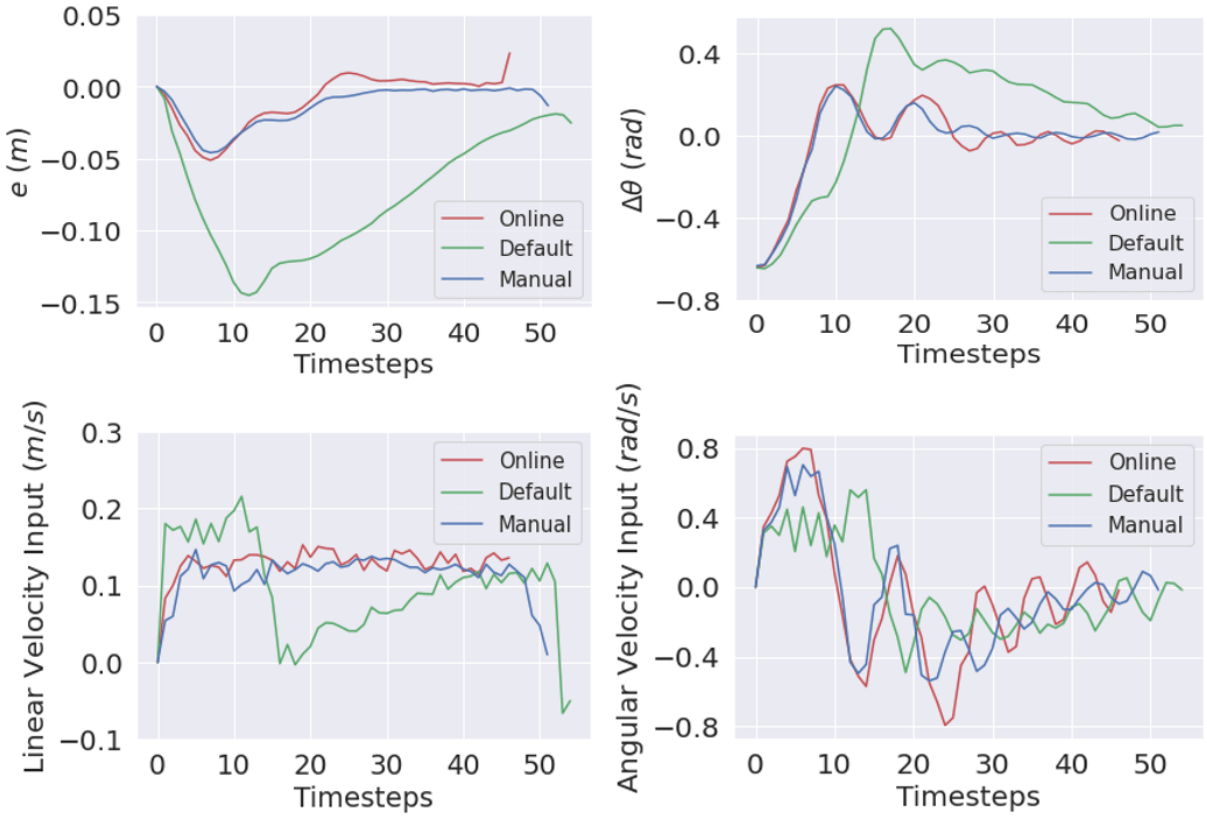


Figure 4.14: Performance comparison of online, manual, and default frameworks for real-world scenario.

online and **manual** achieve similar performance for e and $\Delta\theta$, but **manual** requires extensive offline tuning and our **online** framework was used in a zero-shot manner.

world. $\bar{e}$ improved by 23.1% compared to **default** and 17.5% compared to **manual** for three simulation scenarios. $\overline{\Delta\theta}$ improved by 24.4% compared to **default** and 12.7% compared to **manual** for three simulation scenarios. Also, the fluctuation for e is also the smallest compared to baselines in simulation. The results indicate that adjusting the bound online is better than fixing a best bound. Because setting the bound too low means when the robot deviates too much due to slipping conditions or sharp-turning trajectory, it is unable to quickly adjust its speed to lower the future deviations. If the bound is set too high, when slipping happens, the robot will try to tune the speed too quickly, so the resulting wheel force is large and the deviation cannot be quickly mitigated. Thus, is it critical that the robot maintains a high bound when driving normally and lowers the bound when slipping and sharp-turn happens.

In addition, our online framework adapts to different situations well, which helps with the sim-to-real transfer problem. For different scenarios, acceleration bounds might be different.

In simulation, the best is 1.1 m/s^2 and, in the real-world, it is 0.3 m/s^2. Thus, parameter sweeping is required for different scenarios if a fixed bound is used. On the other hand, no offline parameter sweeping is required for our online MPC framework. Since we do not use a dynamics model and the observations do not contain the entire trajectory information, the trained framework can be easily used under different scenarios without tuning, with terrain conditions and trajectories being drastically different. Our framework only utilizes the kinematic model and local trajectory information to decide if the current condition needs the acceleration bound to be changed. However, notice in real-world experiments, if the scenarios are entirely different, the system conducts a steep acceleration bound change when slip condition is detected. The reason could be that due to different scenarios, the system does not know how much to change the bound, so it just updates the upper bound to a relatively low value and the lower bound to a high value so to mitigate the deviation.

4.6 Conclusion

In this Chapter, we propose an online hierarchical MPC framework that performs robot trajectory tracking under slip conditions. Our method allows the robot to regulate its linear velocity when slip happens by regulating the linear acceleration bounds online. We conducted two sets of experiments to verify the tracking accuracy and adaptation ability of our model. The experiments showed that our model achieves the best performance compared to the baselines. Furthermore, in situations different from training setup, our framework is able to adapt without retraining, and still achieves similar performance to the best manually tuned framework.

Chapter 5

Mobile Robot Slip Rejection: Online Gain Tuning

5.1 Introduction

In Chapter 4, we discussed a solution to mobile robot trajectory tracking under slippery conditions. By using a reinforcement learning module to tune the low-level MPC optimization module, the system is able to adjust the conservative level of the control scheme to mitigate the slipping problem.

In this chapter, we follow a similar approach. However, instead of using a model predictive control scheme, we divide the control part into longitudinal and lateral control modules and propose the use of reinforcement learning to simultaneously tune parameters in the modules. Tuning gains directly within controllers provides a more straightforward method of influencing with lateral and speed tracking errors.

Brief Recap of Background and Motivation

As discussed previously, mobile robots are used in various industrial applications, such as manufacturing, processing, and aerospace. They often run on slippery terrains or routes with rapid cornering, which induce skidding and slipping. Excessive slip may cause motion instability, undermine maneuverability, and lead to possible collisions; thus, it must be prevented.

To mitigate slipping, many works have attempted to identify slip parameters or terrain states and to design simple control laws with robot kinematic models, such as [75] and [42]. [57] and [58] further choose to model slip as disturbance in the kinematics model and estimate by observers; however, the state vectors are of a high order and matrix inverse calculations could be massive. Another line of work has focused on wheel dynamics with traction forces. In [48], the authors use the Magic formula to derive the relationship between traction force and the slip ratio, while [61] formulates a detailed slip dynamics with certain switching conditions. Although dynamic models consider forces in addition to kinematics models,

they usually require system identification for different scenarios and have poor generalization abilities.

In addition, other studies have used reinforcement learning to directly learn a policy, such as in [76] and [64]. However, end-to-end RL approaches require considerable training time and pose challenges in explainability. Instead of end-to-end RL, the authors of [77] use RL only to optimize controllers, but their results are highly dependent on action space discretization.

Contributions

Our work in this chapter focuses on the trajectory tracking control of mobile robots under slippery conditions. We follow a similar approach to [77], and propose the use of a hierarchical framework that optimizes gains for the tracking controllers online. An RL module is used to tune gains in a lateral predictive controller and a longitudinal speed PID controller simultaneously for regulation. Through dividing the control part into longitudinal and lateral control modules and tuning gains directly, we improve lateral and speed tracking errors in a straightforward manner. By using a higher-level RL module, we are able to tune multiple low-level controllers simultaneously in real time. Furthermore, the RL module is only able to determine the conservativeness in the controllers; thus, the entire framework is more explainable than an end-to-end RL controller.

The contributions of our work are as follows: (1) We propose an hierarchical framework that actively optimizes controllers for slippery conditions through RL gain-tuning; (2) we reason the necessity of simultaneous online gain tuning through experiments; and (3) we demonstrate that our adaptive framework outperforms the best fixed-gain baselines by 6.6% and 12.7% for average lateral error and max lateral error by simulation.

The remainder of this chapter is organized as follows: Section 5.2 provides a detailed explanation of the proposed approach; Section 5.3 presents the experiments used to validate the effectiveness of our proposed framework; Section 5.4 discusses the experimental results; and Section 5.5 concludes the chapter.

5.2 Methodology

Problem Overview

Fig. 5.1 illustrates our tracking problem layout. The robot's goal is to travel from x_{start}, following a predefined trajectory to reach x_{end}. Here we define certain terms to describe the robot's motion and the tracking state. We use lateral displacement error e to represent the closest tracking error relative to the reference trajectory (in unit m). $\Delta\theta$ is the yaw error, which is the difference between reference yaw θ_{ref} and actual yaw θ (in unit rad). Δv is the speed error, which is the difference between absolute values of reference velocity v_{ref} and actual velocity v (in unit m/s).

The tracking task can then be formulated as a Markov Decision Process defined by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. $\mathcal{S}$ represents the state space; $\mathcal{A}$ is the action space; $\mathcal{T}(s^{'}|s, a)$ is the state transition model; $\mathcal{R}(s, a)$ is the reward function; and $\gamma \in [0, 1)$ is the discount factor. The RL formulation aims to learn a policy $\pi(a|S)$. The agent then follows the policy π, obtains an observation s_t at time t and performs an action a_t. It then receives from the environment a reward R_t and a new observation s_{t+1}, and π is updated accordingly. The final trained model gives an action selection policy π that maximizes the expectation of a discounted sum of rewards $E[\sum_{t=1}^{T} \gamma^{t-1} R_t]$.

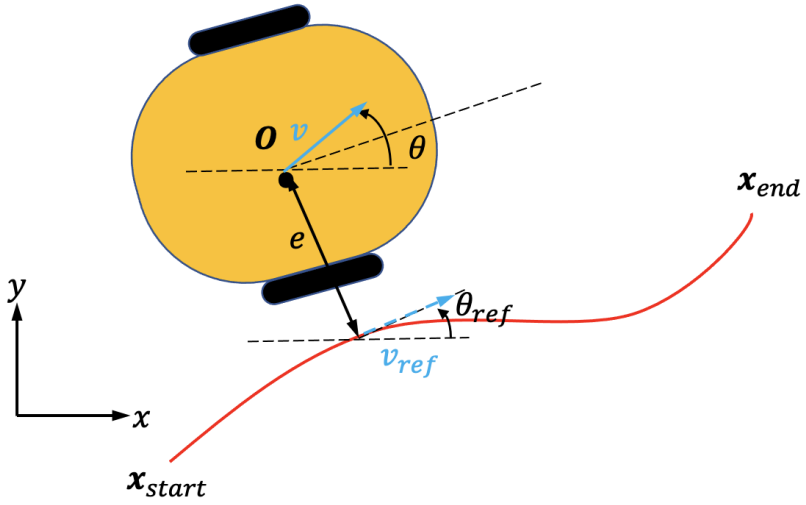


Figure 5.1: Schematic diagram of the problem setup.

The state S has 5 variables: e, $\Delta\theta$, Δv, $\Delta v_{c\_vs\_actual}$, and $\Delta\omega_{c\_vs\_actual}$. e, $\Delta\theta$, and Δv are as discussed in the beginning of the section. $\Delta v_{c\_vs\_actual}$ (in unit m/s)and $\Delta\omega_{c\_vs\_actual}$ (in unit rad/s) represent the difference between actual body velocities and body velocities calculated from wheel velocity commands(shown in Eqn. 5.1 and 5.2). Intuitively, a large value of $\Delta v_{c\_vs\_actual}$ or $\Delta\omega_{c\_vs\_actual}$ indicates the robot is slipping more severely as the wheel velocity commands are not fully transferred to actual body velocities.

$$v = \frac{(\omega_R + \omega_L) \cdot R}{2} \tag{5.1}$$

$$\omega = \frac{(\omega_R - \omega_L) \cdot R}{b} \tag{5.2}$$

Eqn. 1-2: transformation from wheel commands to calculated body linear and angular velocities. ω_R and ω_L are right and left wheel angular velocity commands, R is wheel radius, and b is distance between wheels.

The action A is $[v, \omega]$, which are linear and angular velocity commands. Notice the final input wheel velocity commands are calculated from reversing equations 1-2. The low level controller executes the commands and gets a reward at this step. The reward for each step is defined in Eqn. 5.3. Here we penalize e, $\Delta\theta$ and Δv, with coefficients R_{dist}, R_{ang} and R_{speed}. The cumulative reward is defined as $\sum_{t=1}^{T} \gamma^{t-1} R_t$, where R_t is the step reward.

$$R_t(s_t, a_t) = R_{dist} \cdot e^2 + R_{ang} \cdot \Delta\theta^2 + R_{speed} \cdot \Delta v^2 \tag{5.3}$$

Proposed Framework

We propose to utilize reinforcement learning to actively tune parameters in lateral and longitudinal control modules on a differential drive TurtleBot. The proposed framework consists of a RL-based high-level module, a lateral control module, a longitudinal control module, a low-level tracking controller, and the robot. The framework is visualized in Fig. 5.2. The RL module takes observed robot states *obs*; reference trajectory x_{ref}, and outputs gains $K_{stanley}$ and K_{speed}. The two control modules use the gains accordingly and output acceleration command α and steering angle command δ, and then transfer them into linear and angular velocity commands $[v, \omega]$. The low level controller then executes the command on the robot and feeds the directly observed states back into the RL module to calculate for step rewards, and the policy is updated accordingly. The loop stops when the positional error of the robot and final goal position is within a threshold. The entire framework realizes our MDP formulation, and is trained end-to-end using an RL algorithm.

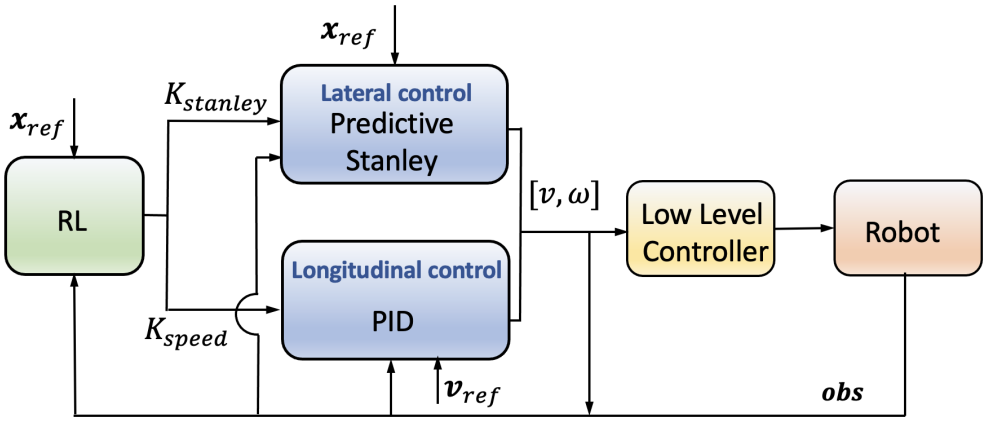


Figure 5.2: Proposed framework.

Lateral Control: Predictive Stanley

We tackle our trajectory tracking problem by breaking it up into longitudinal and lateral control problems. The longitudinal controller is responsible for regulating the robot's speed

while the lateral controller aims to reduce the lateral error during path tracking.

The proposed lateral control approach utilizes a version of Predictive Stanley controller, which is built on the basic Stanley controller. The basic Stanley controller is divided into three regions: saturated low region, saturated high region, and nominal region. ψ is the heading of the vehicle with respect to the heading of the trajectory at the point of the projected shortest distance to the vehicle position, $e(t)$ is the lateral error, v is current speed, and $K = K_{stanley}$ is the controller gain. See Fig. 5.3 for reference. The steering angle command is given by:

$$\delta(t) = \begin{cases} \psi(t) + arctan(\frac{Ke(t)}{v(t)}), & |\psi(t) + arctan(\frac{Ke(t)}{v(t)})| < \delta(max) \\ \delta(max), & \psi(t) + arctan(\frac{Ke(t)}{v(t)}) >= \delta(max) \\ -\delta(max), & \psi(t) + arctan(\frac{Ke(t)}{v(t)}) <= -\delta(max) \end{cases} \tag{5.4}$$

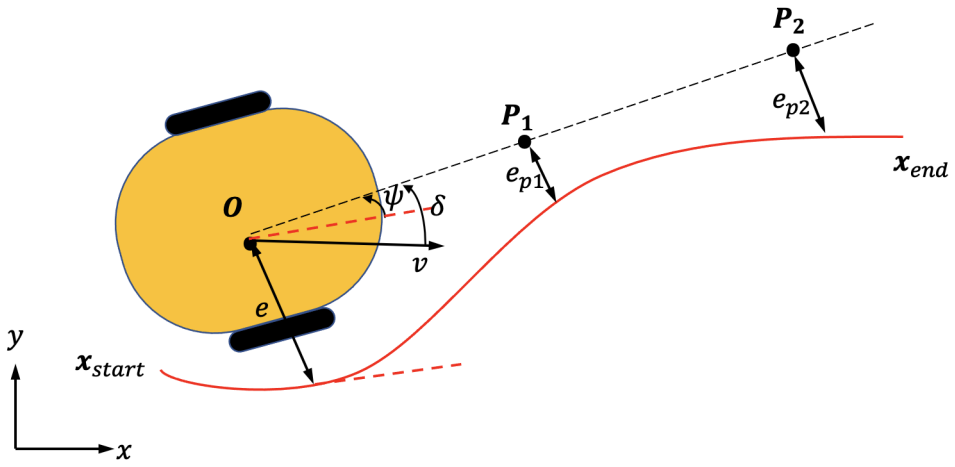


Figure 5.3: Predictive Stanley Representation.

As discussed in [78], the proposed predictive Stanley control approach introduces a third input, which is a developed array of future vehicle states, propagated along the vehicle track, denoted as P_1, P_2... P_N as shown in Fig. 5.3. At each future state, the corresponding δ is calculated based on current e_{pi}. The final steering angle command is calculated by augmenting the output of each basic Stanley controller at each state to eliminate the error along the path not only at the reference point, as shown in Eqn. 5.5 and 5.6. Consequently, the predictive Stanley controller is able to deal with the sudden changes in the heading angle of the trajectory by having this preview capability.

$$\delta(t) = \sum_{i=0}^{N} p_i[\psi_i(t) + arctan(\frac{Ke_{pi}(t)}{v(t)})] \tag{5.5}$$

$$p_i = p_{i-1}^2 \text{ for } i = 2...N \tag{5.6}$$

Eqn. 5-6. The final steering command. p_i is the weight, which represents how each controller contributes in determining the final value of the steering angle. Here we set $N = 2$ and $p_1 = 0.2$.

We show the advantage of our predictive Stanley controller over the basic Stanley controller by running them on a TurtleBot with the same trajectory. The visualization in Fig. 5.4 clearly shows that the predictive Stanley controller is able to adjust for abrupt turns. See detailed comparison in [78].

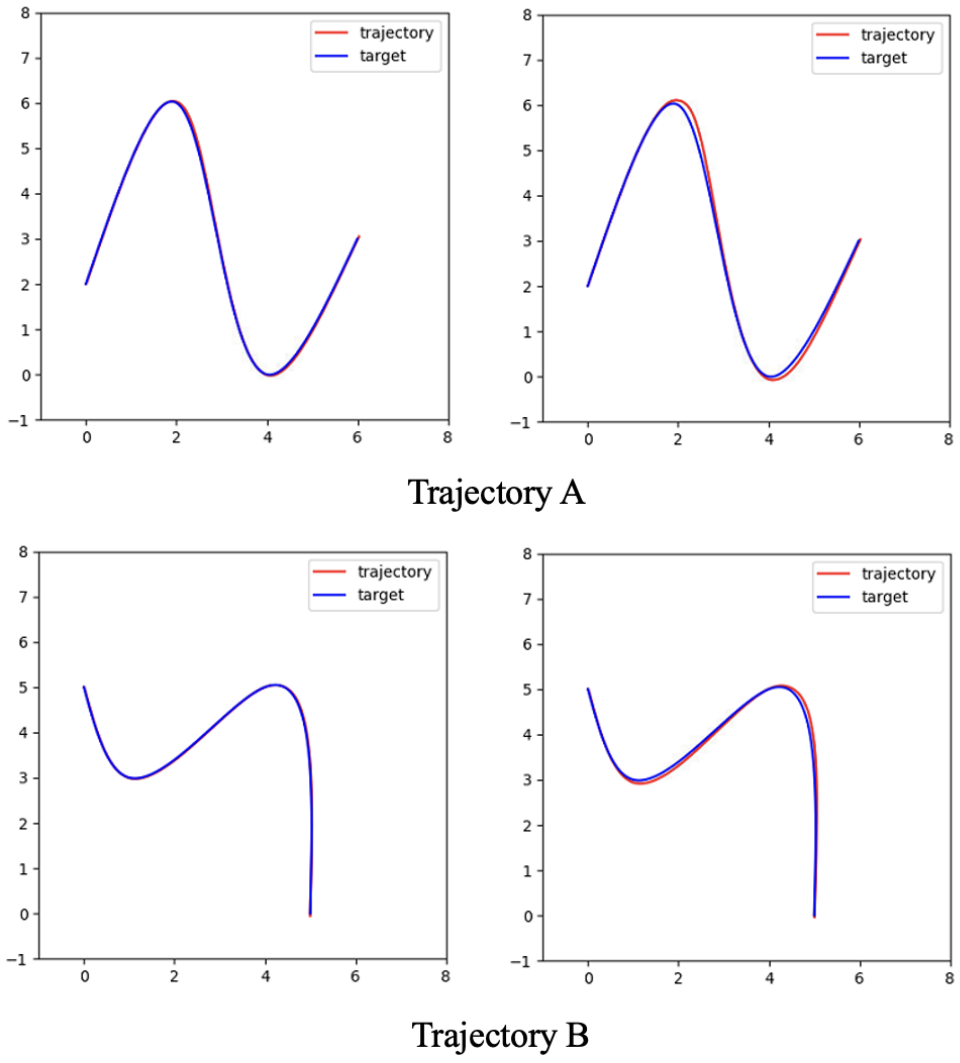


Figure 5.4: Predictive Stanley vs. Basic Stanley Controller.
The average lateral error for trajectory A for predictive and basic Stanley controllers are $0.0147m$ and $0.0374m$, respectively; for trajectory B are $0.0077m$ and $0.0347m$, respectively.

Longitudinal Control: PID

We use a simple proportional control for speed regulation. The acceleration command becomes:

$$\alpha(t) = K_{speed}(v_{ref} - v(t)) \tag{5.7}$$

With the steering and acceleration commands, we can deduce the robot's linear and angular velocity commands $[v, \omega]$ using Eqn. 5.8 and 5.9, which are then executed by the low level controller.

$$v_{command} = v(t) + \alpha(t) \cdot \Delta T \tag{5.8}$$

$$\omega_{command} = \frac{\delta(t)}{\Delta T} \tag{5.9}$$

Reinforcement Learning module

The RL module in Fig. 5.2 consists of actor and critic neural network layers. The entire framework in Fig. 5.2 utilizes soft actor-critic (SAC) during training.

5.3 Experiments

Our experiments were designed and conducted in order to answer the following questions:

1. Is simultaneous gain tuning necessary?
2. Is online gain tuning better than fixing the gains throughout the trajectory?
3. How to interpret our framework's output?

To answer these questions, we carry out simulated experiments using PyBullet by [72], with a TurtleBot waffle-pi model. To evaluate our framework, we propose to use a set of long-term and short-term metrics. Long-term metrics focus on measuring the performance throughout the entire trajectory, while short-term metrics focus on the short-time performance while the robot is slipping. Here we define slipping condition as those robot body states satisfying $\Delta v_{c\_vs\_actual} > |0.7| m/s$ or $\Delta \omega_{c\_vs\_actual} > |3| rad/s$.

For long-term criteria, we define average episodic reward $\overline{r}$, average lateral error $\overline{e}$, average speed error $\overline{\Delta v}$, and average RMS of change in low level control command $\overline{\Delta \mathbf{u}}$(which measures command stability). $\mathbf{u} = [\omega_L, \omega_R]$, which denotes left and right wheel velocity control action commands, and is calculated based on $[v, \omega]$, using Eqn. 5.1 and 5.2. The long-term metrics are calculated with respect to the entire trajectory.

$$\overline{r} = \frac{1}{T_{traj}} \sum_{i=0}^{T_{traj}} r_i \tag{5.10}$$

$$\overline{e} = \frac{1}{T_{traj}} \sum_{i=0}^{T_{traj}} e_i \tag{5.11}$$

$$\overline{\Delta v} = \frac{1}{T_{traj}} \sum_{i=0}^{T_{traj}} |v_i - v_{ref}|, \tag{5.12}$$

$$\overline{\Delta \mathbf{u}} = \frac{1}{T_{traj}} \sum_{i=1}^{T_{traj}} \|\mathbf{u}_{i+1} - \mathbf{u}_i\|, \tag{5.13}$$

For short-term criteria, we define max lateral error throughout the trajectory e_{max}, average lateral error during slipping(T_{slip}) $\overline{e}_{slip}$, average speed error during slipping $\overline{\Delta v}_{slip}$, and average RMS of change in low level control action during slipping $\overline{\Delta \mathbf{u}}_{slip}$.

$$e_{max} = max\{e_i\}_0^{T_{traj}} \tag{5.14}$$

$$\overline{e}_{slip} = \frac{1}{T_{slip}} \sum_{i=0}^{T_{slip}} e_i \tag{5.15}$$

$$\overline{\Delta v}_{slip} = \frac{1}{T_{slip}} \sum_{i=0}^{T_{slip}} |v_i - v_{ref}|, \tag{5.16}$$

$$\overline{\Delta \mathbf{u}}_{slip} = \frac{1}{T_{slip}} \sum_{i=1}^{T_{slip}} \|\mathbf{u}_{i+1} - \mathbf{u}_i\|, \tag{5.17}$$

Simulation environment setup and training

Fig. 5.5 shows bird-eye view simulation renderings of three example setups. Blue color represents high frictional areas with frictional coefficient μ=0.9, and red represents low frictional areas with μ=0.01. The red patches are of size 1 m by 1 m. The green trajectory is generated using a spline generator by [73]. The planner takes n number of 2D points and generates a smooth trajectory connecting all the given points.

To randomize the trajectory, we choose to use 5 random points for curve generation. Each point is $Uniform[1, 2]$ m away from the previous point, with $Uniform[-0.5\pi, 0.5\pi]$ rad angle from the previous point. The initial point $[x, y]$ position follows distribution: $x = Uniform[1, 2]$ m, $y = Uniform[3.5, 4.5]$ m.

To randomize the ground configuration, the red patches are randomly generated for each new trajectory, and we set 30% of the total area to be red.

We use SAC algorithm to train the entire framework. The policy network and the value network in the SAC are fully-connected two-layer neural networks of size 64. Learning rate is 0.0006, γ is set to be 0.99. The coefficients R_{dist}, R_{ang} and R_{speed} are set to be -20, -1, -1, respectively. The entire framework is trained until convergence.

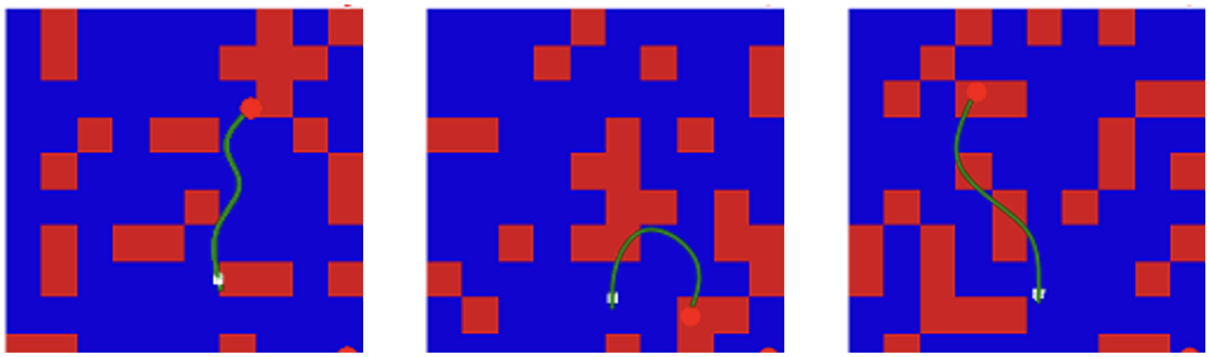


Figure 5.5: Simulation Rendering.
Bird-eye view image with annotations. White dot is the robot, green line is the reference trajectory, and red dot is the goal position.

Is simultaneous gain tuning necessary?

We propose to utilize RL to tune the two gains simultaneously, rather than having two separate frameworks to determine each. To verify the necessity of simultaneous gain tuning, we vary $K_{stanley}$ and K_{speed} from 0.5 to 5.0 with 0.5 increments, and plot heatmaps for each of the criteria discussed previously (Fig. 5.6 and 5.7). Each point on the heatmap represents the result of using a specific combination of $K_{stanley}$ and K_{speed}. Each point result is calculated by averaging the results of running 100 pre-generated random trajectories with random ground setups.

It can be shown that for each criteria, the best result happens when considering $K_{stanley}$ and K_{speed} together. For example, for $\overline{e}$, fixing $K_{stanley}$ to be 2.5 will result in a best K_{speed} of 3.5, but fixing $K_{stanley}$ to be 5.0 will result in a best K_{speed} of 2.0. Therefore the gains have to be tuned simultaneously in order to achieve the best results.

For different criteria, the optimum happens at different combinations of $K_{stanley}$ and K_{speed}, because the criteria are focused on different aspects. For instance, to stabilize command and reduce $\overline{\Delta \mathbf{u}}$, $\overline{e}$ may be compromised because commands need to be tuned less abruptly.

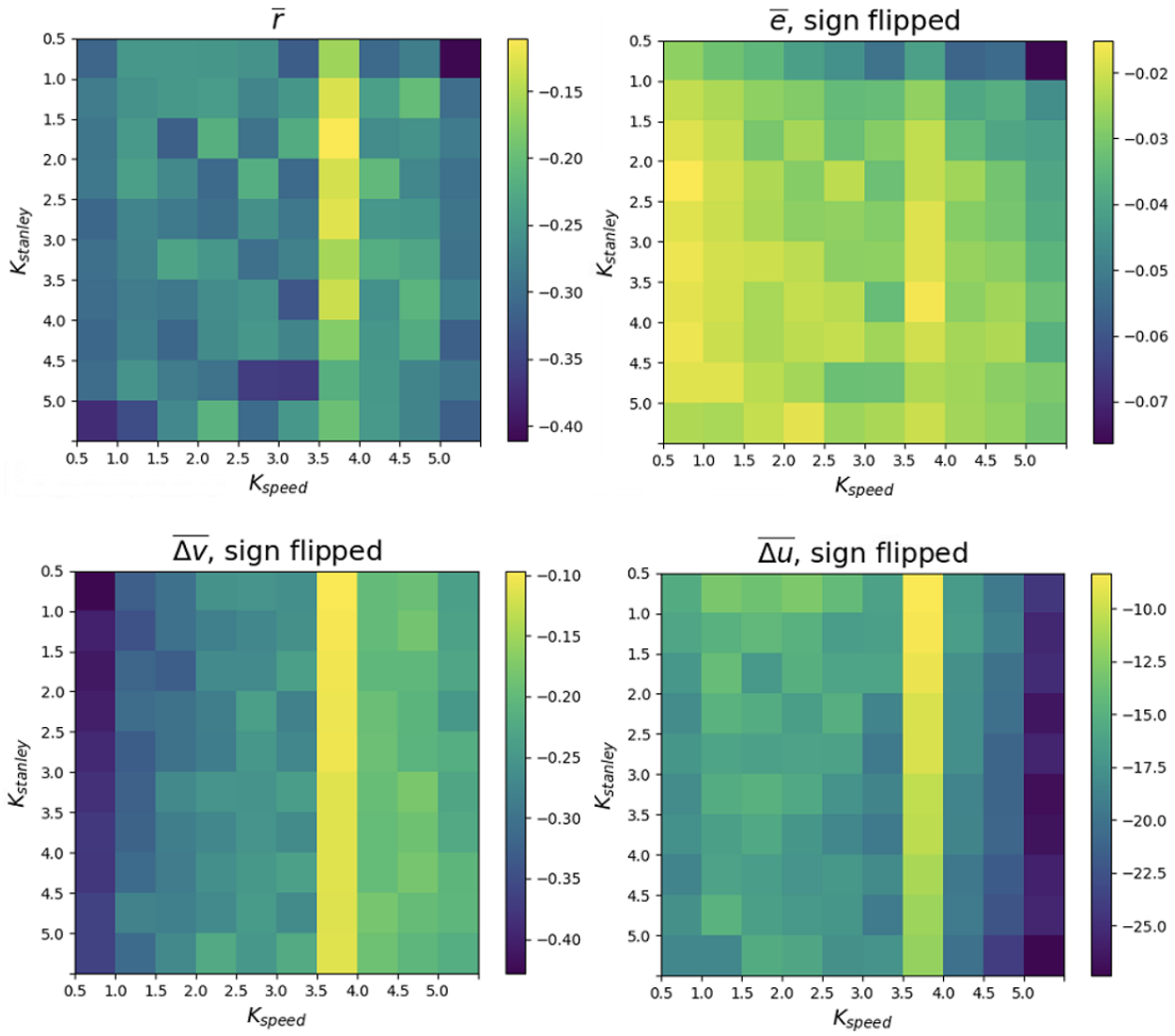


Figure 5.6: Parameter sweeping results for long-term metrics.

Is online gain tuning better than fixed parameters?

We propose to tune the gains online throughout the entire trajectory rather than using fixed gains. To verify this, we use a baseline model. The baseline model uses the same framework as in Fig. 5.2, but without the RL module. Instead of varying the two gains online, the baseline model utilizes fixed best gains found by offline parameter sweeping in $K_{stanley}$ and K_{speed}. The best gain combinations of baseline model for each metric is shown in the second column in *Parameter Sweeping* in Tables 5.1 and 5.2. We run the same 100 pre-generated random trajectories for the baseline model and our trained model, and log the results in the

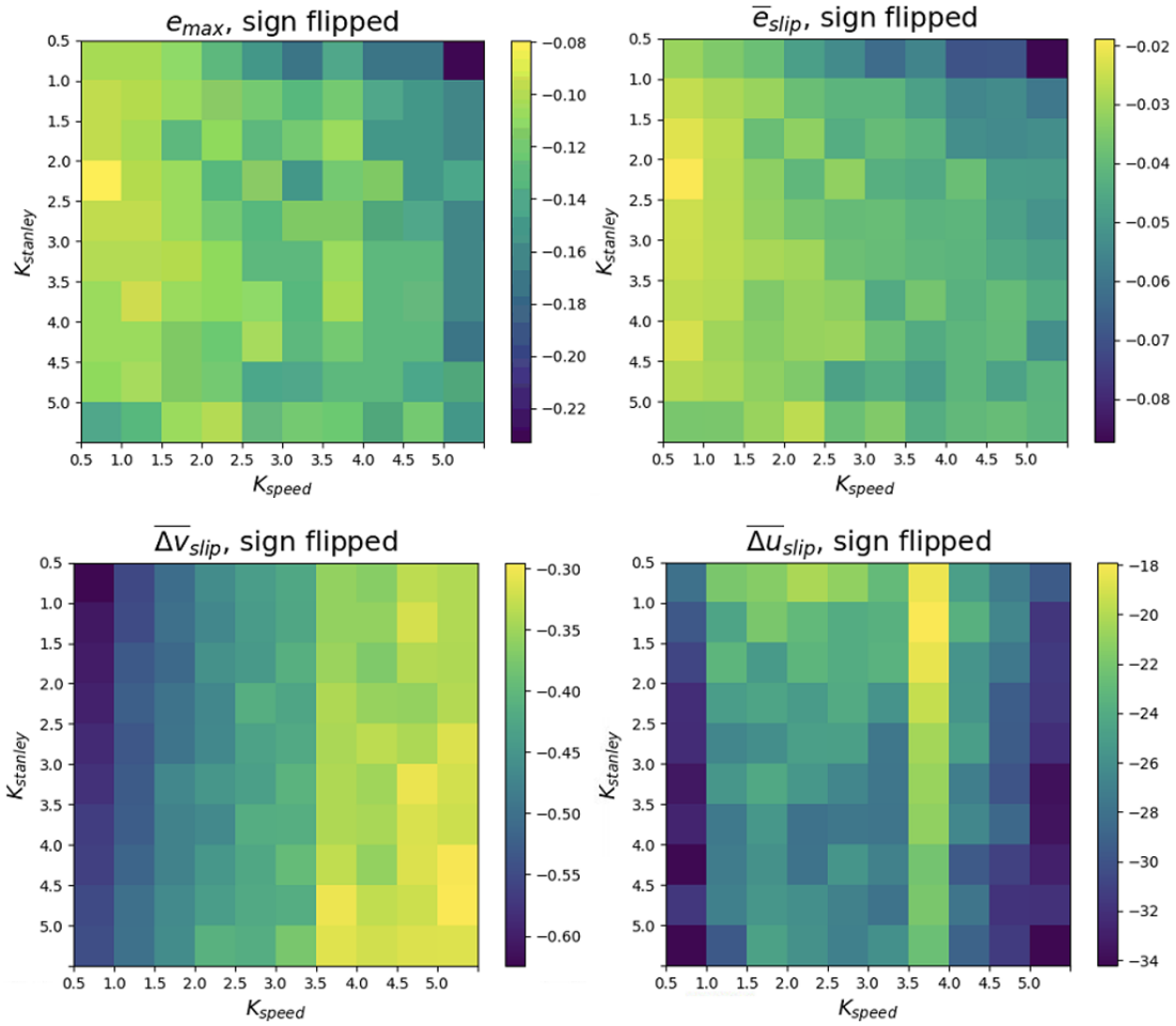


Figure 5.7: Parameter sweeping results for short-term metrics.

two tables. It can be shown that our framework is able to improve $\overline{e}$, e_{max}, $\overline{e}_{slip}$ by 6.6%, 12.7%, and 4.7%, respectively.

One thing to notice is that the best results for each metric happens at different gain combinations with the baseline parameter-sweeping model. For example $\overline{e}$ has best results when setting $K_{stanley} = 2.0$ and $K_{speed} = 0.5$, but for $\overline{\Delta \mathbf{u}}$ it's $K_{stanley} = 0.5$ and $K_{speed} = 3.5$, which means the baseline model will perform worse if using the same combination of gains for all metrics. However our model is still able to beat the best of each baseline model metric with a universal trained policy, in lateral error metrics and $\overline{\Delta \mathbf{u}}$ metric.

Table 5.1: Long-term metrics comparison.

The second column in Parameter Sweeping indicates at what value of gains the best metric result was obtained.

Metrics	Parameter Sweeping		Proposed Framework	Improvement
$(-)\overline{r}$	0.110 ± 0.118	$K_{stanley} = 1.5$ $K_{speed} = 3.5$	0.084 ± 0.125	23.6%
$\overline{e}$	0.015 ± 0.010	$K_{stanley} = 2.0$ $K_{speed} = 0.5$	0.014 ± 0.017	6.6%
$\overline{\Delta v}$	0.096 ± 0.042	$K_{stanley} = 0.5$ $K_{speed} = 3.5$	0.097 ± 0.058	-1.0%
$\overline{\Delta \mathbf{u}}$	8.320 ± 2.040	$K_{stanley} = 0.5$ $K_{speed} = 3.5$	5.917 ± 2.165	28.9%

Table 5.2: Short-term metrics comparison.

Metrics	Parameter Sweeping		Proposed Framework	Improvement
e_{max}	0.079 ± 0.055	$K_{stanley} = 2.0$ $K_{speed} = 0.5$	0.069 ± 0.066	12.7%
$\overline{e}_{slip}$	0.021 ± 0.013	$K_{stanley} = 2.0$ $K_{speed} = 0.5$	0.020 ± 0.032	4.7%
$\overline{\Delta v}_{slip}$	0.295 ± 0.082	$K_{stanley} = 4.5$ $K_{speed} = 5.0$	0.42 ± 0.087	-42.3%
$\overline{\Delta \mathbf{u}}_{slip}$	18.9 ± 5.93	$K_{stanley} = 1.0$ $K_{speed} = 3.5$	21.27 ± 4.76	-12.5%

Explainability of our framework output

We visualize two setup results in Fig. 5.8 and 5.9. The upper figures show the baseline model with best gain combinations, and the lower figures show our proposed model.

In the first setup, the robot using the baseline model failed to reach the end and got stuck when first enters the patch area, while our model is able to succeed. A closer look in the right figure reveals that our model reduces $K_{stanley}$ when the robot enters the low frictional area and detects slipping. It makes sense as when slipping happens, heavy steering will not help much and could worsen the slip. A good tuning on $K_{stanley}$ and K_{speed} helps the robot to control the slip and succeed in the tracking task in this case. And it is up to the RL module to decide when and how much to tune the two gains. Also, the RL module does not have prior knowledge of the location of low-frictional area, and it is able to tune the gains based on current tracking status and reduce lateral error successfully.

Similar observation can be made in the second setup. Both models succeeded in the task, but the baseline model induced more lateral error when the robot entered low-frictional area, because it did not lower $K_{stanley}$ accordingly.

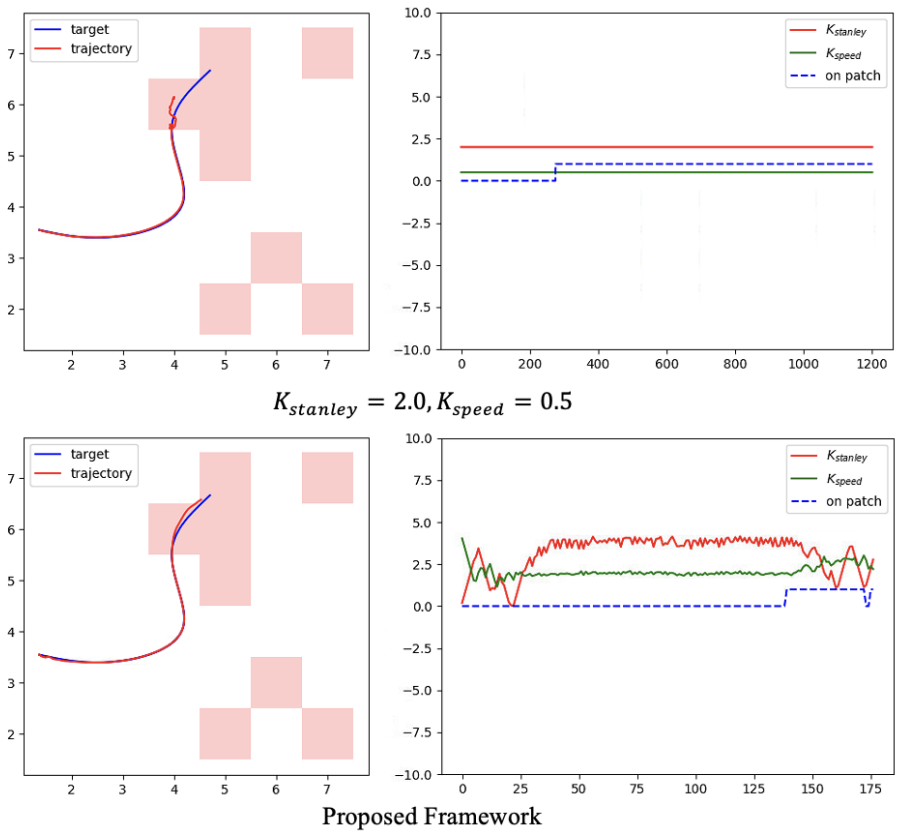


Figure 5.8: Trajectory Comparison 1.
The left figures show the bird-view map. The red patches correspond to the low frictional area. The right figures show how the gains evolve with increasing timesteps. The blue dotted line indicates when the robot is on the low frictional patch. A value of 1 indicates the robot is on patch.

5.4 Discussion

The experiments conducted show that simultaneous and online tuning of gains are necessary for mobile robot trajectory tracking under slippery conditions. Our model is able to tune the gains in lateral and longitudinal controls, and beat the baseline model in terms of lateral error metrics.

To reason the need of simultaneous gain tuning, consider when the robot is slipping or trajectory contains a large curvature. Both acceleration and steering commands should be considered to achieve an optimal tracking performance. For instance, with a large curvature, speed regulation can be relaxed while the steering command needs to have a bigger gain. Or when robot is slipping, both gains might need to be adjusted, as seen in Fig. 5.8 and 5.9. The magnitude of commands needs to be determined simultaneously, and the RL module in

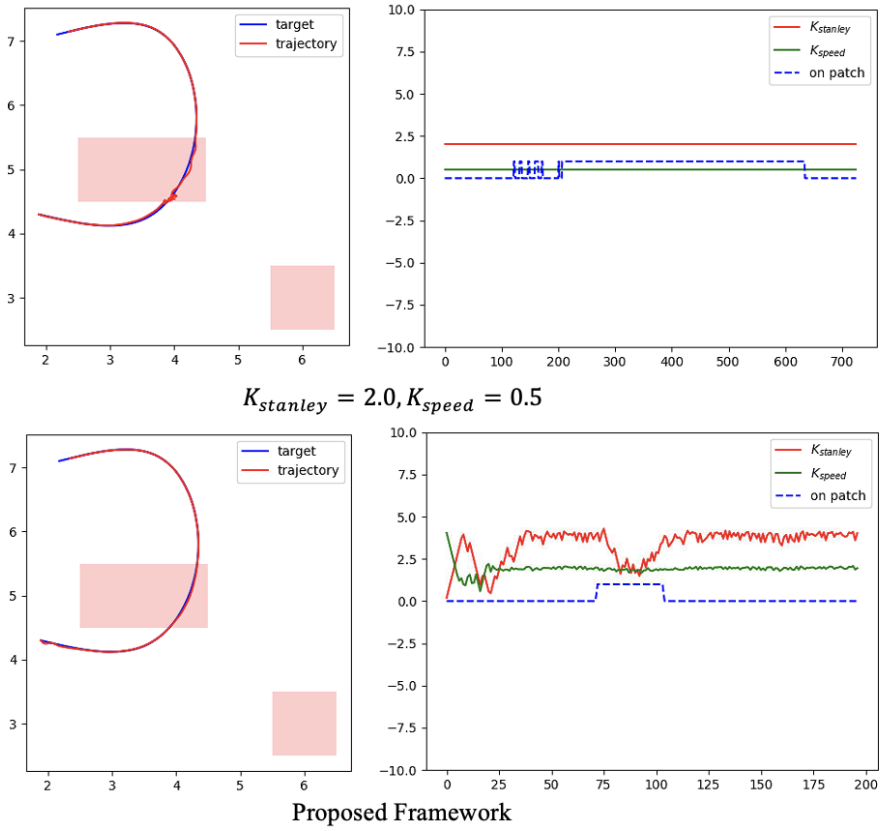


Figure 5.9: Trajectory Comparison 2.
Using online tuning instead of fixing gains alleviates deviation when the robot enters the slippery area. $\overline{e}$ for baseline and proposed framework are 0.0247 and 0.0112, respectively. e_{max} are 0.0624 and 0.0336, respectively.

our framework decides the magnitudes of both gains at each timestep.

Also notice for some metrics such as $\overline{\Delta \mathbf{u}}$ and $\overline{\Delta \mathbf{u}}_{slip}$, K_{speed} tuning dominates the performance. One reason is that speed regulation impacts the wheel velocity change more than angular regulation, because sometimes the curvature is not very abrupt for steering to change much, but speed has to be regulated all the time.

We showed our framework is able to improve $\overline{e}$, e_{max}, $\overline{e}_{slip}$ by 6.6%, 12.7%, and 4.7%, respectively. However the command and speed stability during slipping were compromised a bit. It makes sense as the robot tries to relax other constraints in order to reduce the lateral error. In many mobile robot slipping scenarios such as in a factory setting, the most important aspect is to reduce the lateral error because deviations could lead to robot hitting unwanted objects and causing harms. Therefore a lower stability in speed and command are acceptable in our case.

5.5 Conclusion

To reduce slipping for mobile robots, in this Chapter we propose a hierarchical framework that utilizes an RL module to adapt gains for the tracking controllers simultaneously online. We demonstrated the necessity of simultaneous gain tuning, and showed that our online framework outperforms the best baseline model using fixed gains, especially in terms of long and short term lateral errors.

Chapter 6

Final Words

This dissertation has explored mobile robot prediction and control under uncertainties. Specifically, we considered two tasks – namely mobile robot pushing and mobile robot trajectory tracking under slippery conditions. These two tasks exhibit highly nonlinear behavior and suffer from model parameter uncertainties and external disturbances.

The first task was discussed in Chapters 2 and 3. In Chapter 2, we explored using a combined prediction model and an online learning framework for planar push prediction. By adjusting low-dimensional analytical parameters in online situations, the robot was able to quickly adapt to the new environments. Then, in Chapter 3, we discussed a different approach, which discarded the analytical model altogether and used model-free reinforcement learning.

The second task was discussed in Chapters 4 and 5. In Chapter 4, we used reinforcement learning at a high level to adjust low-level model parameters, thereby achieving robust performance. Specifically, we explored the adaptation of key parameters in a model predictive control optimization formulation, which enabled the control scheme to be adjusted to be more aggressive or conservative. Then, in Chapter 5, we tested using a similar structure. By using reinforcement learning to conduct online gain tuning in lateral and longitudinal controllers, we demonstrated that long- and short-term lateral errors could be reduced.

Many directions exist for future work, which are listed as follows:

- Studies should consider using images or point clouds for tracking:

  The methodologies discussed in this dissertation rely on a tracking system that either provides object and robot positions and orientations in real time or that directly obtains the ground-truth position of the robot in the simulator. However, the position and orientation could be difficult to obtain in practice, especially with novel scenes and new objects. In future work, a possible direction would be to use images or point clouds as input to obtain positions and orientations, which would provide a more viable and possibly more accurate learning approach.

- Studies should consider roads with rough terrain conditions:

For the second task of mobile robot trajectory tracking on slippery terrain, we only considered smooth terrain. However, in the real world, terrain can also be bumpy and rough. This leads to more uncertainties in the tracking problem, as robot wheels may lose contact with the ground over bumpy ground. Thus, in future works, it may be worthwhile including more rough ground scenarios.

- Studies should explore more effective training schemes:

  For the second task of mobile robot trajectory tracking on slippery terrain, the training is performed directly in desired scenarios using Soft Actor-Critic [39]. However, such a training scheme can be further improved using techniques such as curriculum learning, which is a training strategy that trains a machine learning model from easier data to more difficult data. The training setting imitates the learning order as seen in human curricula. For instance, training could start with training the robot to follow straight lines with mild slipping; then, the difficulty level could be elevated to train the robot with complex trajectories under heavy slipping scenarios. The use of curriculum learning in many fields has been demonstrated to improve the generalization capacity and convergence rate of the model, and exploring its uses with our scenario would be valuable.

Many areas still exist to be studied in mobile robot prediction and tracking under uncertainties. Nevertheless, with the development of technology and more researchers paying attention in this field, it will not be long before we see mobile robots being used extensively in our everyday lives.

Bibliography

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

[2] Kevin M Lynch, Hitoshi Maekawa, and Kazuo Tanie. "Manipulation and active sensing by pushing using tactile feedback." In: *IROS*. Vol. 1. 1992.

[3] Tekin Meriçli, Manuela Veloso, and H Levent Akın. "Push-manipulation of complex passive mobile objects using experimentally acquired motion models". In: *Autonomous Robots* 38.3 (2015), pp. 317–329.

[4] Chelsea Finn, Ian Goodfellow, and Sergey Levine. "Unsupervised learning for physical interaction through video prediction". In: *Advances in neural information processing systems*. 2016, pp. 64–72.

[5] Annie Xie et al. "Improvisation through Physical Understanding: Using Novel Objects as Tools with Visual Foresight". In: *arXiv preprint arXiv:1904.05538* (2019).

[6] 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.

[7] 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.

[8] Zi Wang et al. "Focused model-learning and planning for non-Gaussian continuous state-action systems". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3754–3761.

[9] 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.

[10] Alina Kloss, Stefan Schaal, and Jeannette Bohg. "Combining learned and analytical models for predicting action effects". In: *arXiv preprint arXiv:1710.04102* (2017).

[11] 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.

[12] 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.

[13] Pulkit Agrawal et al. "Learning to poke by poking: Experiential learning of intuitive physics". In: *Advances in Neural Information Processing Systems*. 2016, pp. 5074–5082.

[14] Huidong Gao, Yi Ouyang, and Masayoshi Tomizuka. "Online learning in planar pushing with combined prediction model". In: *2021 American Control Conference (ACC)*. IEEE. 2021, pp. 2529–2536.

[15] 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.

[16] *TurtleBot3*. `https://www.turtlebot.com/`. Accessed: 2019-09.

[17] 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*. 2018.

[18] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. "Deep online learning via meta-learning: Continual adaptation for model-based RL". In: *arXiv preprint arXiv:1812.07671* (2018).

[19] 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.

[20] 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.

[21] Seiya Tokui et al. "Chainer: a Next-Generation Open Source Framework for Deep Learning". In: *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*. 2015.

[22] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[23] Chelsea Finn and Sergey Levine. "Deep visual foresight for planning robot motion". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2786–2793.

[24] 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.

[25] Alexander Sasha Vezhnevets et al. "Feudal networks for hierarchical reinforcement learning". In: *arXiv preprint arXiv:1703.01161* (2017).

[26] Ofir Nachum et al. "Data-efficient hierarchical reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2018, pp. 3303–3313.

[27] Tejas D Kulkarni et al. "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation". In: *Advances in neural information processing systems*. 2016, pp. 3675–3683.

[28] Andrew Levy et al. "Learning multi-level hierarchies with hindsight". In: *arXiv preprint arXiv:1712.00948* (2017).

[29] Matthew Thomas Mason. "Manipulator grasping and pushing operations". In: (1982).

[30] Matthew T Mason. "On the scope of quasi-static pushing". In: *International Symposium on Robotics Research, 1986*. 1986, pp. 229–233.

[31] Randy C Brost. "Dynamic analysis of planar manipulation tasks". In: *Proceedings 1992 IEEE International Conference on Robotics and Automation*. IEEE Computer Society. 1992, pp. 2247–2248.

[32] Yan-Bin Jia and Michael Erdmann. "Pose and motion from contact". In: *The International Journal of Robotics Research* 18.5 (1999), pp. 466–487.

[33] Tom Erez, Yuval Tassa, and Emanuel Todorov. "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 4397–4404.

[34] Jimmy Wu et al. "Spatial Action Maps for Mobile Manipulation". In: *Robotics: Science and Systems XVI* (July 2020). DOI: 10.15607/rss.2020.xvi.035. URL: http://dx.doi.org/10.15607/RSS.2020.XVI.035.

[35] Misha Denil et al. "Learning to perform physics experiments via deep reinforcement learning". In: *arXiv preprint arXiv:1611.01843* (2016).

[36] Michael B Chang et al. "A compositional object-based approach to learning physical dynamics". In: *arXiv preprint arXiv:1612.00341* (2016).

[37] F. Xia et al. "ReLMoGen: Leveraging Motion Generation in Reinforcement Learning for Mobile Manipulation". In: *ArXiv* abs/2008.07792 (2020).

[38] Chengshu Li et al. "HRL4IN: Hierarchical Reinforcement Learning for Interactive Navigation with Mobile Manipulators". In: *ArXiv* abs/1910.11432 (2019).

[39] T. Haarnoja et al. "Soft Actor-Critic Algorithms and Applications". In: *ArXiv* abs/1812.05905 (2018).

[40] Michael Burke. "Path-following control of a velocity constrained tracked vehicle incorporating adaptive slip estimation". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 97–102.

[41] Liang Ding et al. "Slip-ratio-coordinated control of planetary exploration robots traversing over deformable rough terrain". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 4958–4963.

[42] Jayoung Kim and Jihong Lee. "A kinematic-based rough terrain control for traction and energy saving of an exploration rover". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 3595–3600.

[43] Andrea Botta et al. "An Estimator for the Kinematic Behaviour of a Mobile Robot Subject to Large Lateral Slip". In: *Applied Sciences* 11.4 (2021), p. 1594.

[44] B.J. Chol and S.V. Sreenivasan. "Motion planning of a wheeled mobile robot with slip-free motion capability on a smooth uneven surface". In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. Vol. 4. 1998, 3727–3732 vol.4. DOI: 10.1109/ROBOT.1998.681420.

[45] K. Iagnemma et al. "Online terrain parameter estimation for wheeled mobile robots with application to planetary rovers". In: *IEEE Transactions on Robotics* 20.5 (2004), pp. 921–927. DOI: 10.1109/TRO.2004.829462.

[46] Isabelle Motte and Guy Campion. "A slow manifold approach for the control of mobile robots not satisfying the kinematic constraints". In: *IEEE Transactions on Robotics and Automation* 16.6 (2000), pp. 875–880.

[47] Naim Sidek and Nilanjan Sarkar. "Dynamic Modeling and Control of Nonholonomic Mobile Robot with Lateral Slip". In: *Third International Conference on Systems (icons 2008)*. 2008, pp. 35–40. DOI: 10.1109/ICONS.2008.22.

[48] Yu Tian, Naim Sidek, and Nilanjan Sarkar. "Modeling and control of a nonholonomic Wheeled Mobile Robot with wheel slip dynamics". In: *2009 IEEE Symposium on Computational Intelligence in Control and Automation*. 2009, pp. 7–14. DOI: 10.1109/CICA.2009.4982776.

[49] Gonzalo Farias et al. "Reinforcement Learning for Position Control Problem of a Mobile Robot". In: *IEEE Access* 8 (Aug. 2020), pp. 152941–152951. DOI: 10.1109/ACCESS.2020.3018026.

[50] Shu Li et al. "Adaptive neural network tracking control-based reinforcement learning for wheeled mobile robots with skidding and slipping". In: *Neurocomputing* 283 (2018), pp. 20–30. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.12.051. URL: https://www.sciencedirect.com/science/article/pii/S0925231217319136.

[51] Huidong Gao et al. "Reinforcement Learning Based Online Parameter Adaptation for Model Predictive Tracking Control Under Slippery Condition". In: *2022 American Control Conference (ACC)*. IEEE. 2022, pp. 2675–2682.

[52] Wei Yu et al. "Analysis and experimental verification for dynamic modeling of a skid-steered wheeled vehicle". In: *IEEE transactions on robotics* 26.2 (2010), pp. 340–353.

[53] Neal Seegmiller et al. "A unified perturbative dynamics approach to online vehicle model identification". In: *Robotics Research*. Springer, 2017, pp. 585–601.

[54] Jorge L Martınez et al. "Approximating kinematics for tracked mobile robots". In: *The International Journal of Robotics Research* 24.10 (2005), pp. 867–878.

[55] Anthony Mandow et al. "Experimental kinematics for wheeled skid-steer mobile robots". In: *2007 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2007, pp. 1222–1227.

[56] W.E. Dixon, D.M. Dawson, and E. Zergeroglu. "Robust control of a mobile robot system with kinematic disturbances". In: *Proceedings of the 2000. IEEE International Conference on Control Applications. Conference Proceedings (Cat. No.00CH37162)*. 2000, pp. 437–442. DOI: 10.1109/CCA.2000.897463.

[57] Bijo Sebastian and Pinhas Ben-Tzvi. "Active disturbance rejection control for handling slip in tracked vehicle locomotion". In: *Journal of Mechanisms and Robotics* 11.2 (2019), p. 021003.

[58] Shuai Wang and Junyong Zhai. "A trajectory tracking method for wheeled mobile robots based on disturbance observer". In: *International Journal of Control, Automation and Systems* 18.8 (2020), pp. 2165–2169.

[59] R.L. Williams et al. "Dynamic model with slip for wheeled omnidirectional robots". In: *IEEE Transactions on Robotics and Automation* 18.3 (2002), pp. 285–293. DOI: 10.1109/TRA.2002.1019459.

[60] Chris C. Ward and Karl Iagnemma. "A Dynamic-Model-Based Wheel Slip Detector for Mobile Robots on Outdoor Terrain". In: *IEEE Transactions on Robotics* 24.4 (2008), pp. 821–831. DOI: 10.1109/TRO.2008.924945.

[61] S. Nandy et al. "Detailed slip dynamics for nonholonomic mobile robotic system". In: *2011 IEEE International Conference on Mechatronics and Automation*. 2011, pp. 519–524. DOI: 10.1109/ICMA.2011.5985616.

[62] Hans B. Pacejka and Egbert Bakker. "THE MAGIC FORMULA TYRE MODEL". In: *Vehicle System Dynamics* 21.sup001 (1992), pp. 1–18. DOI: 10.1080/00423119208969994. eprint: https://doi.org/10.1080/00423119208969994. URL: https://doi.org/10.1080/00423119208969994.

[63] Xiaoshan Gao, Liang Yan, and Chris Gerada. "Modeling and Analysis in Trajectory Tracking Control for Wheeled Mobile Robots with Wheel Skidding and Slipping: Disturbance Rejection Perspective". In: *Actuators* 10.9 (2021). ISSN: 2076-0825. DOI: 10.3390/act10090222. URL: https://www.mdpi.com/2076-0825/10/9/222.

[64] Peide Cai et al. "High-Speed Autonomous Drifting With Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1247–1254. DOI: 10.1109/LRA.2020.2967299.

[65] Shansi Zhang et al. "Trajectory-Tracking Control of Robotic Systems via Deep Reinforcement Learning". In: *2019 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*. 2019, pp. 386–391. DOI: 10.1109/CIS-RAM47153.2019.9095802.

[66] Shuti Wang et al. "Trajectory Tracking Control for Mobile Robots Using Reinforcement Learning and PID". In: *Iranian Journal of Science and Technology, Transactions of Electrical Engineering* 44 (Nov. 2019), pp. 1–10. DOI: 10.1007/s40998-019-00286-4.

[67] Lei Yuan et al. "Nonlinear MPC-based slip control for electric vehicles with vehicle safety constraints". In: *Mechatronics* 38 (2016), pp. 1–15.

[68] Yan Ma et al. "MPC-based slip ratio control for electric vehicle considering road roughness". In: *IEEE Access* 7 (2019), pp. 52405–52413.

[69] Mohit Mehndiratta, Efe Camci, and Erdal Kayacan. "Automated Tuning of Nonlinear Model Predictive Controller by Reinforcement Learning". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 3016–3021. DOI: 10.1109/IROS.2018.8594350.

[70] P. Travis Jardine et al. "Adaptive predictive control of a differential drive robot tuned with reinforcement learning". In: *International Journal of Adaptive Control and Signal Processing* 33.2 (2019), pp. 410–423. DOI: https://doi.org/10.1002/acs.2882. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/acs.2882. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/acs.2882.

[71] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[72] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2021.

[73] Atsushi Sakai et al. "PythonRobotics: a Python code collection of robotics algorithms". In: *arXiv preprint arXiv:1808.10703* (2018).

[74] Engineering ToolBox. *Friction and Friction Coefficients*. 2004. URL: %5Curl%7Bhttps://www.engineeringtoolbox.com/friction-coefficients-d%5C_778.html%7D.

[75] Nabih Pico et al. "Climbing control of autonomous mobile robot with estimation of wheel slip and wheel-ground contact angle". In: *Journal of Mechanical Science and Technology* 36.2 (2022), pp. 959–968.

[76] 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.

[77] Ignacio Carlucho, Mariano De Paula, and Gerardo G Acosta. "Double Q-PID algorithm for mobile robot control". In: *Expert Systems with Applications* 137 (2019), pp. 292–307.

[78] Ahmed AbdElmoniem et al. "A path-tracking algorithm using predictive Stanley lateral controller". In: *International Journal of Advanced Robotic Systems* 17.6 (2020), p. 1729881420974852.