MACHINE LEARNING FOR AGILE ROBOTIC CONTROL

A Dissertation
Presented to
The Academic Faculty

By

Nolan Wagener

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

December 2023

MACHINE LEARNING FOR AGILE ROBOTIC CONTROL

Thesis committee:

Dr. Byron Boots, Advisor
School of Computer Science & Engineering
University of Washington

Dr. Panagiotis Tsiotras, Co-Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Sehoon Ha
School of Interactive Computing
Georgia Institute of Technology

Dr. Seth Hutchinson
School of Interactive Computing
Georgia Institute of Technology

Dr. Andreas Krause
Department of Computer Science
ETH Zürich

Date approved: November 16, 2023

People say to me, “Are you looking for the ultimate laws of physics?” No, I’m not. I’m just looking to find out more about the world, and if it turns out there is a simple, ultimate law which explains everything, so be it; that would be very nice to discover. If it turns out it’s like an onion with millions of layers, and we’re just sick and tired of looking at the layers, then *that’s* the way it is.

Richard Feynman

For my parents, grandparents, and brother

ACKNOWLEDGMENTS

First and foremost, I would like to thank Byron Boots for being an incredible advisor. He took a chance on me when he was a new professor at Georgia Tech, and, despite all the pressure that comes with being a tenure-track professor, he gave me the freedom to work on whatever topic I chose. I greatly appreciate his patience and advice he has given me over the years. Even after moving to the University of Washington, he continued to support me as an advisor and jumped through hoops to keep me funded even though I stayed at Georgia Tech.

I also want to thank Panos Tsiotras for taking me on as a co-advisor after Byron changed universities. He always took an interest in my work (even if it was a bit outside of his research scope), gave me intriguing advice and insight from a controls perspective, and showed patience in me over the years.

I'd like to thank the other members of my committee, Sehoon Ha, Seth Hutchinson, and Andreas Krause, for their support and insight throughout the thesis process.

Being a member of the Robot Learning Lab has been an absolute pleasure. My colleagues in this lab are incredibly talented yet easygoing, and they have been invaluable for bouncing off ideas or venting when experiments weren't working. I'd like to thank: Ching-An Cheng, Xinyan Yan, Mustafa Mukhadam, Amirreza Shaban, Sasha Lambert, Yuxiang Yang, Rosario Scalise, Jacob Sacks, Anqi Li, Mohak Bhardwaj, Sandesh Adhikary, Ivan Rodriguez, Aravind Battaje, Hemanth Sarabu, Nathan Hatch, Jing Dong, Siddarth Srinivasan, and Asif Rana.

The AutoRally project was a groundbreaking and awesome project to be a part of, and I'd like to thank the colleagues that I worked with: Brian Goldfain, Grady Williams, Paul Drews, Evangelos Theodorou, Kelsey Hawkins, Jason Gibson, Keuntaek Lee, Jacob Knaup, Sahit Chintalapudi, and Anthony Liu. I would like to especially thank Brian, Grady, and Paul for on-boarding me onto the project and for all their help when I began conducting

my own test runs.

The SARA and DARPA RACER projects are challenging and ambitious projects, and my colleagues on these projects made them a pleasure to work on. These colleagues include: Nathan Hatch, Matthew Schmittle, Tyler Han, Jakub Filipek, Alex Spitzer, Greg Okopal, and Gilwoo Lee.

My internship at Microsoft Research (MSR) was an intellectually stimulating and a great joy. I'd like to thank my colleagues from the Robotics Group at MSR: Matthew Hausknecht, Ching-An Cheng, Andrey Kolobov, Ricky Loynd, and Felipe Vieira Frujeri. I'd like to especially thank Matthew for being an awesome mentor and for giving me the freedom to approach the internship project in whatever way I wished.

Fridays with the CPL Drinks happy hour group have been phenomenal. It has allowed me to kick back for relaxing and fun evenings. I'd like to thank the friends I've made in the group: Brian Goldfain, Daniel Henderson, Kate Bennett, Walter Hooper, Linda Yang, Brian Hrolenok, Amrita Gupta, Kalesha Bullard, Ashley Kunkle, Shray Bansal, Jason Fernando, Sarah Wiegreffe, Nihal Soans, María Santos, Jon Balloch, Andrew Price, Paul David, Vivian Chu, Tesca Fitzgerald, and Kayla DesPortes.

I'd like to thank Kate Bennett and Davy for their continued love and support over the past few years, especially during the thesis writing process.

Finally, I'd like to thank my parents and grandparents for supporting my education and for giving me an upbringing that allowed me to reach this point.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Roboticists typically exploit structure in a problem, such as by modeling the mechanics of a system, to generate solutions for a given task. However, this structure can limit flexibility and require practitioners to reason about challenging phenomena, such as contacts in mechanics. Data, conversely, provides much more flexibility and, when combined with deep neural networks, has given rise to powerful models in vision and language, all with little hand-engineered structure. While it is tempting to fully forego structure in favor of learning-based methods for robotics, we show how data and learning can be gracefully incorporated in a structured way. In particular, we focus on the control setting, and we demonstrate that robotic control offers a variety of modes where learning and data can be utilized. First, we show that data can be used in a model-based fashion to train a neural network that approximates complex dynamics and which can be used within a model predictive controller (MPC). Then, we show that the MPC process is itself an instance of online learning and demonstrate how to synthesize MPC algorithms from a common online learning algorithm. We apply both of the aforementioned approaches on a real-world aggressive driving task and show that they can accomplish the task. Next, we consider the safe reinforcement learning problem and show that safety interventions can be used as a learning signal to have an agent learn to become safe without needing to execute unsafe actions in the environment. Finally, we consider the simulated humanoid domain and show that pre-collected human motions can act as a strong inductive bias to ground motions learned by the humanoid agent.

CHAPTER 1
INTRODUCTION

As a field, robotics combines many different domains under one common umbrella: mechanical design, control, planning, perception, and so on. Despite the imposing scale of combining all these domains, roboticists can create robotic systems that perform well in the real world (Urmson et al., 2008). To do this, they take advantage of the inherent structure of the problem: using the perception system to understand the environment and get a state estimate of the robot, and then feeding that environment representation and state estimate to a planner to devise a valid plan through the environment, and finally feeding that plan to a controller to compute controls that track the plan.

Traditionally, this reliance on structure has extended to the sub-domains of robotics themselves, such as, in mechanics, by directly deriving the equations of motion from first principles (Craig, 2022) or, in perception, by extracting hand-specified features from an image (Lowe, 2004). These hand-engineered approaches and inductive biases allow for easier understanding of a system and quicker generation of a solution. Nonetheless, this rigid structure provides less flexibility, and it can require the practitioner to model challenging phenomena, such as contact and sliding in mechanics and differentiating pedestrians from the surrounding area in vision.

Data, however, provides much more flexibility. Because data is a reflection of phenomena from the environment, we can use it to form higher fidelity models, such as using system identification (Ljung, 1987). More broadly, machine learning (Bishop, 2006), the field of study concerned with learning models from data, offers an appealing way to generate structures that normally would be hand-engineered. Modern machine learning, specifically deep learning (LeCun, Bengio, et al., 2015), has used deep neural networks and vast amounts of data to produce state-of-the-art models in computer vision (Krizhevsky et al.,

2012), language (Brown et al., 2020; OpenAI, 2023), and challenging games like Go (Silver et al., 2016). In particular, these neural networks take the high-dimensional inputs (like images) and extract their own *learned* features to aid in making a prediction. Therefore, given some task, we imagine that, with enough data, we can train a neural network to accomplish this task.

Unfortunately, we cannot naively transfer this idea to robotics (Kober, Bagnell, et al., 2013; Roy et al., 2021; Sünderhauf et al., 2018). We have much less data available in robotics, and the data is usually of lower quality (e.g., corrupted by sensor noise) or doesn't have clear supervision labels (e.g., what the proper action is). Furthermore, unlike standard machine learning, a trained model is not evaluated on some held-out "test set," but is instead deployed into an embodied world. Because of this, a robot's actions alter the world, and any mistakes made could lead the robot to a part of the world that it wasn't trained on, leading to compounding (and potentially catastrophic) errors (Ross et al., 2011). Thus, we must instead strike a fine balance, taking advantage of known structure in the problem while using learning and the limited data we have where they best work within this structure.

This dissertation focuses on the application of machine learning to robotic control. We show that learning can be gracefully incorporated into the control structure and offers many different modes for utilizing data. In particular, the dissertation centers around the following statement:

Machine learning offers a rich variety of structured ways to utilize data for achieving agile robotic motion.

We explore this notion in this thesis by:

- modeling off-road driving dynamics with neural networks for use with a model predictive controller (Chapter 4),
- re-examining model predictive control from the online learning perspective (Chapter 5),

- using safety interventions as a learning signal for safe reinforcement learning (Chapter 7), and
- leveraging human motions to ground learned robot motions (Chapter 8).

This dissertation is organized as follows. We first give preliminary material in Chapter 2 which is used throughout the document, including Markov decision processes and various probability concepts. We then split the thesis into two parts: Part I on model predictive control, and Part II on reinforcement learning.

Part I focuses on the application of machine learning to model predictive control (MPC). We first give MPC-specific preliminary material in Chapter 3. Then, in Chapter 4, we show how to use data-driven methods to learn dynamics models for control. In particular, we use expressive neural networks to represent the dynamics, and we show that they can accurately model nonlinear and complicated phenomena like sliding. We also derive the model predictive path integral (MPPI) algorithm, a sampling-based MPC method that can work with general dynamics and objective functions. Combining MPPI with neural network dynamics, we achieve performant control of several simulated systems and a real-world off-road vehicle at high speeds. Next, in Chapter 5, we turn our attention to the MPC framework itself. We show that there is a close connection between MPC and online learning, a framework for analyzing online decision making. Through this lens, we frame MPC as an online learning problem and propose a general MPC algorithm based on an algorithm from the online learning setting. From there, we show that several well-known MPC algorithms are special cases of this general online learning algorithm, and we discuss several design parameters that are exposed as a result of this perspective.

Part II focuses on how data can be used as side information for reinforcment learning (RL). We first give RL-specific preliminary material in Chapter 6. Then, in Chapter 7, we turn to the safe reinforcement setting, where we seek to optimize a policy while satisfying safety constraints throughout training. To ensure safety, we allow safety interventions to happen (e.g., through a safety monitor). Our novel contribution is that the interventions

can be used as a *learning signal* for our policy to become safe. We propose a safe RL algorithm that converts the intervention into a penalizing reward, and we give theoretical guarantees that the learned policy will be both safe and performant, even when safety interventions are disabled. Next, in Chapter 8, we show that human motion capture (MoCap) data can be used to teach simulated embodied humanoids realistic motions. Using large-scale reinforcement learning, we generate an embodied dataset corresponding to 3.5 hours of human motion covering a wide variety of behaviors. With this dataset, we bootstrap the behaviors of simulated humanoids to speed up learning for downstream tasks, and we train a GPT-style policy to perform motion completion.

In Chapter 9, we summarize the main contributions of this thesis and discuss directions for future research.

Finally, the appendices give extra proofs and details for the main material.

CHAPTER 2

PRELIMINARIES

2.1 Markov Decision Processes

We consider a Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, p)$, which gives a mathematical description for how an agent interacts with an environment (Puterman, 1994). An MDP consists of the following:

- $\mathcal{S}$: the *state space*, with an element $s \in \mathcal{S}$ corresponding to a *state*. The state is a summary of the system's configuration that is sufficient for making predictions about the system.
- $\mathcal{A}$: the *action space*, with an element $a \in \mathcal{A}$ corresponding to an *action*.
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: the *reward function*.
- $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$: the *transition kernel*, which maps the current state s and current action a to a probability distribution over successive states s'.

The decision rule for choosing action a at state s is given by a stochastic policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. Given some distribution $p_0(s_0)$ of initial states, we define $\tau = (s_0, a_0, s_1, a_1, \dots)$ as a *trajectory* and $\rho^\pi(\tau)$ as the *trajectory distribution under* π, where:

$$\rho^\pi(\tau) \triangleq p_0(s_0) \prod_{t=0}^{\infty} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t).$$

Our goal is to find a policy π that maximizes the average reward:

$$\max_\pi \quad \lim_{T\to\infty} \mathbb{E}_{\rho^\pi(\tau)} \left[\frac{1}{T} \sum_{t=0}^{T-1} r(s_t, a_t) \right] \tag{2.1}$$

This is a very difficult problem to solve, so many methods solve an approximation of it. We briefly cover two paradigms.

2.1.1 Discount Factors

Here, we introduce a *discount factor* $\gamma \in [0, 1)$, which encodes an "effective horizon" for the problem. Then, the objective function is to maximize the expected sum of discounted rewards:

$$\max_{\pi} \quad \mathbb{E}_{\rho^{\pi}(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] \tag{2.2}$$

In this paradigm, the MDP definition usually includes the discount factor, so that $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, p, \gamma)$.

2.1.2 Receding Horizon Planning

Here, we instead compute optimal actions on the fly by planning over a finite horizon. We introduce a *planning horizon* $H \in \mathbb{N}$, and we define $\hat{\tau}_t = (\hat{s}_t, \hat{a}_t, \ldots, \hat{s}_{t+H-1}, \hat{a}_{t+H-1}, \hat{s}_{t+H})$ as a *lookahead trajectory at time step* t. We define $\hat{\pi}_t \in \Delta(\mathcal{A}^H)$ as an open-loop *lookahead policy*. Given our system is at state s_t at time step t, we define the lookahead trajectory distribution $\hat{\rho}^{\hat{\pi}_t}$ as:

$$\hat{\rho}^{\hat{\pi}_t}(\hat{\tau}_t \mid \hat{s}_t = s_t) = \hat{\pi}_t(\hat{a}_t, \ldots, \hat{a}_{t+H-1}) \prod_{h=0}^{H-1} p(\hat{s}_{t+h+1} | \hat{s}_{t+h}, \hat{a}_{t+h}).$$

Under this formulation, we optimize the following problem at each time step t (and corresponding state s_t):

$$\max_{\hat{\pi}_t} \quad \mathbb{E}_{\hat{\rho}^{\hat{\pi}_t}(\hat{\tau}_t | \hat{s}_t = s_t)}\left[\sum_{h=0}^{H-1} r(\hat{s}_{t+h}, \hat{a}_{t+h}) + r_{\text{term}}(\hat{s}_{t+H})\right] \tag{2.3}$$

where r_{term} a terminal reward function. For example, the terminal reward can be designed to estimate rewards past the planning horizon.

Let $\hat{\pi}_t^\star$ be the optimizer. For control, we then sample an action sequence $(\hat{a}_t, \ldots, \hat{a}_{t+H-1})$ from $\hat{\pi}_t^\star$ and set $a_t = \hat{a}_t$ (i.e., perform action $\hat{a}_t$).

2.2 Concepts Related to Probability

2.2.1 KL Divergence

For probability distributions p and q over a common random variable x satisfying $q(x) = 0 \implies p(x) = 0$, the *KL divergence* is defined as:

$$\mathrm{KL}(p(x) \,\|\, q(x)) = \mathbb{E}_{p(x)}\left[\log \frac{p(x)}{q(x)}\right]. \tag{2.4}$$

Intuitively, the KL divergence measures the distance between p and q. It's important to note, however, that it is not a metric since it is not necessarily symmetric, i.e., $\mathrm{KL}(p(x) \,\|\, q(x)) \neq \mathrm{KL}(q(x) \,\|\, p(x))$, in general. One important property of the KL divergence is that it is always non-negative and is zero if and only if $p = q$.

2.2.2 Importance Sampling

Importance sampling is a Monte Carlo technique which allows for estimating an expectation under one distribution while sampling from a different distribution. Given some function f and distributions p and q satisfying $q(x) = 0 \implies p(x)f(x) = 0$, we have the following:

$$\mathbb{E}_{p(x)}[f(x)] = \mathbb{E}_{q(x)}\left[\frac{p(x)}{q(x)} f(x)\right] \approx \frac{1}{K} \sum_{k=1}^{K} \frac{p(x_k)}{q(x_k)} f(x_k) \qquad (x_1, \ldots, x_K \overset{\text{iid}}{\sim} q). \tag{2.5}$$

By properly setting q, we can get lower variance estimates of $\mathbb{E}_{p(x)}[f(x)]$ by sampling from q instead of p.

Part I

Model Predictive Control

CHAPTER 3

PRELIMINARIES

First, we make the following modifications to the notation established in Section 2.1 to be more in line with notation commonly used in the model predictive control community:

- We use $\mathcal{X}$ for the state space and x for the state. We also assume that $\mathcal{X} \subseteq \mathbb{R}^n$ for some $n \in \mathbb{N}$.
- We use the terms *control space* and *control* in place of action space and action, respectively. We also use $\mathcal{U}$ to denote control space and u to denote controls. We assume that $\mathcal{U} \subseteq \mathbb{R}^m$ for some $m \in \mathbb{N}$.
- We use the term *cost* in place of reward. We also use c to denote a cost function. In general, a cost function is treated as the negative of a given reward function.

For purposes of clarity, when involving planning horizons, we use lightface to denote variables that are meant for a single time step and boldface to denote variables congregated across the planning horizon. For example, we use $\hat{u}_t$ to denote the planned action at time step t and $\hat{\boldsymbol{u}}_t \triangleq (\hat{u}_t, \dots, \hat{u}_{t+H-1})$ to denote an H-step planned control sequence starting from time step t. We use a subscript to extract elements from a congregated variable. For example, we use $\hat{u}_{t,h}$ to denote the h^{th} element in $\hat{\boldsymbol{u}}_t$. Note that the subscript index starts from zero, meaning that $\hat{u}_{t,h} = \hat{u}_{t+h}$.

As opposed to maximizing the average reward as given by Eq. (2.1), we instead seek to *minimize* the average cost:

$$\min_{\pi} \quad \lim_{T \to \infty} \mathbb{E}_{\rho^{\pi}(\tau)} \left[\frac{1}{T} \sum_{t=0}^{T-1} c(x_t, u_t) \right]. \tag{3.1}$$

One other modification we will make is that we assume access to an *approximate* tran-

sition kernel $\hat{p}$ that models the true transition kernel p, i.e., $\hat{p} \approx p$. Given some state x_t and control sequence $\hat{\boldsymbol{u}}_t = (\hat{u}_t, \ldots, \hat{u}_{t+H-1}) \in \boldsymbol{\mathcal{U}}$ (where $\boldsymbol{\mathcal{U}} \triangleq \mathcal{U}^H$), we shall find it useful to reason about the predicted distribution of future states $\hat{\boldsymbol{x}}_t \triangleq (\hat{x}_t, \ldots, \hat{x}_{t+H})$ based on $\hat{p}$, which is defined as:

$$\hat{p}(\hat{\boldsymbol{x}}_t \mid \hat{x}_t = x_t, \hat{\boldsymbol{u}}_t) \triangleq \prod_{h=0}^{H-1} \hat{p}(\hat{x}_{t+h+1} \mid \hat{x}_{t+h}, \hat{u}_{t+h}). \tag{3.2}$$

Given this rollout distribution, we define the *rollout cost* $\hat{C}_t$ at time step t under the control sequence $\hat{\boldsymbol{u}}_t$ as:

$$\hat{C}_t(\hat{\boldsymbol{u}}_t) = \hat{C}(x_t, \hat{\boldsymbol{u}}_t) \triangleq \mathbb{E}_{\hat{p}(\hat{\boldsymbol{x}}_t \mid \hat{x}_t = x_t, \hat{\boldsymbol{u}}_t)} \left[\sum_{h=0}^{H-1} c(\hat{x}_{t+h}, \hat{u}_{t+h}) + c_{\mathrm{term}}(\hat{x}_{t+H}) \right], \tag{3.3}$$

where c_{term} is a terminal cost function, analagous to the terminal reward function from Eq. (2.3).

CHAPTER 4
INFORMATION-THEORETIC MODEL PREDICTIVE CONTROL FOR MODEL-BASED REINFORCEMENT LEARNING

We introduce an information-theoretic model predictive control (MPC) algorithm capable of handling complex cost criteria and general nonlinear dynamics. The generality of the approach makes it possible to use multi-layer neural networks as dynamics models, which we incorporate into our MPC algorithm in order to solve model-based reinforcement learning tasks. We test the algorithm in simulation on a cartpole swing-up and quadrotor navigation task, as well as on actual hardware in an aggressive driving task. Empirical results demonstrate that the algorithm is capable of achieving a high level of performance and does so only utilizing data collected from the system. A video of the aggressive driving experiments is available at `https://www.youtube.com/watch?v=f2at-cqaJMM`.

4.1 Introduction

Many robotic tasks can be phrased as reinforcement learning (RL) problems, where a robot seeks to optimize a cost function encoding a task by utilizing data collected from the system. The types of reinforcement learning problems encountered in robotic tasks are frequently continuous state-action and high dimensional (Kober, Bagnell, et al., 2013). The methods for solving these problems are often categorized into model-free and model-based approaches.

Model-free approaches to RL such as policy gradient methods have been successfully applied to many challenging tasks (Peters and Schaal, 2006a; Benbrahim and Franklin, 1997; Kober and Peters, 2008; Kormushev et al., 2010; Peters and Schaal, 2006b; Peters and Schaal, 2008b; Peters and Schaal, 2008a; E. Theodorou et al., 2010; Buchli et al., 2011; Stulp et al., 2012). These approaches typically require an expert demonstration to

Figure 4.1: Aggressive driving with MPPI and neural network dynamics.

initialize the learning process, followed by many interactions with the actual robotic system. Unfortunately, model-free approaches often require a large amount of data from these interactions, which limits their applicability. Additionally, while optimization of the initial demonstrated policy leads to improved task performance, in the most popular gradient-based appraches the resulting solution remains within the envelope of the initially demonstrated policy. This limits the method's ability to discover novel optimal control behaviors.

In the second paradigm, model-based RL approaches first learn a model of the system and then train a feedback control policy using the learned model (Abbeel et al., 2010; Schaal, 1996; C. G. Atkeson and Santamaria, 1997). Other techniques for model-based reinforcement learning incorporate trajectory optimization with model learning (Mitrovic et al., 2010) or disturbance learning (Morimoto and C. Atkeson, 2002). This means that interactions with the robotic system must be performed at every iteration of the trajectory optimization algorithm.

Despite all the progress on both model-based and model-free RL methods, generalization remains a primary challenge. Robotic systems that operate in changing and stochastic environments must adapt to new situations and be equipped with fast decision making processes. Model predictive control (MPC) or receding horizon control tackles this problem by relying on online optimization of the cost function and is one of the most effective ways to achieve generalization for RL tasks. However, most variations of MPC rely on tools

from constrained optimization, which means that convexification (such as with quadratic approximation) of the cost function and first- or second-order approximations of the dynamics are required.

A more flexible MPC method is model predictive path integral (MPPI) control, a sampling-based algorithm which can optimize for general cost criteria, convex or not (Williams, Drews, et al., 2016; Williams, Aldrich, et al., 2017; Gómez et al., 2016). However, in prior work, MPPI could only be applied to systems with control affine dynamics. In this paper, we extend the method so that it is applicable to a larger class of stochastic systems and representations. In particular, we demonstrate how the update law used in MPPI can be derived through an information-theoretic framework, *without* making the control affine assumption as done by Williams, Drews, et al. (2016). This is a significant step forward because it enables a purely data-driven approach to model learning within the MPPI framework. We use a multi-layer neural network to approximate the system dynamics and demonstrate the ability of MPPI to perform difficult real-time control tasks using the approximate system model. We test the MPPI controller in simulation, using purely learned neural network dynamics, on a simulated cartpole swing-up task and a quadrotor obstacle navigation task. Simulation results demonstrate that the controller performs comparably to an "ideal" MPPI controller, which we define as the MPPI controller which has access to the actual simulation dynamics. To further demonstrate the practicality and effectiveness of the approach, we demonstrate it on real hardware in an aggressive driving task with the Georgia Tech AutoRally platform and obtain comparable results to MPPI with a hand-designed physics-based vehicle model used by Williams, Drews, et al. (2016).

4.2 Model Predictive Control

The theory for model predictive control for linear systems is well understood and has many successful applications in the process industry (Qin and Badgwell, 2003). For nonlinear systems, MPC is an increasingly active area of research in control theory (Mayne, 2014).

Despite the progress in terms of theory and successful applications, most prior work on MPC focuses on stabilization or trajectory tracking tasks. The key difference between classical MPC and MPC for reinforcement learning is that RL tasks have more general objectives beyond stabilization or tracking. The complexity of the objectives in RL tasks increases the computational cost of the optimization, a major problem since optimization must occur in real time. The most tractable approach to date is receding-horizon differential dynamic programming (Erez et al., 2013), which is capable of controlling complex animated characters in realistic physics simulators, albeit using a ground truth model. The fusion of learned system models with the type of generalized MPC necessary for solving RL problems is a new area of research.

The information-theoretic MPC algorithm that we develop is originally based on path integral control theory. In its traditional form, path integral control involves taking an exponential transform of the value function of an optimal control problem and then applying the Feynman-Kac formula to express the solution to the Hamilton-Jacobi-Bellman partial differential equation in terms of an expectation over all possible system paths. To make this transformation, the dynamics must be affine in controls and satisfy a special relationship between noise and controls. E. A. Theodorou and Todorov (2012) connect this approach to the information-theoretic notions of free energy and relative entropy (also known as KL divergence), which was then exploited by Williams, Drews, et al. (2016) to derive a slightly generalized path integral expression. Here, we take this one step further, and completely remove the control affine requirement. The resulting derivation and update law are closely related to the cross-entropy method (W. Zhang et al., 2014), policy improvement with path integrals (E. Theodorou et al., 2010), and reward-weighted regression (Peters and Schaal, 2007). Although those algorithms are geared towards updating the parameters of a feedback control policy, we focus on optimizing an open-loop control sequence for use with MPC.

4.3 Information-Theoretic Control

In this section, we introduce the theoretical basis for our sampling-based MPC algorithm, which relies on the concept of KL divergence drawn from information theory. The result of the derivation is an expression for an optimal control law in terms of a weighted average over sampled trajectories. This leads to a gradient-free update law which is highly parallelizable, making it ideal for online computation.

4.3.1 Objective Function

Suppose that at time step t the system is at state x_t. Letting $\hat{C}_t$ be the rollout cost as defined in Eq. (3.3), we seek a lookahead policy $\hat{\pi}_t(\hat{\boldsymbol{u}}_t)$ that optimizes the following objective:

$$\min_{\hat{\pi}_t} \quad \mathbb{E}_{\hat{\pi}_t(\hat{\boldsymbol{u}}_t)}[\hat{C}_t(\hat{\boldsymbol{u}}_t)] + \lambda \,\mathrm{KL}(\hat{\pi}_t(\hat{\boldsymbol{u}}_t) \,\|\, p_t(\hat{\boldsymbol{u}}_t)), \tag{4.1}$$

where $p_t(\hat{\boldsymbol{u}}_t)$ is a given prior control distribution (e.g., zero mean Gaussian). The KL divergence term attempts to keep the control distribution close to the prior and is a common regularization choice in the stochastic optimal control literature (E. A. Theodorou and Todorov, 2012; Todorov, 2006; E. A. Theodorou, 2015). We can analytically solve this problem by turning the objective into a KL divergence:

$$\begin{aligned}
\mathbb{E}_{\hat{\pi}_t(\hat{\boldsymbol{u}}_t)}[\hat{C}_t(\hat{\boldsymbol{u}}_t)] + \lambda \,\mathrm{KL}(\hat{\pi}_t(\hat{\boldsymbol{u}}_t) \,\|\, p_t(\hat{\boldsymbol{u}}_t)) &= \mathbb{E}_{\hat{\pi}_t(\hat{\boldsymbol{u}}_t)}\left[\hat{C}_t(\hat{\boldsymbol{u}}_t) + \lambda \log \frac{\hat{\pi}_t(\hat{\boldsymbol{u}}_t)}{p_t(\hat{\boldsymbol{u}}_t)}\right] \\
&= \mathbb{E}_{\hat{\pi}_t(\hat{\boldsymbol{u}}_t)}\left[-\lambda \log e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda} + \lambda \log \frac{\hat{\pi}_t(\hat{\boldsymbol{u}}_t)}{p_t(\hat{\boldsymbol{u}}_t)}\right] \\
&= \lambda \,\mathbb{E}_{\hat{\pi}_t(\hat{\boldsymbol{u}}_t)}\left[\log \frac{\hat{\pi}_t(\hat{\boldsymbol{u}}_t)}{p_t(\hat{\boldsymbol{u}}_t) e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}}\right] \\
&= \lambda \,\mathrm{KL}(\hat{\pi}_t(\hat{\boldsymbol{u}}_t) \,\|\, \tfrac{1}{Z_t} p_t(\hat{\boldsymbol{u}}_t) e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}) - \lambda \log Z_t,
\end{aligned}$$

where $Z_t = \displaystyle\int_{\boldsymbol{\mathcal{U}}} p_t(\hat{\boldsymbol{u}}) e^{-\hat{C}_t(\hat{\boldsymbol{u}})/\lambda} \,\mathrm{d}\hat{\boldsymbol{u}} = \mathbb{E}_{p_t(\hat{\boldsymbol{u}})}[e^{-\hat{C}_t(\hat{\boldsymbol{u}})/\lambda}]$ is a normalizer. The term $-\lambda \log Z_t$ is a constant, so we can minimize the objective by making the two distributions in the KL

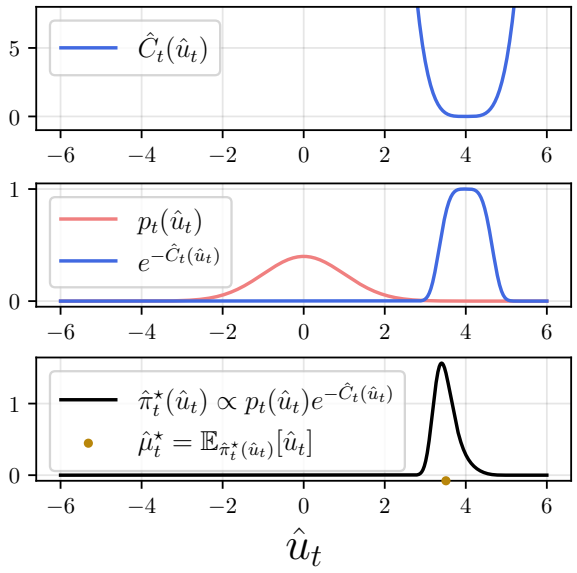


Figure 4.2: Toy example illustrating an optimal distribution from Eq. (4.2). Here, we have $\hat{C}_t(\hat{u}_t) = 4(\hat{u}_t - 4)^4$, $p_t(\hat{u}_t) = \mathcal{N}(\hat{u}_t; 0, 1)$, and $\lambda = 1$.

divergence match. Therefore, the optimal lookahead policy $\hat{\pi}_t^\star$ is:

$$\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t) = \frac{p_t(\hat{\boldsymbol{u}}_t)e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}}{\mathbb{E}_{p_t(\hat{\boldsymbol{u}})}[e^{-\hat{C}_t(\hat{\boldsymbol{u}})/\lambda}]}. \tag{4.2}$$

A simple 1-D example of an optimal policy is shown in Fig. 4.2.

However, it is intractable to sample from this policy, and approximate sampling techniques such as Markov chain Monte Carlo methods (Brooks et al., 2011) are too slow for the control setting.

4.3.2 KL Divergence Minimization

Instead of trying to sample from $\hat{\pi}_t^\star$, we consider a simpler class of lookahead policies and find the one closest to the optimal lookahead policy. In particular, we consider a lookahead policy $\hat{\pi}_{\boldsymbol{\mu}_t}(\hat{\boldsymbol{u}}_t)$ paramaterized by $\hat{\boldsymbol{\mu}}_t \in \mathbb{R}^{Hm}$ which defines a Gaussian distribution with fixed covariance $\boldsymbol{\Sigma} \succ 0$, i.e., $\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t) = \mathcal{N}(\hat{\boldsymbol{u}}_t; \hat{\boldsymbol{\mu}}_t, \boldsymbol{\Sigma})$. We then find the mean $\hat{\boldsymbol{\mu}}_t$ that

brings $\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}$ as close as possible to $\hat{\pi}_t^\star$ in the sense of (forward) KL divergence:

$$\min_{\hat{\boldsymbol{\mu}}_t} \quad \mathrm{KL}(\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t) \,\|\, \hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)). \tag{4.3}$$

One may recognize this objective as the maximum likelihood over the class of Gaussian distributions. Using a classic result from statistics and machine learning (Wainwright and Jordan, 2008; Murphy, 2012), we conclude that our Gaussian's mean should match that of the optimal lookahead policy. In other words, $\hat{\boldsymbol{\mu}}_t = \mathbb{E}_{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}[\hat{\boldsymbol{u}}_t]$ optimizes Eq. (4.3). For completeness, we derive this result in full by expanding the KL divergence from Eq. (4.3):

$$\begin{aligned}
\mathrm{KL}(\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t) \,\|\, \hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)) &= \mathbb{E}_{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}\left[\log \frac{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\right] \\
&= \mathbb{E}_{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}[-\log \hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)] + \text{constant} \\
&= \mathbb{E}_{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}\left[\frac{1}{2}(\hat{\boldsymbol{u}}_t - \hat{\boldsymbol{\mu}}_t)^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\hat{\boldsymbol{u}}_t - \hat{\boldsymbol{\mu}}_t)\right] + \text{constant} \\
&= \mathbb{E}_{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}\left[\frac{1}{2}\hat{\boldsymbol{\mu}}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{\mu}}_t - \hat{\boldsymbol{\mu}}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{u}}_t\right] + \text{constant} \\
&= \frac{1}{2}\hat{\boldsymbol{\mu}}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{\mu}}_t - \hat{\boldsymbol{\mu}}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\,\mathbb{E}_{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}[\hat{\boldsymbol{u}}_t] + \text{constant} \\
&= \frac{1}{2}(\mathbb{E}_{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}[\hat{\boldsymbol{u}}_t] - \hat{\boldsymbol{\mu}}_t)^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\mathbb{E}_{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}[\hat{\boldsymbol{u}}_t] - \hat{\boldsymbol{\mu}}_t) + \text{constant}.
\end{aligned}$$

Since $\boldsymbol{\Sigma}^{-1}$ is positive definite, this function is minimized at $\hat{\boldsymbol{\mu}}_t^\star = \mathbb{E}_{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}[\hat{\boldsymbol{u}}_t]$.

However, while we have found the optimal form of our Gaussian distribution, computing this mean requires either integrating over $\boldsymbol{\mathcal{U}}$ or sampling from $\hat{\pi}_t^\star$, both of which are intractable. We can take advantage of the form of $\hat{\pi}_t^\star$ (Eq. (4.2)) to get a more appealing

expression of the mean:

$$\begin{aligned}
\hat{\boldsymbol{\mu}}_t^\star &= \mathbb{E}_{\hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)}[\hat{\boldsymbol{u}}_t] \\
&= \int_{\boldsymbol{\mathcal{U}}} \hat{\pi}_t^\star(\hat{\boldsymbol{u}}_t)\, \hat{\boldsymbol{u}}_t \, \mathrm{d}\hat{\boldsymbol{u}}_t \\
&= \frac{\displaystyle\int_{\boldsymbol{\mathcal{U}}} p_t(\hat{\boldsymbol{u}}_t) e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}\, \hat{\boldsymbol{u}}_t \, \mathrm{d}\hat{\boldsymbol{u}}_t}{\displaystyle\int_{\boldsymbol{\mathcal{U}}} p_t(\hat{\boldsymbol{u}}_t) e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda} \, \mathrm{d}\hat{\boldsymbol{u}}_t} \\
&= \frac{\mathbb{E}_{p_t(\hat{\boldsymbol{u}}_t)}[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}\, \hat{\boldsymbol{u}}_t]}{\mathbb{E}_{p_t(\hat{\boldsymbol{u}}_t)}[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}]}.
\end{aligned} \tag{4.4}$$

Thus, we can estimate $\hat{\boldsymbol{\mu}}_t^\star$ by sampling K sequences $\hat{\boldsymbol{u}}_t^1, \ldots, \hat{\boldsymbol{u}}_t^K$ from $p_t(\hat{\boldsymbol{u}}_t)$ and using empirical expectations in Eq. (4.4):

$$\hat{\boldsymbol{\mu}}_t^\star \approx \frac{\displaystyle\sum_{k=1}^{K} e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t^k)/\lambda}\, \hat{\boldsymbol{u}}_t^k}{\displaystyle\sum_{k=1}^{K} e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t^k)/\lambda}} \qquad (\hat{\boldsymbol{u}}_t^1, \ldots, \hat{\boldsymbol{u}}_t^K \overset{\text{iid}}{\sim} p_t). \tag{4.5}$$

We recognize this as a "softmin" over the rollout costs to find the best trajectory. Thus, with enough samples from p_t, we can imagine that there will be some sample $\hat{\boldsymbol{u}}_t^{k^\star}$ with low rollout cost $\hat{C}_t(\hat{\boldsymbol{u}}_t^{k^\star})$ that will therefore be given a large softmin weight. However, if p_t does not cover low-cost areas, then most samples will have large rollout costs. Referring to the toy example in Fig. 4.2, note that the low-cost region lies in the tail of p_t so that it's sampled with very low probability. This causes the estimate from Eq. (4.4) to have high variance, thus requiring an enourmous number of samples from p_t to get an accurate estimate (Williams, Aldrich, et al., 2017).

4.3.3 Importance Sampling

One powerful technique for estimating an expectation is importance sampling (Section 2.2.2), a Monte Carlo technique where we sample from a different distribution and modulate the

samples by probability ratios. Given some distribution $q_t(\hat{\boldsymbol{u}}_t)$, we have the following expression for $\hat{\boldsymbol{\mu}}_t^\star$ after applying importance sampling to Eq. (4.4):

$$\hat{\boldsymbol{\mu}}_t^\star = \frac{\mathbb{E}_{q_t(\hat{\boldsymbol{u}}_t)}\left[\frac{p_t(\hat{\boldsymbol{u}}_t)}{q_t(\hat{\boldsymbol{u}}_t)} e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}\, \hat{\boldsymbol{u}}_t\right]}{\mathbb{E}_{q_t(\hat{\boldsymbol{u}}_t)}\left[\frac{p_t(\hat{\boldsymbol{u}}_t)}{q_t(\hat{\boldsymbol{u}}_t)} e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}\right]}. \tag{4.6}$$

We make the practical choice for q_t to be the Gaussian distribution $\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}$ we are already using and solve Eq. (4.3) to have a low variance estimate of $\hat{\boldsymbol{\mu}}_t^\star$. In practice, because we can only approximate Eq. (4.6) via sampling, we opt to instead *improve* our mean $\hat{\boldsymbol{\mu}}_t$ towards $\hat{\boldsymbol{\mu}}_t^\star$. So by setting $q_t = \hat{\pi}_{\hat{\boldsymbol{\mu}}_t}$, we have

$$\hat{\boldsymbol{\mu}}_t^\star = \frac{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\left[\frac{p_t(\hat{\boldsymbol{u}}_t)}{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)} e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}\, \hat{\boldsymbol{u}}_t\right]}{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\left[\frac{p_t(\hat{\boldsymbol{u}}_t)}{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)} e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}\right]}. \tag{4.7}$$

Now, we instantiate our prior distribution $p_t(\hat{\boldsymbol{u}}_t)$ to be a Gaussian with covariance $\boldsymbol{\Sigma}$, i.e., $p_t(\hat{\boldsymbol{u}}_t) = \mathcal{N}(\hat{\boldsymbol{u}}_t; \boldsymbol{\mu}_t, \boldsymbol{\Sigma})$ for some $\boldsymbol{\mu}_t \in \mathbb{R}^{mH}$. Under this choice of prior, the importance sampling weight is:

$$\begin{aligned}
\frac{p_t(\hat{\boldsymbol{u}}_t)}{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)} &= \frac{\exp(-\frac{1}{2}(\hat{\boldsymbol{u}}_t - \boldsymbol{\mu}_t)^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\hat{\boldsymbol{u}}_t - \boldsymbol{\mu}_t))}{\exp(-\frac{1}{2}(\hat{\boldsymbol{u}}_t - \hat{\boldsymbol{\mu}}_t)^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\hat{\boldsymbol{u}}_t - \hat{\boldsymbol{\mu}}_t))} \\
&= \exp\big((\boldsymbol{\mu}_t - \hat{\boldsymbol{\mu}}_t)^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{u}}_t + \tfrac{1}{2}\hat{\boldsymbol{\mu}}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{\mu}}_t - \tfrac{1}{2}\boldsymbol{\mu}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_t\big).
\end{aligned}$$

Therefore, the mean is:

$$\begin{aligned}
\hat{\boldsymbol{\mu}}_t^\star &= \frac{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\big[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda + (\boldsymbol{\mu}_t - \hat{\boldsymbol{\mu}}_t)^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{u}}_t + \frac{1}{2}\hat{\boldsymbol{\mu}}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{\mu}}_t - \frac{1}{2}\boldsymbol{\mu}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_t}\, \hat{\boldsymbol{u}}_t\big]}{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\big[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda + (\boldsymbol{\mu}_t - \hat{\boldsymbol{\mu}}_t)^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{u}}_t + \frac{1}{2}\hat{\boldsymbol{\mu}}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{\mu}}_t - \frac{1}{2}\boldsymbol{\mu}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_t}\big]} \\
&= \frac{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\big[e^{-(\hat{C}_t(\hat{\boldsymbol{u}}_t) + \lambda(\hat{\boldsymbol{\mu}}_t - \boldsymbol{\mu}_t)^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{u}}_t)/\lambda}\, \hat{\boldsymbol{u}}_t\big]}{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\big[e^{-(\hat{C}_t(\hat{\boldsymbol{u}}_t) + \lambda(\hat{\boldsymbol{\mu}}_t - \boldsymbol{\mu}_t)^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{u}}_t)/\lambda}\big]},
\end{aligned}$$

where we cancel out $e^{\frac{1}{2}\hat{\boldsymbol{\mu}}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{\mu}}_t - \frac{1}{2}\boldsymbol{\mu}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_t}$ in the second line since it is a constant. For

clarity, we define $\hat{C}'_t(\hat{\boldsymbol{u}}_t) \triangleq \hat{C}_t(\hat{\boldsymbol{u}}_t) + \lambda(\hat{\boldsymbol{\mu}}_t - \boldsymbol{\mu}_t)^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{u}}_t$, so that:

$$\hat{\boldsymbol{\mu}}_t^\star = \frac{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\big[e^{-\hat{C}'_t(\hat{\boldsymbol{u}}_t)/\lambda}\,\hat{\boldsymbol{u}}_t\big]}{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\big[e^{-\hat{C}'_t(\hat{\boldsymbol{u}}_t)/\lambda}\big]}.$$

Thus, by sampling K sequences $\hat{\boldsymbol{u}}_t^1, \ldots, \hat{\boldsymbol{u}}_t^K$ from $\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}$, we get the following approximation:

$$\begin{aligned}\hat{\boldsymbol{\mu}}_t^\star &\approx \frac{\displaystyle\sum_{k=1}^{K} e^{-\hat{C}'_t(\hat{\boldsymbol{u}}_t^k)/\lambda}\,\hat{\boldsymbol{u}}_t^k}{\displaystyle\sum_{k=1}^{K} e^{-\hat{C}'_t(\hat{\boldsymbol{u}}_t^k)/\lambda}} \qquad\qquad (\hat{\boldsymbol{u}}_t^1, \ldots, \hat{\boldsymbol{u}}_t^K \overset{\text{iid}}{\sim} \hat{\pi}_{\hat{\boldsymbol{\mu}}_t}) \\ &= \sum_{k=1}^{K} w_k \hat{\boldsymbol{u}}_t^k,\end{aligned}$$

where $w_k = e^{-\hat{C}'_t(\hat{\boldsymbol{u}}_t^k)/\lambda} \Big/ \sum_{\ell=1}^{K} e^{-\hat{C}'_t(\hat{\boldsymbol{u}}_t^\ell)/\lambda}$. Thus, we have an iterative scheme to improve some given mean $\hat{\boldsymbol{\mu}}_t$ to one that is closer to the optimal $\hat{\boldsymbol{\mu}}_t^\star$.

Prior Distribution

It's also important to touch on the choice of mean $\boldsymbol{\mu}_t$ for the prior distribution p_t. Two example choices are:

- $\boldsymbol{\mu}_t = 0$: Under this choice of prior mean, the update becomes:

$$\hat{\boldsymbol{\mu}}_t^\star = \frac{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\big[e^{-(\hat{C}_t(\hat{\boldsymbol{u}}_t)+\lambda\hat{\boldsymbol{\mu}}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{u}}_t)/\lambda}\,\hat{\boldsymbol{u}}_t\big]}{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\big[e^{-(\hat{C}_t(\hat{\boldsymbol{u}}_t)+\lambda\hat{\boldsymbol{\mu}}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}\hat{\boldsymbol{u}}_t)/\lambda}\big]}. \tag{4.8}$$

  The underlying optimal control problem (Eq. (4.1)) seeks to regularize $\hat{\boldsymbol{\mu}}_t^\star$ towards zero.

- $\boldsymbol{\mu}_t = \hat{\boldsymbol{\mu}}_t$: Under this choice of prior mean, the update becomes:

$$\hat{\boldsymbol{\mu}}_t^\star = \frac{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\big[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}\,\hat{\boldsymbol{u}}_t\big]}{\mathbb{E}_{\hat{\pi}_{\hat{\boldsymbol{\mu}}_t}(\hat{\boldsymbol{u}}_t)}\big[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}\big]}. \tag{4.9}$$

In particular, this update does *not* use importance sampling, and the underlying optimal control problem (Eq. (4.1)) seeks to keep $\hat{\boldsymbol{\mu}}_t^\star$ close to the previous solution $\hat{\boldsymbol{\mu}}_t$. In light of this, we call Eq. (4.9) the *proximal MPPI update rule*.

4.3.4 MPPI Algorithm

We present two versions of computing the rollout cost $\hat{C}_t$: one with a deterministic transition model (Algorithm 1) and another with a stochastic transition model (Algorithm 2). When using a deterministic transition model (i.e., $\hat{x}_{t+1} = \hat{f}(\hat{x}_t, \hat{u}_t)$ for some function $\hat{f}$), the corresponding transition distribution $\hat{p}$ is a Dirac delta function, so we only need to query the model once for each sample $\hat{\boldsymbol{u}}_t^k$. When using a stochastic transition model $\hat{p}$, we must estimate $\hat{C}_t(\hat{\boldsymbol{u}}_t)$ via Monte Carlo, i.e., by propagating $\hat{\boldsymbol{u}}_t^k$ through the model L times and taking an empirical average of the rollout cost.

We give the pseudocode for MPPI in Algorithm 3. For the rest of this chapter (including the experiments), we opt for a deterministic transition model (Algorithm 1) and the update rule in Eq. (4.8).

4.4 MPC with Neural Network Dynamics

To deploy Algorithm 3, we need a model to sample from. In the model-based RL setting, this means learning a model from data. In this section, we describe our learning procedure and the real-time implementation of MPPI.

4.4.1 Learning Neural Network Models

The kinematics for our systems of interest are trivial given the velocities, so we need only learn the acceleration of each system. Specifically, given that the state x is partitioned as $x = (q, \dot{q})$, where q is the configuration of the system and $\dot{q}$ is its time derivative, we seek

Algorithm 1: Rollout cost (deterministic model)

Given: x: Initial state for rollout
$\hat{\boldsymbol{u}} = (\hat{u}_0, \ldots, \hat{u}_{H-1})$: Control seq.
$\hat{f}$: Deterministic transition model
c: Step cost function
c_{term}: Terminal cost function
Set $\hat{x}_0 = x$.
Set $\hat{x}_{h+1} = \hat{f}(\hat{x}_h, \hat{u}_h)$ for $h = 0, \ldots, H-1$.
Set $\hat{C} = \sum_{h=0}^{H-1} c(\hat{x}_h, \hat{u}_h) + c_{\text{term}}(\hat{x}_H)$.
return $\hat{C}$

Algorithm 2: Rollout cost (stochastic model)

Given: x: Initial state for rollout
$\hat{\boldsymbol{u}} = (\hat{u}_0, \ldots, \hat{u}_{H-1})$: Control seq.
$\hat{p}$: Stochastic transition model
L: Number of samples from $\hat{p}$
c: Step cost function
c_{term}: Terminal cost function
for $\ell = 1, \ldots, L$ **do**
Set $\hat{x}_0^\ell = x$.
Sample $\hat{x}_{h+1}^\ell \sim \hat{p}(\hat{x}_h^\ell, \hat{u}_h)$ for $h = 0, \ldots, H-1$.
Set $\hat{C}_\ell = \sum_{h=0}^{H-1} c(\hat{x}_h^\ell, \hat{u}_h) + c_{\text{term}}(\hat{x}_H^\ell)$.
end
Set $\hat{C} = \frac{1}{L} \sum_{\ell=1}^{L} \hat{C}_\ell$.
return $\hat{C}$

Algorithm 3: Model predictive path integral (MPPI)

Given: $\hat{C}$: Rollout cost function (Algorithm 1 or Algorithm 2)
K: Number of sampled control sequences
H: Number of planning timesteps
$\boldsymbol{\Sigma}$: Sampling covariance
λ: Temperature
Initialize $\hat{\boldsymbol{u}} = (\hat{u}_0, \ldots, \hat{u}_{H-1})$ (e.g., all zeros).
while task not completed **do**
Set x to be system's state.
Set $\boldsymbol{u}$ to be the prior mean for this time step. (Section 4.3.3)
for $k = 1, \ldots, K$ **do**
Sample $\hat{\boldsymbol{u}}_k \sim \mathcal{N}(\hat{\boldsymbol{u}}, \boldsymbol{\Sigma})$.
Set $C_k = \hat{C}(x, \hat{\boldsymbol{u}}_k) + \lambda(\hat{\boldsymbol{u}} - \boldsymbol{u})^\mathsf{T} \boldsymbol{\Sigma}^{-1} \hat{\boldsymbol{u}}_k$.
end
Set $w_k = e^{-C_k/\lambda} \Big/ \sum_{\ell=1}^{K} e^{-C_k/\lambda}$ for $k = 1, \ldots, K$.
Set $\hat{\boldsymbol{u}} = \sum_{k=1}^{K} w_k \hat{\boldsymbol{u}}_k$.
Set $u = \hat{u}_0$ and send u to control system.
Shift control sequence $\hat{\boldsymbol{u}} = (\hat{u}_0, \ldots, \hat{u}_{H-1})$ forward one time step.
end

Algorithm 4: MPPI with dynamics model training

Input: task
N: Iterations
M: Trials per iteration
Set $\mathcal{D}$ to be a dynamics dataset collected from bootstrap data (or empty if there's no bootstrap data).
for $i = 1, \ldots, N$ **do**
  Set $\hat{f}$ to be dynamics model trained on $\mathcal{D}$.
  for $j = 1, \ldots, M$ **do**
    Collect dynamics dataset $\mathcal{D}_j$ by applying MPPI with model $\hat{f}$ to the task.
    Aggregate $\mathcal{D}_j$ into $\mathcal{D}$.
  end
end

a function $\ddot{q}(x, u)$ so that the full state transition is:

$$x_{t+1} = f(x_t, u_t) = \begin{bmatrix} q_t + \dot{q}_t \Delta t \\ \dot{q}_t + \ddot{q}(x_t, u_t)\Delta t \end{bmatrix},$$

where Δt is a discrete time increment. We approximate $\ddot{q}$ with a neural network $\hat{\ddot{q}}$ and train it on a dataset of state-control-acceleration triplets $\mathcal{D} = \{(x_t, u_t, \frac{\dot{q}_{t+1} - \dot{q}_t}{\Delta t})\}_t$ using minibatch gradient descent with the RMSprop optimizer (Tieleman and Hinton, 2012).

To create a dataset for learning the model, we follow a two-phase approach. In the first phase, we collect system identification data and then train the neural network. For simulation data, the MPPI controller with the ground truth model is run, whereas a human driver is used in real-world experiments. The ability to collect a bootstrapping dataset in this manner is one of the main benefits of model-based RL approaches: they can use data collected from *any* interaction with the system since the dynamics do not usually depend on the algorithm controlling the system or the task being performed. In the second phase, we repeatedly run the MPPI algorithm with the neural network model, augment the dataset from the system interactions, and re-train the neural network using the augmented dataset. In some cases, the initial dataset is enough to adequately perform the task. This procedure

is shown in Algorithm 4.

We perform this procedure using a range of neural network sizes in order to determine the effect of network configuration on control performance. Table 4.1 describes the network configurations that we use in our simulations and experiments.

Table 4.1: Layer sizes and activations of models

System	Layer 1 Size	Layer 2 Size	Activation
cartpole	16	16	tanh
cartpole	32	32	tanh
cartpole	64	64	tanh
quadrotor	16	16	tanh
quadrotor	32	32	tanh
quadrotor	64	64	tanh
AutoRally	32	32	tanh

4.4.2 Practical Details

In MPC, optimization and execution take place simultaneously: a control sequence is computed, and then the first element of the sequence is executed. This process is repeated using the un-executed portion of the previous control sequence as the importance sampling trajectory for the next iteration. The key requirement for sampling-based MPC is to produce a large number of samples in real time. As with Williams, Drews, et al. (2016), we perform sampling in parallel on an Nvidia GPU using custom-written and highly optimized CUDA kernels.

The use of neural networks as models makes sampling in real-time considerably more involved because forward propagation of the network can be expensive, and this operation must be performed KH times. For example, the dynamics model for the AutoRally vehicle by Williams, Drews, et al. (2016) consists of 100 parameters and a single matrix multiply, whereas the neural network model that we learn for the AutoRally task has 1412 parameters and consists of successive large matrix multiplications and non-linearities. To make this tractable we take advantage of the parallel nature of neural networks and further parallelize

the algorithm by using multiple CUDA threads per trajectory sample.

4.5 Experimental Results

We tested our approach both in simulation and in a real-world aggressive driving task. For all our experiments, the sampling covariance has the form $\mathbf{\Sigma} = \mathrm{diag}(\Sigma, \ldots, \Sigma)$ for some $m \times m$ covariance matrix Σ. This corresponds to using temporally independent Gaussian distributions.

4.5.1 Simulated Tasks

In simulation, we tested our approach on a cartpole swing-up and quadrotor navigation task. In these simulated scenarios, a convenient benchmark is the MPPI algorithm with access to the ground-truth model used for the simulation. This provides a metric for how much performance is lost using an approximate model of the system.

Cartpole Swing-Up

In this task, the controller has to swing and hold a pendulum upright using only actuation on the attached cart, starting with the pendulum oriented downwards. The full system definition is given in Section A.1. We set the system noise to $\Sigma = 0.9^2$ and the temperature to $\lambda = 1$. The bootstrapping dataset for the cartpole comes from 5 minutes of multiple MPPI demonstrations using known dynamics but a different cost function for the swing-up task. These system identification trajectories show the cartpole's behavior when the pole is upright, but they don't exhaust enough of the state-action space for the MPPI controller to act correctly immediately. The cartpole is of low enough dimensionality that no bootstrap dataset is required to perform the task, though at the cost of more training iterations. The relative trajectory costs are shown in Fig. 4.3, where each iteration consists of one 10 second trial.

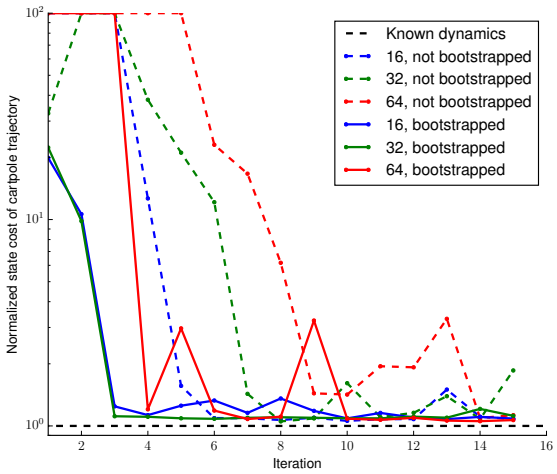


Figure 4.3: Normalized state costs of executed cartpole trajectories. The cost is normalized so that the ground-truth MPPI controller has a cost of 1. Average costs are computed from ten trials. Note the logarithmic scale and that relative costs are clamped to a maximum of 100.

Quadrotor Navigation

For this task, a quadrotor must fly from one corner of a field to the other while avoiding circular obstacles. The system and problem description are given in Section A.2. We set the temperature to $\lambda = 1$ and the sampling covariance to $\Sigma = \mathrm{diag}(2.5^2, 0.25^2, 0.25^2, 0.25^2)$, where the 2.5^2 value corresponds to the thrust input. We found that running the algorithm without bootstrapping the neural network dynamics resulted in repeated failures. We bootstraped the neural network with 30 minutes of an MPPI demonstration with known dynamics and a moving target but no obstacles.

All network models yield similar results, as shown in Fig. 4.4. The bootstrap data is enough for the MPPI controller with the medium-sized network to navigate the field. However, the smallest and largest networks require an extra iteration to become competent at the task. After one iteration, the algorithm achieves the same level of performance regardless of which network is being used. An example trajectory successfully navigating the field is also shown in Fig. 4.4.

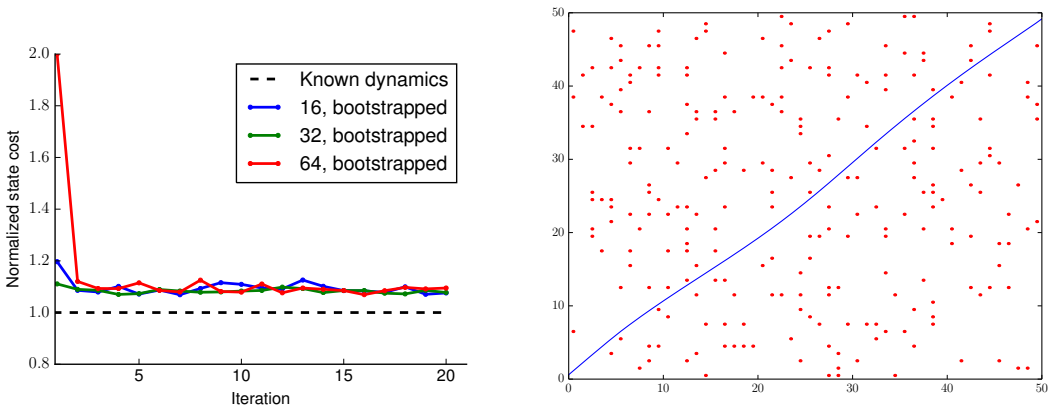


Figure 4.4: Left: Normalized state costs of executed quadrotor trajectories. The costs are normalized so that the controller with the ground-truth model has a cost of 1, the cost is bounded at 2. There are five trials per iteration. Right: Example run through the virtual obstacle field, units are in meters.

Multi-Step Error

We trained the neural network dynamics on one-step prediction error, which does not necessarily result in accurate multi-step prediction. In the worst case, compounding multi-step errors can grow exponentially (Venkatraman et al., 2015). The multi-step error over the prediction horizon for cartpole is shown in Fig. 4.5. For cartpole dynamics, the smaller, bootstrapped networks perform best. Note that the worst performers on multi-step error for the cartpole directly correlate with the the worst performers on the swing-up task, as one would expect.

None of the networks for the quadrotor dynamics perform significantly better or worse in multi-step error, which is reflected in the near identical performance of the MPPI controller with each of the three networks. The final positional and orientation errors after the 2.5 second prediction horizon are approximately 1.5 meters and 0.4 radians, respectively.

Mitigating the build up of model error is a primary challenge in model-based RL. MPC has two characteristics which help in this regard. The first is that it only requires a short time horizon, and the second is that it constantly recomputes the planned control sequence. The final performance margins for both the cartpole and quadrotor are within 10% of what

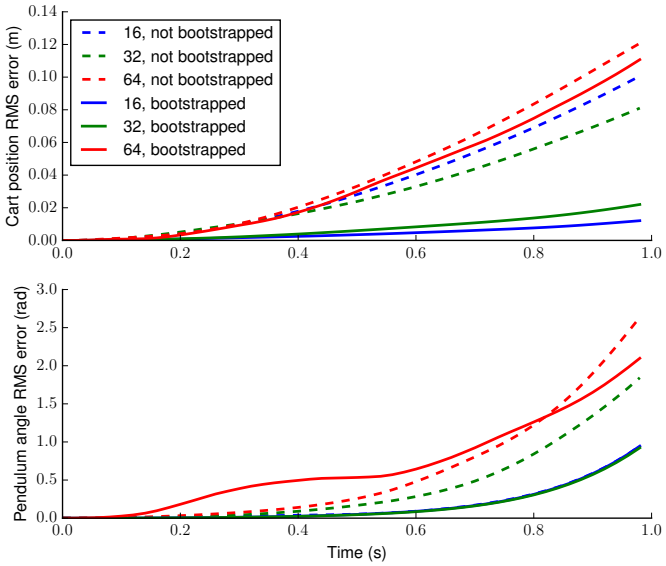


Figure 4.5: Multi-step prediction error for cart position and pendulum angle

can be achieved with perfect model knowledge, which indicates that, in this case, our MPC algorithm is robust to these errors.

4.5.2 Real-World AutoRally Task

We applied our approach to the task of aggressively driving around a dirt track with the Georgia Tech AutoRally vehicle (Goldfain et al., 2019). In prior work, Williams, Drews, et al. (2016) successfully applied MPPI to this task using a physics-inspired model. In our experiments, we used a neural network in place of this hand-designed model.

Bootstrapping Dataset

To train an initial model, we collected a system identification dataset of approximately 30 minutes of human-controlled driving at speeds varying between 4 and 10 m/s. The driving demonstrations are broken into five distinct behaviors:

1. normal driving at low speeds $(4\text{–}6\ \mathrm{m/s})$
2. zig-zag maneuvers performed at low speeds $(4\text{–}6\ \mathrm{m/s})$,

Figure 4.6: Experimental setup at the Georgia Tech Autonomous Racing Facility.

3. linear acceleration maneuvers which consist of accelerating the vehicle as much as possible in a straight line, and then braking before starting to turn,

4. sliding maneuvers where the pilot attempts to slide the vehicle as much as possible, and

5. high speed driving where the pilot simply tries to drive the vehicle around the track as fast as possible.

Each one of these maneuvers was performed for three minutes while moving around the track clockwise and for another three minutes moving counter-clockwise.

Experimental Setup

The experiments took place at the Georgia Tech Autonomous Racing Facility (Fig. 4.6). This facility consists of an elliptical dirt track approximately 3 meters wide and 30 meters across at its furthest point. We provided the MPPI controller with a global map of the track in the form of a cost-map grid. This cost-map is a smoothed occupancy grid with values of 0 corresponding to the center of the track and which increases to 1 as we move to the boundaries of the track. The cost-map values are 1 for terrain that is outside of the boundaries. The cost-map grid has a 10-centimeter resolution and is stored in CUDA texture memory for efficient look-ups inside the optimization kernel. The AutoRally system and task descriptions are given in Section A.3.

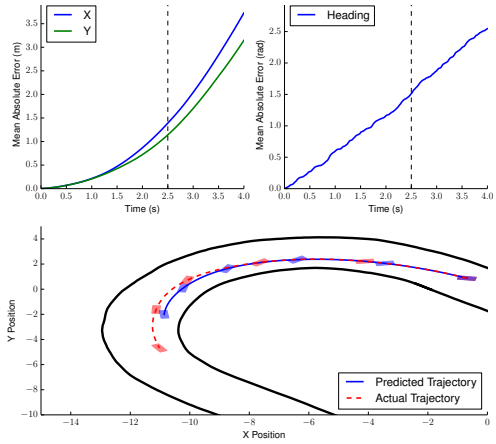


Figure 4.7: Top: Multi-step prediction error on AutoRally dynamics, the vertical bar denotes the planning horizon. Bottom: Actual trajectory vs. predicted trajectory sequence. The prediction is made off-line from an initial condition and executed control sequence that was observed while running the MPC algorithm. Orientation markers are evenly spaced in time.

We used a neural network with 2 hidden layers of 32 neurons each and hyperbolic tangent non-linearities. The MPPI controller used a time horizon of 2.5 seconds, a control frequency of 40 Hz (so that $H = 100$ time steps), and $K = 1200$ samples every time-step. This corresponds to 4.8 million forward passes through the neural network every second. We performed on-board computation using an Nvidia GTX 750 Ti GPU, which has 640 CUDA cores.

During training, we set the slip angle threshold to 15.8° (0.275 rad), and for the final testing runs we raised it to 21.5° (0.375 rad). The sampling covariance is $\Sigma = \text{diag}(0.45^2, 0.5^2)$. During training, we set the desired speed to 9 m/s (20 mph) and then gradually raised it to 13 m/s (29 mph) for the final testing run. For collecting statistics, we define a trial as 3 laps around the track starting from a full stop. Each training/test iteration consisted of three separate trial runs.

Results

With training settings of 9 m/s and 0.275 rad, the controller successfully maneuvered the vehicle around the track using the model only trained on the demonstrated system identi-

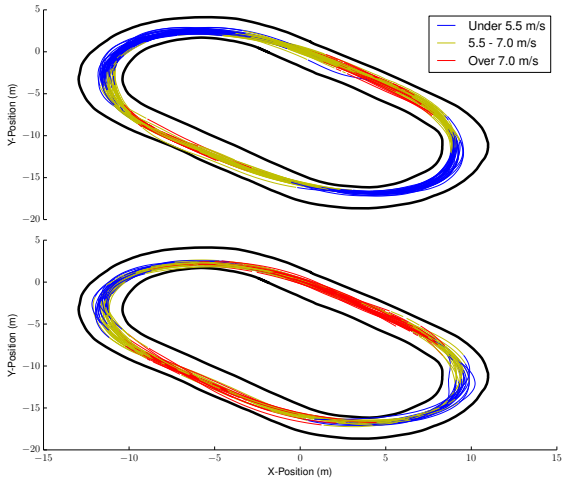


Figure 4.8: Trajectory traces and speeds during training runs (top) and more aggressive testing runs (bottom). Direction of travel is counter-clockwise.

fication data. We performed 5 iterations, each consisting of 3 trials, for a total of 45 laps around the track. This corresponds to slightly over 8 minutes of total run-time.

Adding new data into the training set and re-training the neural network model did not noticeably improve the performance of MPPI. One explanation for this is that the initial dataset was deliberately collected for system identification and consisted of a variety of maneuvers meant to excite various modes of the dynamics. This is in contrast to the simulated experiments where the initial dataset consisted of a *single* task repeatedly performed by an expert controller.

After the training runs, we tested the limits of the MPPI controller, using the model from the final training iteration. Specifically, we increased the threshold for penalized slip angle and the desired speed. We started with the desired speed at $10\ \mathrm{m/s}$ and gradually increase it to $13\ \mathrm{m/s}$. At $13\ \mathrm{m/s}$, the vehicle only completed two of three runs. At the lower settings, it successfully completed all trials.

Fig. 4.8 shows the paths taken by the controller along with their velocity profiles. In both cases, the vehicle slowed down coming into turns and accelerates out of them, eventually reaching a high speed along the middle of the straight. This is an intuitively obvious strategy; however, it is worth noting that there is no portion of the cost function which ex-

Table 4.2: Training statistics

Iteration	Avg. Lap (s)	Best Lap (s)	Top Speed (m/s)	Max Slip Angle
1	10.98	10.50	8.13	22.1°
2	10.79	10.32	7.84	27.4°
3	11.05	10.55	8.00	33.5°
4	10.85	10.43	7.60	25.8°
5	11.11	10.84	7.49	22.6°

Table 4.3: Testing statistics

Target Speed (m/s)	Avg. Lap (s)	Best Lap (s)	Top Speed (m/s)	Max Slip Angle
10	10.34	9.93	8.05	38.7°
11	9.97	9.43	8.71	34.7°
12	9.88	9.47	8.63	43.7°
13	9.74	9.36	8.44	48.7°

plicitly forces this behavior. Rather, this behavior emergered from the interaction between the neural network dynamics and cost function.

Tables 4.2 and 4.3 show the statistics for the training and testing runs. The more aggressive runs completed the trials in faster times and obtained a higher top speed and maximum slip angle than the training runs. The velocity profiles of the training and test trials also differed dramatically. In the conservative training runs, the vehicle hit its top speed in the first half of the straight and immediately slowed down, eventually coasting into the corner. In the aggressive setting, the vehicle maintained its speed all the way into the corner and then power-slid through the turn (Fig. 4.1). This is demonstrated by the high slip angles achieved by the more aggressive runs. The overall lap times in our experiments are slightly faster than what Williams, Drews, et al. (2016) achieved previously.

4.6 Conclusion

We derived an information-theoretic version of model predictive path integral control which generalizes previous interpretations by allowing for general dynamics. We exploited this generalization by applying the MPPI algorithm in the context of model-based reinforcement learning and used multi-layer neural networks to learn a dynamics model. In two

challenging simulation tasks, the controller with the learned neural network model achieved performance within 10% of what is obtained with the ground-truth model.

We demonstrated the scalability and practicality of the algorithm on real hardware in an aggressive driving scenario, where the algorithm successfully raced a one-fifth scale rally car around a 30 meter long track at speeds over $8\ \mathrm{m/s}$. In doing this, the controller performed difficult controlled power-slides around corners. This success came despite the presence of significant disturbances, such as deep grooves on portions of the track and patches of very fine loose dirt which could have easily caused the vehicle to lose traction.

This type of model-based reinforcement learning that we propose, combining generalized model predictive control with machine learning approaches for learning dynamics, is a promising new direction for solving the challenging problems that arise in robotics. The key tools in this approach are the information-theoretic concept of KL divergence, scalable machine learning algorithms for learning dynamics, and intensive parallel computation for online optimization. The result of our approach is the emergence of complex behaviors, such as power-sliding through turns when racing, that arise purely due to the interaction between the optimization, cost function, and learned dynamics.

CHAPTER 5
AN ONLINE LEARNING APPROACH TO MODEL PREDICTIVE CONTROL

Model predictive control (MPC) is a powerful technique for solving dynamic control tasks. In this chapter, we show that there exists a close connection between MPC and online learning, an abstract theoretical framework for analyzing online decision making in the optimization literature. This new perspective provides a foundation for leveraging powerful online learning algorithms to design MPC algorithms. Specifically, we propose a new algorithm based on dynamic mirror descent (DMD), an online learning algorithm that is designed for non-stationary setups. Our algorithm, Dynamic Mirror Descent Model Predictive Control (DMD-MPC), represents a general family of MPC algorithms that includes many existing techniques as special instances. DMD-MPC also provides a fresh perspective on previous heuristics used in MPC and suggests a principled way to design new MPC algorithms. In the experimental section of this chapter, we demonstrate the flexibility of DMD-MPC, presenting a set of new MPC algorithms on a simple simulated cartpole and a simulated and real-world aggressive driving task. Videos of the real-world experiments are available at `https://youtu.be/vZST3v0_S9w` and `https://youtu.be/MhuqiHo2t98`.

5.1 Introduction

Model predictive control (MPC) (Mayne, 2014) is an effective tool for control tasks involving dynamic environments, such as helicopter aerobatics (Abbeel et al., 2010) and aggressive driving (Williams, Drews, et al., 2016). One reason for its success is the pragmatic principle it adopts in choosing controls: rather than wasting computational power to optimize a complicated controller for the full-scale problem (which may be difficult to accurately model), MPC instead optimizes a simple controller (e.g., an open-loop control sequence) over a shorter planning horizon that is just sufficient to make a sensible decision

at the current moment. By alternating between optimizing the simple controller and applying its corresponding control on the real system, MPC results in a closed-loop policy that can handle modeling errors and dynamic changes in the environment.

Various MPC algorithms have been proposed, using tools ranging from constrained optimization techniques (Camacho and Alba, 2013; Mayne, 2014; Tassa et al., 2014) to sampling-based techniques (Williams, Drews, et al., 2018). In this chapter, we show that, while these algorithms were originally designed differently, if we view them through the lens of *online learning* (Hazan, 2016), many of them actually follow the *same* general update rule. Online learning is an abstract theoretical framework for analyzing online decision making. Formally, it concerns iterative interactions between a learner and an environment over T rounds. At round t, the learner makes a decision $\tilde{\boldsymbol{\theta}}_t$ from some decision set $\boldsymbol{\Theta}$. The environment then chooses a loss function ℓ_t based on the learner's decision, and the learner suffers a cost $\ell_t(\tilde{\boldsymbol{\theta}}_t)$. In addition to seeing the decision's cost, the learner may be given additional information about the loss function (e.g., its gradient evaluated at $\tilde{\boldsymbol{\theta}}_t$) to aid in choosing the next decision $\tilde{\boldsymbol{\theta}}_{t+1}$. The learner's goal is to minimize the accumulated costs $\sum_{t=1}^{T} \ell_t(\tilde{\boldsymbol{\theta}}_t)$, e.g., by minimizing regret (Hazan, 2016).

We find that the MPC process bears a strong similarity with online learning. At time t (i.e., round t), an MPC algorithm optimizes a controller (i.e., the decision) over some cost function (i.e., the per-round loss). To do so, it observes the cost of the initial controller (i.e., $\ell_t(\tilde{\boldsymbol{\theta}}_t)$), improves the controller, and executes a control based on the improved controller in the environment to get to the next state (which in turn defines the next per-round loss) with a new controller $\tilde{\boldsymbol{\theta}}_{t+1}$.

In view of this connection, we propose a generic framework, *DMD-MPC* (Dynamic Mirror Descent Model Predictive Control), for synthesizing MPC algorithms. DMD-MPC is based on a first-order online learning algorithm called dynamic mirror descent (DMD) (Hall and Willett, 2013), a generalization of mirror descent (Beck and Teboulle, 2003) for dynamic settings. We show that several existing MPC algorithms (Williams, Wagener, et al.,

2017; Williams, Drews, et al., 2018) are special cases of DMD-MPC, given specific choices of step sizes, loss functions, and regularization. Furthermore, we demonstrate how new MPC algorithms can be derived systematically from DMD-MPC with only mild assumptions on the regularity of the cost function. This allows us to even work with discontinuous cost functions (like indicators) and discrete controls. Thus, DMD-MPC offers a spectrum from which practitioners can easily customize new algorithms for their applications.

In the experiments, we apply DMD-MPC to design a range of MPC algorithms and study their empirical performance. Our results indicate the extra design flexibility offered by DMD-MPC does make a difference in practice; by properly selecting hyperparameters which are obscured in the previous approaches, we are able to improve the performance of existing algorithms. Finally, we apply DMD-MPC on a real-world AutoRally car platform (Goldfain et al., 2019) for autonomous driving tasks and show it can yield competent performance, including achieving aggressive yet stable driving at high speeds even under noisy control updates by properly setting the DMD-MPC hyperparameters.

5.2 An Online Learning Perspective on MPC

5.2.1 The MPC Problem Setup

In this chapter, we adopt the MPC approach to choosing u_t: at state x_t, we imagine controlling a stochastic dynamics model $\hat{p}(x'|x,u)$ (which approximates our actual system $p(x'|x,u)$) for H time steps into the future. Our planned controls come from a control distribution $\hat{\pi}_{\boldsymbol{\theta}}$ that is parameterized by some vector $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, where $\boldsymbol{\Theta}$ is the feasible parameter set. In each simulation (i.e., rollout), we sample a control sequence $\hat{\boldsymbol{u}}_t$ from the control distribution $\hat{\pi}_{\boldsymbol{\theta}}$ and then sample from the state sequence from the rollout distribution $\hat{p}(\hat{\boldsymbol{x}}_t \mid \hat{x} = x_t, \hat{\boldsymbol{u}}_t)$ (Eq. (3.2)). Through these simulations, we desire to select a parameter $\boldsymbol{\theta}_t \in \boldsymbol{\Theta}$ that minimizes an MPC objective $\hat{J}(\boldsymbol{\theta}; x_t)$, which aims to predict the performance

of the system if we were to apply the control distribution $\hat{\pi}_{\boldsymbol{\theta}}$ starting from x_t.[1] In other words, we wish to find the parameter setting that solves

$$\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \quad \hat{J}(\boldsymbol{\theta}; x_t). \tag{5.1}$$

Once $\boldsymbol{\theta}_t$ is decided, we then sample[2] $\hat{\boldsymbol{u}}_t$ from $\hat{\pi}_{\boldsymbol{\theta}}$, extract the first control $\hat{u}_t$, and apply it on the real dynamical system $p(x_{t+1}|x_t, u_t)$ (i.e., set $u_t = \hat{u}_t$) to go to the next state x_{t+1}. Because $\boldsymbol{\theta}_t$ is determined based on x_t, MPC is effectively state-feedback.

The motivation behind MPC is to use the MPC objective $\hat{J}$ to reason about the controls required to achieve desirable long-term behaviors. A popular MPC objective is:

$$\hat{J}(\boldsymbol{\theta}; x_t) \triangleq \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}[\hat{C}(x_t, \hat{\boldsymbol{u}}_t)] = \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)} \mathbb{E}_{\hat{p}(\hat{\boldsymbol{x}}_t|\hat{x}_t = x_t, \hat{\boldsymbol{u}}_t)} \left[\sum_{h=0}^{H-1} c(\hat{x}_{t+h}, \hat{u}_{t+h}) + c_{\text{term}}(\hat{x}_{t+H}) \right],$$

which estimates the future costs over H steps. Later, in Section 5.3.1, we will discuss several MPC objectives and their properties.

Although the idea of MPC sounds intuitively promising, the optimization can only be approximated in practice (e.g., using an iterative algorithm like gradient descent), because Eq. (5.1) is often a stochastic program (like the example above) and the control command u_t needs to be computed at a high frequency. In consideration of this imperfection, it is common to heuristically *bootstrap* the previous approximate solution as the initialization to the current problem. Specifically, let $\boldsymbol{\theta}_{t-1}$ be the approximate solution to the previous problem and $\tilde{\boldsymbol{\theta}}_t$ denote the initial condition of $\boldsymbol{\theta}$ in solving Eq. (5.1). The bootstrapping step can then written as

$$\tilde{\boldsymbol{\theta}}_t = \Phi(\boldsymbol{\theta}_{t-1}) \tag{5.2}$$

[1] $\hat{J}$ can be seen as a surrogate for the long-term performance of our controller. Typically, we set the planning horizon H to be much smaller than T to reduce the optimization difficulty and to mitigate modeling errors.

[2] This setup can also optimize deterministic policies, e.g., by defining $\hat{\pi}_{\boldsymbol{\theta}}$ to be a Gaussian policy with the mean being the deterministic policy.

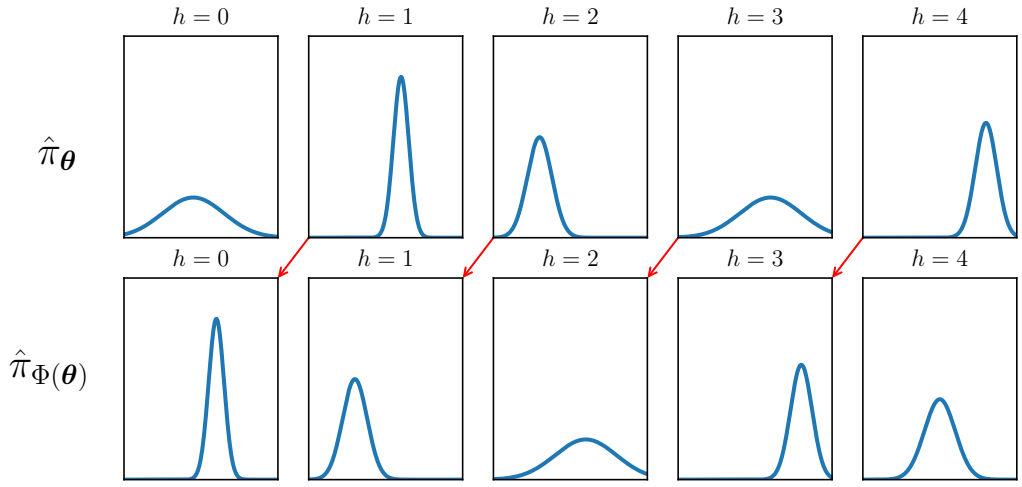


Figure 5.1: A simple example of the shift operator Φ. Here, the control distribution $\hat{\pi}_{\boldsymbol{\theta}}$ consists of a sequence of $H = 5$ independent Gaussian distributions. The shift operator moves the parameters of the Gaussians one time step forward and replaces the parameters at $h = 4$ with some default parameters.

by defining a *shift operator* Φ that outputs a new parameter in $\boldsymbol{\Theta}$. This Φ can be chosen to satisfy desired properties, e.g., $\hat{\pi}_{\Phi(\boldsymbol{\theta}_{t-1})}(\hat{u}_{t,h}|\hat{u}_{t-1}) = \hat{\pi}_{\boldsymbol{\theta}_{t-1}}(\hat{u}_{t-1,h+1}|\hat{u}_{t-1})$ for $h = 0, \dots, H-2$. A simple example of this property is shown in Fig. 5.1. Note that $\hat{\boldsymbol{u}}_t$ also involves a new control $\hat{u}_{t+H-1}$ that is not in $\hat{\boldsymbol{u}}_{t-1}$, so the choice of Φ is not unique but algorithm dependent. Because the subproblems in Eq. (5.1) of two consecutive time steps share all control variables except for the first and the last ones, the "shifted" parameter $\Phi(\boldsymbol{\theta}_{t-1})$ to the current problem should be almost as good as the optimized parameter $\boldsymbol{\theta}_{t-1}$ is to the previous problem. In other words, setting $\tilde{\boldsymbol{\theta}}_t = \Phi(\boldsymbol{\theta}_{t-1})$ provides a warm start to Eq. (5.1) and amortizes the computational complexity of solving for $\boldsymbol{\theta}_t$.

5.2.2 The Online Learning Perspective

As discussed, the iterative update process of MPC resembles the setup of online learning (Hazan, 2016). Here, we provide the details to convert an MPC setup into an online learning problem. Recall from the introduction that online learning mainly consists of three

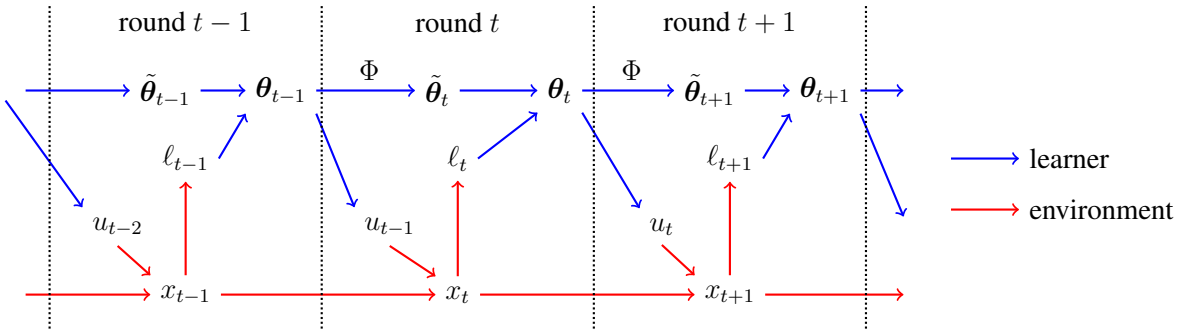


Figure 5.2: Diagram of the online learning perspective.

components: the decision set, the learner's strategy for updating decisions, and the environment's strategy for updating per-round losses. We show that MPC has counterparts that correspond to each component.

We use the concept of per-round loss in online learning as a mechanism to measure the decision uncertainty in MPC, and propose the following identification (shown in Fig. 5.2) for the MPC setup described in the previous section: we set the rounds in online learning to synchronize with the time steps of our control system, set the decision set $\boldsymbol{\Theta}$ as the space of feasible parameters of the control distribution $\hat{\pi}_{\boldsymbol{\theta}}$, set the learner as the MPC algorithm which in round t outputs the decision $\tilde{\boldsymbol{\theta}}_t \in \boldsymbol{\Theta}$ and side information u_{t-1}, and set the per-round loss as

$$\ell_t(\cdot) \triangleq \hat{J}(\cdot\,; x_t). \tag{5.3}$$

In other words, in round t of this online learning setup, the learner plays a decision $\tilde{\boldsymbol{\theta}}_t$ along with a side information u_{t-1} (based on the optimized solution $\boldsymbol{\theta}_{t-1}$ and the shift operator in Eq. (5.2)), the environment selects the per-round loss $\ell_t(\cdot) = \hat{J}(\cdot; x_t)$ (by applying u_{t-1} to the real dynamical system to transit the state to x_t), and finally the learner receives ℓ_t and incurs cost $\ell_t(\tilde{\boldsymbol{\theta}}_t)$ (which measures the sub-optimality of the future plan made by the MPC algorithm).

This online learning setup differs slightly from the standard setup in its separation of the

decision $\tilde{\boldsymbol{\theta}}_t$ and the side information u_{t-1}; while our setup can be converted into a standard one that treats $\boldsymbol{\theta}_{t-1}$ as the sole decision played in round t, we adopt this explicit separation in order to emphasize that the variable part of the incurred cost $\ell_t(\tilde{\boldsymbol{\theta}}_t)$ pertains to only $\tilde{\boldsymbol{\theta}}_t$. That is, the learner cannot go back and revert the previous control u_{t-1} already applied on the system, but only uses ℓ_t to update the current and future controls $\hat{u}_t, \ldots, \hat{u}_{t+H-1}$.

The performance of the learner in online learning (which by our identification is the MPC algorithm) is measured in terms of the accumulated costs $\sum_{t=1}^{T} \ell_t(\tilde{\boldsymbol{\theta}}_t)$. For problems in non-stationary setups, a normalized way to describe the accumulated costs in the online learning literature is through the concept of *dynamic regret* (Hall and Willett, 2013; L. Zhang et al., 2018), which is defined as

$$\sum_{t=1}^{T} (\ell_t(\tilde{\boldsymbol{\theta}}_t) - \ell_t(\boldsymbol{\theta}_t^\star)), \tag{5.4}$$

where $\boldsymbol{\theta}_t^\star \in \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \ell_t(\boldsymbol{\theta})$. Dynamic regret quantifies how suboptimal the played decisions $\tilde{\boldsymbol{\theta}}_1, \ldots, \tilde{\boldsymbol{\theta}}_T$ are on the corresponding loss functions. In our proposed problem setup, the optimality concept associated with dynamic regret conveys a *consistency* criterion desirable for MPC: we would like to make a decision $\boldsymbol{\theta}_{t-1}$ at state x_{t-1} such that, after applying control u_{t-1} and entering the new state x_t, its shifted plan $\tilde{\boldsymbol{\theta}}_t$ remains close to optimal with respect to the new loss function ℓ_t. If the dynamics model $\hat{p}$ is accurate and the MPC algorithm is ideally solving Eq. (5.1), we can expect that bootstrapping the previous solution $\boldsymbol{\theta}_{t-1}$ through Eq. (5.2) into $\tilde{\boldsymbol{\theta}}_t$ would result in a small instantaneous gap $\ell_t(\tilde{\boldsymbol{\theta}}_t) - \ell_t(\boldsymbol{\theta}_t^\star)$ which is solely due to unpredictable future information (such as the stochasticity in the dynamical system). In other words, an online learning algorithm with small dynamic regret, if applied to our online learning setup, would produce a consistently optimal MPC algorithm with regard to the solution concept discussed above. However, we note that having small dynamic regret here does not directly imply good absolute performance on the control system, because the overall performance of the MPC algorithm is largely dependent on the

form of the MPC objective $\hat{J}$ (e.g., through choice of H and accuracy of $\hat{p}$). Small dynamic regret more precisely means whether the plan produced by an MPC algorithm is consistent with the given MPC objective.

5.3 A Family of MPC Algorithms Based on Dynamic Mirror Descent

The online learning perspective on MPC suggests that good MPC algorithms can be designed from online learning algorithms that achieve small dynamic regret. This is indeed the case. We will show that a range of existing MPC algorithms are in essence applications of a classical online learning algorithm called dynamic mirror descent (DMD) (Hall and Willett, 2013). DMD is an extension of mirror descent (Beck and Teboulle, 2003) to problems involving dynamic comparators (in this case, the $(\boldsymbol{\theta}_t^\star)_t$ in dynamic regret in Eq. (5.4)). In round t, DMD applies the following update rule:

$$\begin{aligned} \boldsymbol{\theta}_t &= \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \ \alpha \boldsymbol{g}_t^\mathsf{T}\boldsymbol{\theta} + D_\psi(\boldsymbol{\theta} \,\|\, \tilde{\boldsymbol{\theta}}_t) \\ \tilde{\boldsymbol{\theta}}_{t+1} &= \Phi(\boldsymbol{\theta}_t), \end{aligned} \tag{5.5}$$

where $\boldsymbol{g}_t = \nabla \ell_t(\tilde{\boldsymbol{\theta}}_t)$ (which can be approximated by sampling if $\nabla \ell_t(\tilde{\boldsymbol{\theta}}_t)$ is an expectation), Φ is called the shift model, $\alpha > 0$ is the step size, and, for some $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \boldsymbol{\Theta}$, $D_\psi(\boldsymbol{\theta} \,\|\, \boldsymbol{\theta}') \triangleq \psi(\boldsymbol{\theta}) - \psi(\boldsymbol{\theta}') - \nabla\psi(\boldsymbol{\theta}')^\mathsf{T}(\boldsymbol{\theta} - \boldsymbol{\theta}')$ is the Bregman divergence generated by a strictly convex function ψ on $\boldsymbol{\Theta}$.

The first step of DMD in Eq. (5.5) is reminiscent of the proximal update in the usual mirror descent algorithm. It can be thought of as an optimization step where the Bregman divergence acts as a regularization to keep $\boldsymbol{\theta}$ close to $\tilde{\boldsymbol{\theta}}_t$. Although $D_\psi(\boldsymbol{\theta} \,\|\, \boldsymbol{\theta}')$ is not necessarily a metric (since it may not be symmetric), it is still useful to view it as a distance between $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$. Indeed, familiar examples of the Bregman divergence include the squared Euclidean distance and KL divergence (Eq. (2.4)) (Banerjee et al., 2005).

The second step of DMD in Eq. (5.5) uses the shift model Φ to anticipate the optimal

decision for the next round. In the context of MPC, a natural choice for the shift model is the shift operator in Eq. (5.2) defined previously in Section 5.2.1 (hence the same notation), because the per-round losses in two consecutive rounds here concern problems with shifted time indices. Hall and Willett (2013) show that the dynamic regret of DMD scales with how much the optimal decision sequence $(\boldsymbol{\theta}_t^\star)_t$ deviates from Φ (i.e., $\sum_t \|\boldsymbol{\theta}_{t+1}^\star - \Phi(\boldsymbol{\theta}_t^\star)\|$), which arises from the unpredictable elements of the problem.

Algorithm 5: Dynamic Mirror Descent Model Predictive Control (DMD-MPC)

for $t = 1, 2, \ldots, T$ **do**
  Set $\ell_t(\cdot) = \hat{J}(\cdot\,; x_t)$.
  Set $\boldsymbol{g}_t = \nabla \ell_t(\tilde{\boldsymbol{\theta}}_t)$ (or some estimate of $\nabla \ell_t(\tilde{\boldsymbol{\theta}}_t)$).
  Set $\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \ \alpha \boldsymbol{g}_t^\mathsf{T} \boldsymbol{\theta} + D_\psi(\boldsymbol{\theta} \,\|\, \tilde{\boldsymbol{\theta}}_t)$.
  Sample $\hat{\boldsymbol{u}}_t \sim \hat{\pi}_{\boldsymbol{\theta}_t}(\hat{\boldsymbol{u}}_t)$ and set $u_t = \hat{u}_t$.
  Sample $x_{t+1} \sim p(x_{t+1}|x_t, u_t)$.
  Set $\tilde{\boldsymbol{\theta}}_{t+1} = \Phi(\boldsymbol{\theta}_t)$.
end

Applying DMD in Eq. (5.5) to the online learning problem described in Section 5.2.2 leads to an MPC algorithm shown in Algorithm 5, which we call *DMD-MPC*. More precisely, DMD-MPC represents a family of MPC algorithms in which a specific instance is defined by a choice of:

1. the MPC objective $\hat{J}$ in Eq. (5.3),
2. the form of the control distribution $\hat{\pi}_{\boldsymbol{\theta}}$, and
3. the Bregman divergence D_ψ in Eq. (5.5).

Thus, we can use DMD-MPC as a generic strategy for synthesizing MPC algorithms. In the following, we use this recipe to recreate several existing MPC algorithms and demonstrate new MPC algorithms that naturally arise from this framework.

Table 5.1: Loss functions considered for DMD-MPC

Loss function	$\ell_t(\boldsymbol{\theta})$	Gradient weight $w_t(\hat{\boldsymbol{u}}_t)$
Expected cost	$\mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}[\hat{C}_t(\hat{\boldsymbol{u}}_t)]$	$-\hat{C}_t(\hat{\boldsymbol{u}}_t)$
Expected utility	$-\log \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}[U_t(\hat{C}_t(\hat{\boldsymbol{u}}_t))]$	$\frac{U_t(\hat{C}_t(\hat{\boldsymbol{u}}_t))}{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}})}[U_t(\hat{C}_t(\hat{\boldsymbol{u}}))]}$
Expected threshold utility	$-\log \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}[\mathbf{1}\{\hat{C}_t(\hat{\boldsymbol{u}}_t) \leq C_{t,\max}\}]$	$\frac{\mathbf{1}\{\hat{C}_t(\hat{\boldsymbol{u}}_t) \leq C_{t,\max}\}}{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}})}[\mathbf{1}\{\hat{C}_t(\hat{\boldsymbol{u}}) \leq C_{t,\max}\}]}$
Expected exponential utility	$-\log \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}]$	$\frac{e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}}{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}})}[e^{-\hat{C}_t(\hat{\boldsymbol{u}})/\lambda}]}$

5.3.1 Loss Functions

We discuss several definitions of the per-round loss ℓ_t, which all result from the formulation in Eq. (5.3) but with different $\hat{J}$. These loss functions are based on the statistic $\hat{C}(x_t, \hat{\boldsymbol{u}}_t)$ (Eq. (3.3)) which measures the average H-step accumulated costs of a given control trajectory $\hat{\boldsymbol{u}}_t$ starting from x_t. For transparency of exposition, we will suppose henceforth that the control distribution $\hat{\pi}_{\boldsymbol{\theta}}$ is open-loop[3]; similar derivations follow naturally for closed-loop control distributions. For convenience of practitioners, we also provide expressions of their gradients in terms of the likelihood-ratio derivative[4] (Glynn, 1990). It turns out that all these gradients will have the form

$$\nabla \ell_t(\tilde{\boldsymbol{\theta}}_t) = -\,\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[w_t(\hat{\boldsymbol{u}}_t)\nabla_{\boldsymbol{\theta}} \log \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)] \tag{5.6}$$

for some function w_t (Table 5.1). We approximate these gradients by sampling K sequences $\hat{\boldsymbol{u}}_t^1, \ldots, \hat{\boldsymbol{u}}_t^K$ from $\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}$ and evaluating:

$$\nabla \ell_t(\tilde{\boldsymbol{\theta}}_t) \approx \frac{1}{K}\sum_{k=1}^{K} w_t(\hat{\boldsymbol{u}}_t^k)\nabla_{\boldsymbol{\theta}} \log \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t) \qquad (\hat{\boldsymbol{u}}_t^1, \ldots, \hat{\boldsymbol{u}}_t^K \overset{\text{iid}}{\sim} \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}). \tag{5.7}$$

[3]Note again that even while using open-loop control distributions, the overall control law of MPC is state-feedback.

[4]We assume the control distribution is sufficiently regular with respect to its parameter so that the likelihood-ratio derivative rule holds.

In practice, because w_t involves expectations such as that in $\hat{C}_t$ (Eq. (3.3)), we must also separately estimate w_t via Monte Carlo.

Expected Cost

The most commonly used MPC objective is the H-step expected accumulated cost function under model dynamics, because it directly estimates the expected long-term behavior when the dynamics model $\hat{p}$ is accurate and H is large enough. Its per-round loss function and gradient are:[5]

$$\ell_t(\boldsymbol{\theta}) = \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}[\hat{C}_t(\hat{\boldsymbol{u}}_t)] \tag{5.8}$$

$$\nabla \ell_t(\tilde{\boldsymbol{\theta}}_t) = \mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[\hat{C}_t(\hat{\boldsymbol{u}}_t) \nabla_{\boldsymbol{\theta}} \log \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)]. \tag{5.9}$$

Expected Utility

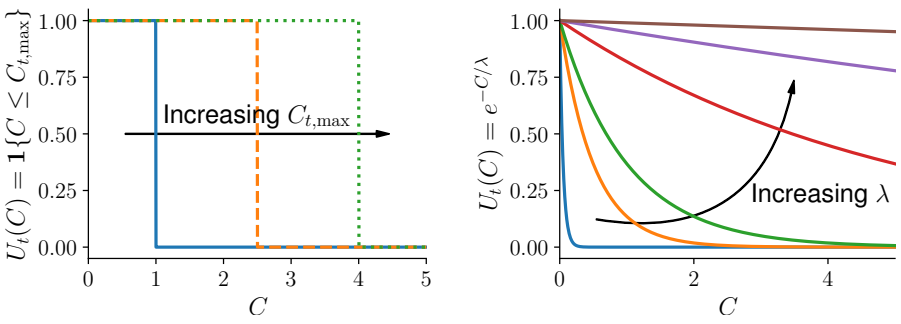
Instead of optimizing for average cost, we may care to optimize for some preference related to the average trajectory cost $\hat{C}_t$, such as having the cost be below some threshold. This idea can be formulated as a *utility* that returns a normalized score related to the preference for a given average trajectory cost $\hat{C}_t(\hat{\boldsymbol{u}}_t)$. Specifically, suppose that $\hat{C}_t$ is lower bounded by zero[6] and at some round t define the utility $U_t : \mathbb{R}_+ \to [0, 1]$ to be a function with the following properties:

- $U_t(0) = 1$,
- U_t is monotonically decreasing, and
- $\lim_{C \to \infty} U_t(C) = 0$.

These are sensible properties since we attain maximum utility when we have zero cost, the utility never increases with the cost, and the utility approaches zero as the cost increases

[5] In experiments, we subtract the empirical average of the sampled costs from $\hat{C}_t$ in Eq. (5.9) to reduce the variance, at the cost of a small amount of bias.

[6] If this is not the case, let $\hat{C}_{\min} \triangleq \inf_{\hat{\boldsymbol{u}}_t} \hat{C}_t(\hat{\boldsymbol{u}}_t)$, which we assume is finite. We can then replace $\hat{C}_t(\hat{\boldsymbol{u}}_t)$ with $\hat{C}_t(\hat{\boldsymbol{u}}_t) - \hat{C}_{\min}$.



(a) Threshold utility (b) Exponential utility

Figure 5.3: Visualization of different utilities.

without bound. We then define the per-round loss as

$$\ell_t(\boldsymbol{\theta}) = -\log \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}[U_t(\hat{C}_t(\hat{\boldsymbol{u}}_t))] \tag{5.10}$$

$$\nabla \ell_t(\tilde{\boldsymbol{\theta}}_t) = -\frac{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[U_t(\hat{C}_t(\hat{\boldsymbol{u}}_t))\, \nabla_{\boldsymbol{\theta}} \log \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)]}{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[U_t(\hat{C}_t(\hat{\boldsymbol{u}}_t))]}. \tag{5.11}$$

The gradient in Eq. (5.11) is particularly appealing when estimated with samples. Suppose we sample K control sequences $\hat{\boldsymbol{u}}_t^1, \ldots, \hat{\boldsymbol{u}}_t^K$ from $\hat{\pi}_{\boldsymbol{\theta}}$. Then, the estimate of Eq. (5.11) is a convex combination of gradients:

$$\nabla \ell_t(\tilde{\boldsymbol{\theta}}_t) \approx -\sum_{k=1}^{K} w_k \nabla_{\boldsymbol{\theta}} \log \hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t^k) \qquad (\hat{\boldsymbol{u}}_t^1, \ldots, \hat{\boldsymbol{u}}_t^K \overset{\text{iid}}{\sim} \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}),$$

where $w_k = U_t(C_k) \Big/ \sum_{\ell=1}^{K} U_t(C_\ell)$ and $C_k = \hat{C}_t(\hat{\boldsymbol{u}}_t^k)$ for $k = 1, \ldots, K$. We see that each weight w_k is computed by considering the relative utility of its corresponding trajectory. A cost C_k with high relative utility will push its corresponding weight w_k closer to one, whereas a low relative utility will cause w_k to be close to zero, effectively rejecting the corresponding sample.

We give two examples of utilities and their related losses.

Threshold Utility For example, we may care about the system being below some cost threshold as often as possible. To encode this preference, we can use the threshold utility

$U_t(C) \triangleq \mathbf{1}\{C \leq C_{t,\max}\}$, where $\mathbf{1}\{\cdot\}$ is the indicator function and $C_{t,\max}$ is a threshold parameter. Under this choice, the loss and its gradient become

$$\begin{aligned} \ell_t(\boldsymbol{\theta}) &= -\log \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}[\mathbf{1}\{\hat{C}_t(\hat{\boldsymbol{u}}_t) \leq C_{t,\max}\}] \qquad (5.12) \\ &= -\log \mathbb{P}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}(\hat{C}_t(\hat{\boldsymbol{u}}_t) \leq C_{t,\max}) \\ \nabla \ell_t(\tilde{\boldsymbol{\theta}}_t) &= -\frac{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[\mathbf{1}\{\hat{C}_t(\hat{\boldsymbol{u}}_t) \leq C_{t,\max}\} \nabla_{\boldsymbol{\theta}} \log \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)]}{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[\mathbf{1}\{\hat{C}_t(\hat{\boldsymbol{u}}_t) \leq C_{t,\max}\}]}. \qquad (5.13) \end{aligned}$$

As we can see, this loss function also gives the probability of achieving cost below some threshold. As a result (Fig. 5.3a), costs below $C_{t,\max}$ are treated the same in terms of the utility. This can potentially make optimization easier since we are trying to make good trajectories as likely as possible instead of finding the best trajectories as in Eq. (5.8).

However, if the threshold $C_{t,\max}$ is set too low and the gradient is estimated with samples, the gradient estimate may have high variance due to the large number of rejected samples. Because of this, in practice, the threshold is set adaptively, e.g., as the largest cost of the top *elite fraction* of the sampled trajectories with smallest costs (Botev et al., 2013). This allows the controller to make the best sampled trajectories more likely and therefore improve the controller.

Exponential Utility We can also opt for a continuous surrogate of the indicator function, in this case the exponential utility $U_t(C) \triangleq e^{-C/\lambda}$, where $\lambda > 0$ is a scaling parameter. Unlike the indicator function, the exponential utility provides nonzero feedback for any given cost and allows us to discriminate between costs (i.e., if $C_1 > C_2$, then $U_t(C_1) < U_t(C_2)$), as shown in Fig. 5.3b. Furthermore, λ acts as a continuous alternative to $C_{t,\max}$ and dictates how quickly or slowly U_t decays to zero, which in a soft way determines the cutoff point for rejecting given costs.

Under this choice, the loss and its gradient become

$$\ell_t(\boldsymbol{\theta}) = -\log \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}] \tag{5.14}$$

$$\nabla \ell_t(\tilde{\boldsymbol{\theta}}_t) = -\frac{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda} \nabla_{\boldsymbol{\theta}} \log \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)]}{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}]} \tag{5.15}$$

The loss function in Eq. (5.14) is also known as the risk-seeking objective in optimal control (Broek et al., 2010); this classical interpretation is based on a Taylor expansion of Eq. (5.14) showing

$$\ell_t(\boldsymbol{\theta}) \approx \frac{1}{\lambda} \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}[C_t(\hat{\boldsymbol{u}}_t)] - \frac{1}{\lambda^2} \operatorname{var}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}(C_t(\hat{\boldsymbol{u}}_t))$$

when λ is large, where $\operatorname{var}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)}(C_t(\hat{\boldsymbol{u}}_t))$ is the variance of $C_t(\hat{\boldsymbol{u}}_t)$ under $\hat{\pi}_{\boldsymbol{\theta}}$. Here we derive Eq. (5.14) from a different perspective that treats it as a continuous approximation of Eq. (5.12). The use of exponential transformations to approximate indicators is a common machine learning trick, e.g., the Chernoff bound (Chernoff, 1952).

5.3.2 Algorithms

We instantiate DMD-MPC with different choices of loss function, control distribution, and Bregman divergence as concrete examples to showcase the flexibility of our framework. In particular, we are able to recover well-known MPC algorithms as special cases of Algorithm 5.

Quadratic Divergence

We start with perhaps the most common Bregman divergence: the quadratic divergence. Choosing $\psi(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^\mathsf{T}\boldsymbol{A}\boldsymbol{\theta}$ for some positive definite matrix $\boldsymbol{A}$, the resulting Bregman divergence has the quadratic form $D_\psi(\boldsymbol{\theta} \,\|\, \boldsymbol{\theta}') \triangleq \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}')^\mathsf{T}\boldsymbol{A}(\boldsymbol{\theta} - \boldsymbol{\theta}')$. Then, the update

rule becomes:

$$\begin{aligned}\boldsymbol{\theta}_t &= \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \alpha \boldsymbol{g}_t^\mathsf{T}\boldsymbol{\theta} + \frac{1}{2}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}}_t)^\mathsf{T}\boldsymbol{A}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}}_t) \\ &= \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} (\boldsymbol{\theta}-\boldsymbol{\theta}_t^\star)^\mathsf{T}\boldsymbol{A}(\boldsymbol{\theta}-\boldsymbol{\theta}_t^\star)\end{aligned}$$

where $\boldsymbol{\theta}_t^\star \triangleq \tilde{\boldsymbol{\theta}}_t - \alpha \boldsymbol{A}^{-1}\boldsymbol{g}_t$. Below we discuss different choices of $\boldsymbol{A}$ and their corresponding update rules.

Projected Gradient Descent This basic update rule is a special case when $\boldsymbol{A}$ is the identity matrix. Equivalently, the update can be written as $\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \|\boldsymbol{\theta} - (\tilde{\boldsymbol{\theta}}_t - \alpha\boldsymbol{g}_t)\|^2$.

Natural Gradient Descent We can recover the natural gradient descent algorithm (Akimoto et al., 2012) by defining $\boldsymbol{A} = \mathcal{F}(\tilde{\boldsymbol{\theta}}_t)$, where

$$\mathcal{F}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}})}[\nabla_{\boldsymbol{\theta}} \log \hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}) \nabla_{\boldsymbol{\theta}} \log \hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}})^\mathsf{T}]$$

is the Fisher information matrix. This rule uses the natural Riemannian metric of distributions to normalize the effects of different parameterizations of the same distribution (Rattray et al., 1998).

KL Divergence and the Exponential Family

We show that for control distributions in the exponential family (Nielsen and Garcia, 2009), the Bregman divergence in Eq. (5.5) can be set to the KL divergence, which is a natural way to measure distances between distributions. Toward this end, we review the basics of the exponential family. We say a distribution $p_\eta(x)$ with natural parameter η of random variable x over some support $\mathcal{X}$ belongs to the *exponential family* if its probability function satisfies $p_\eta(x) = b(x)e^{\eta^\mathsf{T}\phi(x) - A(\eta)}$, where $\phi(x)$ is the sufficient statistics, $b(x)$ is the carrier

measure, and $A(\eta)$ is the log-partition function which satisfies

$$A(\eta) = \log \int_{\mathcal{X}} b(x) e^{\eta^\mathsf{T} \phi(x)} \, \mathrm{d}x.$$

The distribution p_η can also be described by its expectation parameter $\xi \triangleq \mathbb{E}_{p_\eta(x)}[\phi(x)]$, and there is a duality between the two parameterizations:

$$\xi = \nabla A(\eta) \quad \text{and} \quad \eta = \nabla A^*(\xi),$$

where $A^*(\xi) = \sup_{\eta \in \mathcal{H}} \eta^\mathsf{T} \xi - A(\eta)$ is the Legendre transformation of A and $\mathcal{H} = \{\eta : A(\eta) < \infty\}$. That is, $\nabla A = (\nabla A^*)^{-1}$. The duality results in the property below.

Fact 1. (Nielsen and Garcia, 2009) $\mathrm{KL}(p_\eta(x) \,\|\, p_{\eta'}(x)) = D_A(\eta' \,\|\, \eta) = D_{A^*}(\xi \,\|\, \xi')$.

We can use Fact 1 to define the Bregman divergence (Eq. (5.5)) to optimize a control distribution $\hat{\pi}_{\boldsymbol{\theta}}$ in the exponential family:

- if $\boldsymbol{\theta}$ is an expectation parameter, we can set $D_\psi(\boldsymbol{\theta} \,\|\, \tilde{\boldsymbol{\theta}}_t) \triangleq \mathrm{KL}(\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}) \,\|\, \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}))$, or
- if $\boldsymbol{\theta}$ is a natural parameter, we can set $D_\psi(\boldsymbol{\theta} \,\|\, \tilde{\boldsymbol{\theta}}_t) \triangleq \mathrm{KL}(\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}) \,\|\, \hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}))$.

We demonstrate some examples using this idea below.

Expectation Parameters and Categorical Distributions We first discuss the case where $\boldsymbol{\theta}$ is an expectation parameter and the first step in Eq. (5.5) is

$$\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \; \alpha \boldsymbol{g}_t^\mathsf{T} \boldsymbol{\theta} + \mathrm{KL}(\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t) \,\|\, \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)). \tag{5.16}$$

To illustrate, we consider an MPC problem with a *discrete* control space $\mathcal{U} = \{1, 2, \ldots, m\}$ and use the categorical distribution as the control distribution. Though we can model the full distribution over the planning horizon (i.e., each element corresponds to a control sequence from $\{1, \ldots, m\}^H$), this would require m^H parameters. This is intractable to rep-

resent for even moderate values of H, and the resulting estimate for $\boldsymbol{g}_t$ would have large variance. Instead, we model the distribution as a temporally independent sequence of categorical distributions, i.e.,

$$\hat{\pi}_{\boldsymbol{\theta}_t}(\hat{\boldsymbol{u}}_t) = \prod_{h=0}^{H-1} \hat{\pi}_{\theta_{t,h}}(\hat{u}_{t,h}),$$

where each $\hat{\pi}_{\theta_{t,h}}(\hat{u}_{t,h}) = \mathrm{Cat}(\hat{u}_{t,h}; \theta_{t,h})$, $\theta_{t,h} \in \Delta^m$ is the probability of choosing each control among $\{1, \ldots, m\}$ at the h^{th} predicted time step, and Δ^m denotes the probability simplex in $\mathbb{R}^m$. With this representation, we have mH parameters, and each $\theta_{t,h}$ is an expectation parameter of $\hat{\pi}_{\theta_{t,h}}$ that corresponds to the one-hot sufficient statistics

$$\phi(\hat{u}_{t,h}) = (\mathbf{1}\{\hat{u}_{t,h} = 1\}, \ldots, \mathbf{1}\{\hat{u}_{t,h} = m\}).$$

With the structure of Eq. (5.6), the update direction is

$$g_{t,h} = -\,\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[w_t(\hat{\boldsymbol{u}}_t)\, \phi(\hat{u}_{t,h}) \oslash \tilde{\theta}_{t,h}] \qquad (h = 0, 1, \ldots, H-1),$$

where $\oslash$ denotes elementwise division. The mirror descent update (Eq. (5.16)) then becomes the exponentiated gradient algorithm (Hazan, 2016; Kivinen and Warmuth, 1997):

$$\theta_{t,h} = \frac{1}{Z_{t,h}} \tilde{\theta}_{t,h} \odot e^{-\alpha g_{t,h}} \qquad (h = 0, 1, \ldots, H-1), \tag{5.17}$$

where $Z_{t,h}$ is the normalizer for $\theta_{t,h}$, $\odot$ denotes elementwise multiplication, and the exponentiation of the gradient is done elementwise. That is, instead of applying an additive gradient step to the parameters, the update in Eq. (5.17) exponentiates the gradient and performs elementwise multiplication. This does a better job of accounting for the geometry of the problem, and makes projection be a simple operation of normalizing a distribution.

Natural Parameters and Gaussian Distributions Alternatively, we can set $\boldsymbol{\theta}$ as a natural parameter and use

$$\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \ \alpha \boldsymbol{g}_t^\mathsf{T}\boldsymbol{\theta} + \mathrm{KL}(\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t) \,\|\, \hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}_t)) \tag{5.18}$$

as the first step in Eq. (5.5). In particular, we show that, with Eq. (5.18), the structure of the likelihood-ratio derivative in Eq. (5.6) can be leveraged to design an efficient update. The main idea follows from the observation that when the gradient is computed through Eq. (5.6) and $\tilde{\boldsymbol{\theta}}_t$ is the natural parameter, we have

$$\begin{aligned}\nabla_{\tilde{\boldsymbol{\theta}}_t} \log \hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t) &= \phi(\hat{\boldsymbol{u}}_t) - \nabla A(\tilde{\boldsymbol{\theta}}_t) \\ &= \phi(\hat{\boldsymbol{u}}_t) - \tilde{\boldsymbol{\xi}}_t\end{aligned}$$

so that the gradient $\boldsymbol{g}_t$ is:

$$\boldsymbol{g}_t = \nabla \ell_t(\tilde{\boldsymbol{\theta}}_t) = \mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t(\hat{\boldsymbol{u}}_t)}}[w_t(\hat{\boldsymbol{u}}_t)(\tilde{\boldsymbol{\xi}}_t - \phi(\hat{\boldsymbol{u}}_t))]. \tag{5.19}$$

We combine the factorization in Eq. (5.19) with a property of the proximal update below to derive our algorithm.

Proposition 1. *Let $\tilde{\boldsymbol{\xi}}_t \in \mathcal{M}$ and $\tilde{\boldsymbol{\eta}}_t = \nabla A^*(\tilde{\boldsymbol{\xi}}_t)$. Let $\boldsymbol{g}_t$ be an update vector and $\mathcal{M}$ be the image of $\mathcal{H}$ under ∇A. If $\tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t \in \mathcal{M}$ and we update the natural parameters via $\boldsymbol{\eta}_t = \arg\min_{\boldsymbol{\eta}\in\mathcal{H}} \ \alpha \boldsymbol{g}_t^\mathsf{T}\boldsymbol{\eta} + D_A(\boldsymbol{\eta} \,\|\, \tilde{\boldsymbol{\eta}}_t)$, then the equivalent update for the expectation parameters is $\boldsymbol{\xi}_t = \tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t$.*

Proof. We start by expanding the objective for $\boldsymbol{\eta}_t$:

$$\begin{aligned}
\boldsymbol{\eta}_t &= \arg\min_{\boldsymbol{\eta}\in\mathcal{H}} \alpha \boldsymbol{g}_t^\mathsf{T}\boldsymbol{\eta} + D_A(\boldsymbol{\eta} \,\|\, \tilde{\boldsymbol{\eta}}_t) \\
&= \arg\min_{\boldsymbol{\eta}\in\mathcal{H}} \alpha \boldsymbol{g}_t^\mathsf{T}\boldsymbol{\eta} + A(\boldsymbol{\eta}) - \nabla A(\tilde{\boldsymbol{\eta}}_t)^\mathsf{T}\boldsymbol{\eta} \\
&= \arg\min_{\boldsymbol{\eta}\in\mathcal{H}} (\alpha \boldsymbol{g}_t - \tilde{\boldsymbol{\xi}}_t)^\mathsf{T}\boldsymbol{\eta} + A(\boldsymbol{\eta}) \\
&= \arg\max_{\boldsymbol{\eta}\in\mathcal{H}} (\tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t)^\mathsf{T}\boldsymbol{\eta} - A(\boldsymbol{\eta}).
\end{aligned}$$

Letting $\boldsymbol{\eta}'_t = \nabla A^*(\tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t)$, the gradient of the objective evaluated at $\boldsymbol{\eta} = \boldsymbol{\eta}'_t$ is:

$$\begin{aligned}
\tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t - \nabla A(\boldsymbol{\eta}'_t) &= \tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t - \nabla A(\nabla A^*(\tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t)) \\
&= (\tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t) - (\tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t) \\
&= 0.
\end{aligned}$$

Since the objective's gradient at $\boldsymbol{\eta}'_t$ is zero and the objective is concave, we conclude that $\boldsymbol{\eta}'_t$ maximizes the objective, so that $\boldsymbol{\eta}_t = \boldsymbol{\eta}'_t = \nabla A^*(\tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t)$. Then, applying ∇A to both sides and using the fact that $\nabla A = (\nabla A^*)^{-1}$, we have $\boldsymbol{\xi}_t = \nabla A(\boldsymbol{\eta}_t) = \tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t$. □

Now, suppose we use the expected utility (Eq. (5.10)) as the loss function. We find that, under the assumption[7] in Proposition 1, the update rule in Eq. (5.18) becomes

$$\boldsymbol{\xi}_t = (1-\alpha)\tilde{\boldsymbol{\xi}}_t + \alpha\, \mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[w_t(\hat{\boldsymbol{u}}_t)\phi(\hat{\boldsymbol{u}}_t)], \tag{5.20}$$

where we take advantage of the fact that $\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[w_t(\hat{\boldsymbol{u}}_t)] = 1$ under the expected utility loss function (Table 5.1). In other words, when $\alpha \in [0, 1]$, the update to the expectation parameter $\boldsymbol{\xi}_t$ in Eq. (5.5) is simply a convex combination of the sufficient statistics and the previous expectation parameter $\tilde{\boldsymbol{\xi}}_t$.

[7] If $\tilde{\boldsymbol{\xi}}_t - \alpha \boldsymbol{g}_t$ is not in $\mathcal{M}$, the update in Eq. (5.18) needs to perform a projection, the form of which is algorithm dependent.

We provide a concrete example of an MPC algorithm that follows from Eq. (5.20). Let us consider a continuous control space and use the Gaussian distribution as the control distribution, i.e., we set $\hat{\pi}_{\boldsymbol{\theta}}(\hat{\boldsymbol{u}}) = \mathcal{N}(\hat{\boldsymbol{u}}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ for some mean $\boldsymbol{\mu} \in \mathbb{R}^{Hm}$ and covariance matrix $\boldsymbol{\Sigma} \succ 0$. We keep the covariance fixed and have the mean $\boldsymbol{\mu}$ be parameterized by the natural parameters $\boldsymbol{\theta}$. Under the exponential family parameterization, the sufficient statistics are $\phi(\hat{\boldsymbol{u}}) = \boldsymbol{\Sigma}^{-1/2}\hat{\boldsymbol{u}}$, implying the expectation parameters are $\boldsymbol{\xi} = \boldsymbol{\Sigma}^{-1/2}\boldsymbol{\mu}$. The nautral parameters are $\boldsymbol{\theta} = \boldsymbol{\eta} = \boldsymbol{\Sigma}^{-1/2}\boldsymbol{\mu}$. Substituting these into Eq. (5.20) and canceling out the common $\boldsymbol{\Sigma}^{-1/2}$ factor, the equivalent update rule based on the Gaussian's mean is:

$$\boldsymbol{\mu}_t = (1-\alpha)\tilde{\boldsymbol{\mu}}_t + \alpha\, \mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[w_t(\hat{\boldsymbol{u}}_t)\hat{\boldsymbol{u}}_t]. \tag{5.21}$$

Several existing algorithms are special cases of Eq. (5.21).

- *Cross-entropy method (CEM)* (Botev et al., 2013; Goschin et al., 2013):

  If we set ℓ_t to the threshold utility (Eq. (5.12)), then Eq. (5.21) becomes

  $$\boldsymbol{\mu}_t = (1-\alpha)\tilde{\boldsymbol{\mu}}_t + \alpha \frac{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[\mathbf{1}\{\hat{C}_t(\hat{\boldsymbol{u}}_t) \le C_{t,\max}\}\, \hat{\boldsymbol{u}}_t]}{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[\mathbf{1}\{\hat{C}_t(\hat{\boldsymbol{u}}_t) \le C_{t,\max}\}]}, \tag{5.22}$$

  which reduces to the cross-entropy method for fixed covariance Gaussians when $\alpha = 1$.

- *Model predictive path integral (MPPI)* (Williams, Wagener, et al., 2017):

  If we set ℓ_t to the exponential utility (Eq. (5.14)), then Eq. (5.21) becomes

  $$\boldsymbol{\mu}_t = (1-\alpha)\tilde{\boldsymbol{\mu}}_t + \alpha \frac{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}\, \hat{\boldsymbol{u}}_t]}{\mathbb{E}_{\hat{\pi}_{\tilde{\boldsymbol{\theta}}_t}(\hat{\boldsymbol{u}}_t)}[e^{-\hat{C}_t(\hat{\boldsymbol{u}}_t)/\lambda}]}, \tag{5.23}$$

  which reduces to the proximal MPPI update rule (Eq. (4.9)) when $\alpha = 1$.

5.3.3 Shift Model

While a variety of shift models Φ (Eq. (5.2)) are possible, we only consider the case where we have a temporally independent distribution. That is, we assume that $\hat{\pi}_{\boldsymbol{\theta}_t}$ factorizes as

$$\hat{\pi}_{\boldsymbol{\theta}_t}(\hat{\boldsymbol{u}}_t) = \prod_{h=0}^{H-1} \hat{\pi}_{\theta_{t,h}}(\hat{u}_{t,h}), \tag{5.24}$$

and $\boldsymbol{\theta}_t = (\theta_{t,0}, \theta_{t,1}, \ldots, \theta_{t,H-1})$ for some *basic control distribution* $\hat{\pi}_\theta$ parameterized by some $\theta \in \Theta$, where Θ denotes the feasible set for the basic control distribution. For control distributions of the form in Eq. (5.24), one valid shift operator Φ is:

$$\Phi((\theta_{t,0}, \theta_{t,1}, \ldots, \theta_{t,H-2}, \theta_{t,H-1})) = (\theta_{t,1}, \theta_{t,2}, \ldots, \theta_{t,H-1}, \bar{\theta}),$$

where $\bar{\theta}$ is some default parameter.

5.3.4 Extensions

In the previous sections, we discussed multiple instantiations of DMD-MPC, showing the flexibility of the proposed framework. But these are by no means exhaustive. The control distributions in DMD-MPC can be fairly general (in addition to the categorical and Gaussian distributions that we discussed) and control constraints on the problem (e.g., control limits) can be directly incorporated through proper choices of control distributions, such as the beta distribution, or through mapping the unconstrained control through some squashing function (e.g., $\tanh$ or clamp). Though our framework cannot directly handle state constraints as in constrained optimization approaches, a constraint can be relaxed to an indicator function which activates if the constraint is violated. The indicator function can then be added to the cost function c with some weight that encodes how strictly the constraint should be enforced.

Moreover, different integration techniques, such as Gaussian quadrature (Bellman and

Casti, 1971), can be adopted to replace the likelihood-ratio derivative in Eq. (5.6) for computing the required gradient direction.

5.4 Related Work

Recent work on MPC has studied sampling-based approaches, which are flexible in the sense that they do not require differentiability of a cost function. One such algorithm that can be used with general cost functions and dynamics is MPPI, which Williams, Wagener, et al. (2017) propose as a generalization of the control affine case (Williams, Drews, et al., 2016). The algorithm is derived by considering an optimal control distribution defined by the control problem. This optimal distribution is intractable to sample from, so the algorithm instead tries to bring a tractable distribution (in this case, Gaussian with fixed covariance) as close as possible in the sense of KL divergence. This ends up being the same as finding the mean of the optimal control distribution. The mean is then approximated as a weighted sum of sampled control trajectories, where the weight is determined by the exponentiated costs. Although this algorithm works well in practice (including a robust variant (Williams, Goldfain, et al., 2018) achieving state-of-the-art performance in aggressive driving (Drews et al., 2019)), it is not clear that matching the mean of the distribution should guarantee good performance, such as in the case of a multimodal optimal distribution. By contrast, our update rule in Eq. (5.23) results from optimizing an exponential utility.

A closely related approach is the cross-entropy method (CEM) (Botev et al., 2013), which also assumes a Gaussian sampling distribution but minimizes the KL divergence between the Gaussian distribution and a uniform distribution over low cost samples. CEM has found applicability in reinforcement learning (Mannor et al., 2003; Menache et al., 2005; Szita and Lörincz, 2006), motion planning (Helvik and Wittner, 2002; Kobilarov, 2012), and MPC (Chua et al., 2018; Williams, Drews, et al., 2018; Finn and Levine, 2017; Yang et al., 2020).

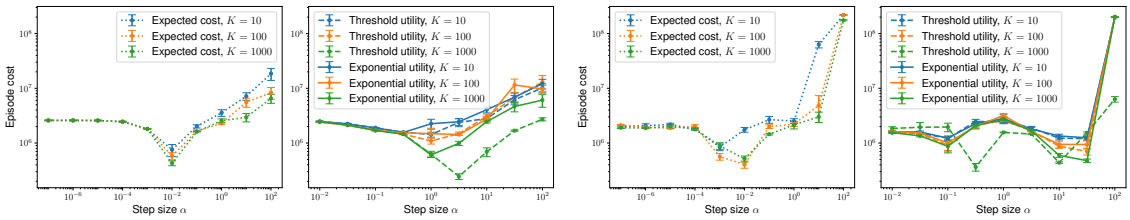
These sampling-based control algorithms can be considered special cases of general derivative-free optimization algorithms, such as covariance matrix adaptation evolutionary strategies (CMA-ES) (Hansen et al., 2003) and natural evolutionary strategies (NES) (Wierstra et al., 2014). CMA-ES samples points from a multivariate Gaussian, evaluates their fitness, and adapts the mean and covariance of the sampling distribution accordingly. On the other hand, NES optimizes the parameters of the sampling distribution to maximize some expected fitness through steepest ascent, where the direction is provided by the natural gradient. Akimoto et al. (2012) show that CMA-ES can also be interpreted as taking a natural gradient step on the parameters of the sampling distribution. As we show in Section 5.3.2, natural gradient descent is a special case of the DMD-MPC framework. Okada and Taniguchi (2018) make a similar observation that connects MPPI with mirror descent, but their derivation is limited to the KL divergence and Gaussian case.

5.5 Experiments

We use experiments to validate the flexibility of DMD-MPC. We show that this framework can handle both continuous (Gaussian distribution) and discrete (categorial distribution) variations of control problems, and that MPC algorithms like MPPI and CEM can be generalized using different step sizes and control distributions to improve performance. Additional system and task details are included in Chapter A.

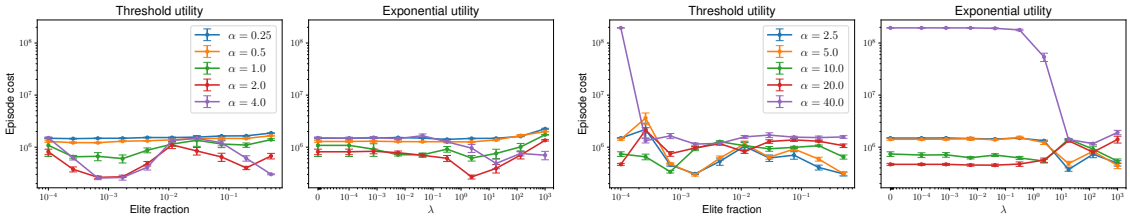
5.5.1 Cartpole

We first consider the classic cartpole problem where we seek to swing a pole upright and keep it balanced only using actuation on the attached cart. We consider both the continuous and discrete control variants. For the continuous case, we choose the Gaussian distribution as the control distribution and keep the covariance fixed. For the discrete case, we choose the categorical distribution and use update Eq. (5.17). In either case, we have access to a biased stochastic model (uses a different pole length compared to the real cart).



(a) Continuous controls (b) Discrete controls

Figure 5.4: Varying step size α and number of samples K (same legends for (a) and (b)). Threshold utility Eq. (5.12) uses elite fraction $= 10^{-3}$. Exponential utility Eq. (5.14) uses $\lambda = 1$.



(a) Continuous controls (b) Discrete controls

Figure 5.5: Varying loss parameter and step size ($K = 1000$).

We consider the interaction between the choice of loss ℓ_t, step size α, and number of samples K used to estimate Eq. (5.6),[8] shown in Figs. 5.4 and 5.5. For this environment, we can achieve low cost when optimizing the expected cost in Eq. (5.8) with a proper step size (10^{-2} for both continuous and discrete problems) while being fairly robust to the number of samples. When using either of the utilities, the number of samples is more crucial in the continuous domain, with more samples allowing for larger step sizes. In the discrete domain (Fig. 5.4b), performance is largely unaffected by the number of samples when the step size is below 10, excluding the threshold utility with $K = 1000$ samples. In Fig. 5.5a, for a large range of utility parameters, we see that using step sizes above 1 (the step size set in MPPI and CEM) give significant performance gains. In Fig. 5.5b, there's a more complicated interaction between the utility parameter and step size, with huge changes in cost when altering the utility parameter and keeping the step size fixed.

[8]For our experiments, we vary the number of samples K from $\hat{\pi}_{\boldsymbol{\theta}}$ and fix the number of samples from $\hat{p}$ to 10. Furthermore, we use common random numbers when sampling from $\hat{p}$ to reduce estimation variance.

Figure 5.6: AutoRally car.

5.5.2 AutoRally

Platform Description

We used the autonomous AutoRally platform (Goldfain et al., 2019) to run a high-speed driving task on a dirt track, with the goal of the task being to achieve as low a lap time as possible. The robot (Fig. 5.6) is a 1:5 scale RC chassis capable of driving over $20\ \mathrm{m/s}$ ($45\ \mathrm{mph}$) and has an Intel i7-7700 CPU and Nvidia GTX 1050 Ti GPU. Our code for the control algorithm is based on modifications of code available on the AutoRally repository.[9] For real-world experiments, we estimated the car's pose using a particle filter from Drews et al. (2019) which relies on a monocular camera, IMU, and GPS. In both simulated and real-world experiments, the dynamics model is a neural network which has been fitted to data collected from human demonstrations. We note that the dynamics model is deterministic, so we don't need to estimate any expectations with respect to the dynamics.

Simulated Experiments

We first used the Gazebo simulator (Fig. 5.7) from the AutoRally repo to perform a sweep of algorithm parameters, particularly the step size α and number of samples K, to evaluate how changing these parameters can affect the performance of DMD-MPC. For all of the

[9] `https://github.com/AutoRally/autorally`

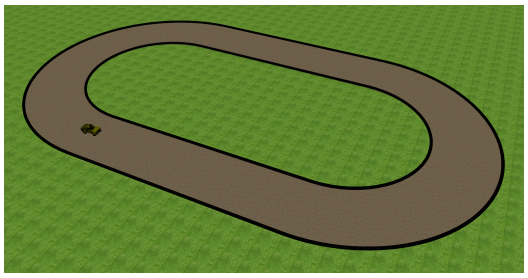


Figure 5.7: Simulated AutoRally task.

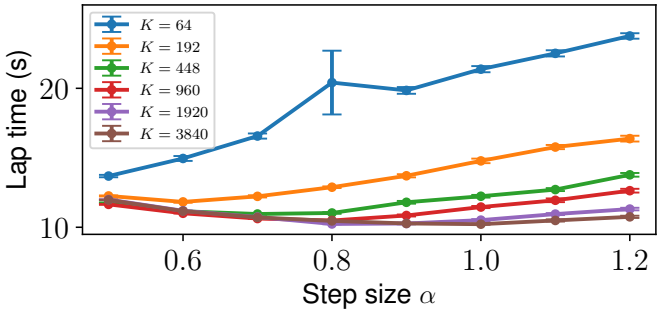


Figure 5.8: Simulated AutoRally performance with different step sizes and number of samples. Though many samples coupled with large steps yield the smallest lap times, the performance gains are small past $K = 1920$ samples. With fewer samples, a lower step size helps recover some lost performance.

experiments, the control distribution was a Gaussian with fixed covariance, and we used update Eq. (5.23) (i.e., the loss is the exponential utility Eq. (5.14)) with $\lambda = 6.67$. The resulting lap times are shown in Fig. 5.8.[10] We see that although using more samples does result in smaller lap times, there are diminishing returns past $K = 1920$ samples. Indeed, with a proper step size, even as few as 192 samples can yield lap times within a couple seconds of 3840 samples and a step size of 1. We also observe that the curves converge as the step size decreases further, implying that only a certain number of samples are needed for a given step size. This is a particularly important advantage of DMD-MPC over methods like MPPI: by changing the step size, DMD-MPC can perform much more effectively with fewer samples, making it a good choice for embedded systems which can't produce many samples due to computational constraints.

[10]The large error bar for $K = 64$ samples and step size $\alpha = 0.8$ is due to one particular lap where the car stalled at a turn for about 60 seconds.

Table 5.2: Statistics for real-world experiments at target of $9\ \mathrm{m/s}$.

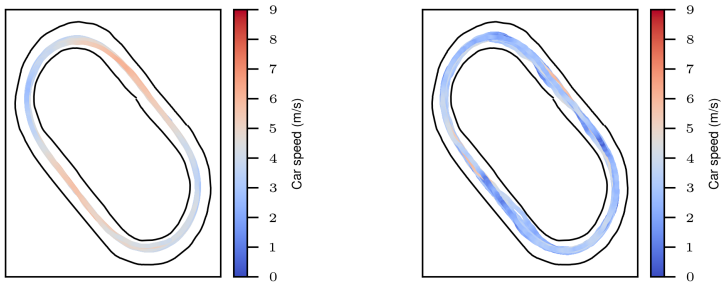
Samples K	Step size α	Lap time (s)	Avg. speed ($\mathrm{m/s}$)	Max speed ($\mathrm{m/s}$)
1920	1	31.76 ± 0.55	5.70 ± 0.16	9.21 ± 0.30
	0.8	31.81 ± 0.21	5.75 ± 0.03	9.03 ± 0.19
	0.6	32.83 ± 0.31	5.60 ± 0.05	8.62 ± 0.12
64	1	33.74 ± 0.78	5.45 ± 0.16	9.50 ± 0.22
	0.8	33.84 ± 0.80	5.46 ± 0.11	9.12 ± 0.26
	0.6	33.61 ± 0.74	5.50 ± 0.13	9.14 ± 0.42

We qualitatively evaluate two particular extremes: few vs. many samples (64 vs. 3840) and small vs. large step size (0.5 vs. 1) by looking at the path and speed of the car during the episode (Fig. 5.9). At small step sizes (Figs. 5.9a and 5.9c), the path and speed profiles are rather similar, while with few samples and a large step size (Fig. 5.9b), the car drives much more slowly and erratically, sometimes even stopping. In the ideal scenario with many samples and a large step size, the car can achieve consistently high speed while driving smoothly (Fig. 5.9d).

We also experimented with optimizing the expected cost (Eq. (5.8)) and found performance is dramatically worse (Fig. 5.10), even when using $K = 3840$ samples per gradient. At best, the car drove in the center of the track at speeds below $4\ \mathrm{m/s}$ (Fig. 5.10c), and at worst, the car either slowly drove along the track walls (Fig. 5.10a) or the controller eventually produced NaN controls that prematurely ended the experiment (Fig. 5.10d). This poor performance is likely due to most samples in the estimate of Eq. (5.9) having very high cost (e.g., due to leaving the track) and contributing significantly to the gradient estimate. On the other hand, when estimating Eq. (5.15), as in the experiments in Section 5.5.2, these high cost trajectories are assigned very low weights so that only low cost trajectories contribute to the gradient estimate.

Real-World Experiments

In the real-world setting (Fig. 5.11), the control distribution was a Gaussian with fixed covariance, and we used update Eq. (5.23) with $\lambda = 8$. We ran two sets of experiments,



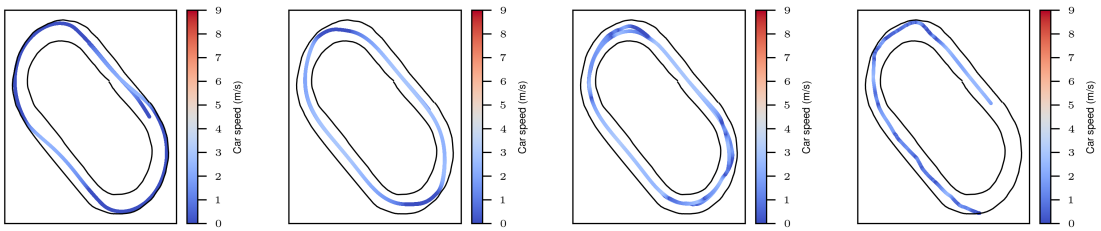
(a) $K = 64$, $\alpha = 0.5$

(b) $K = 64$, $\alpha = 1$

(c) $K = 3840$, $\alpha = 0.5$

(d) $K = 3840$, $\alpha = 1$

Figure 5.9: Simulated car speeds when optimizing the exponential utility (Eq. (5.14)). The speeds and trajectories are very similar at step size 0.5, irrespective of the number of samples. At step size 1, though, 64 samples result in capricious maneuvers and low speeds, whereas 3840 samples result in smooth driving at high speeds.



(a) $\alpha = 0.025$ (b) $\alpha = 0.05$ (c) $\alpha = 0.075$ (d) $\alpha = 0.1$

Figure 5.10: Simulated car speeds when optimizing the expected cost (Eq. (5.8)). All tested step sizes result in low speeds. At too low or too high of a step size, the car will drive along the wall or crash into it.

Figure 5.11: Real-world AutoRally task.

Table 5.3: Statistics for real-world experiments at target of $11\ \mathrm{m/s}$.

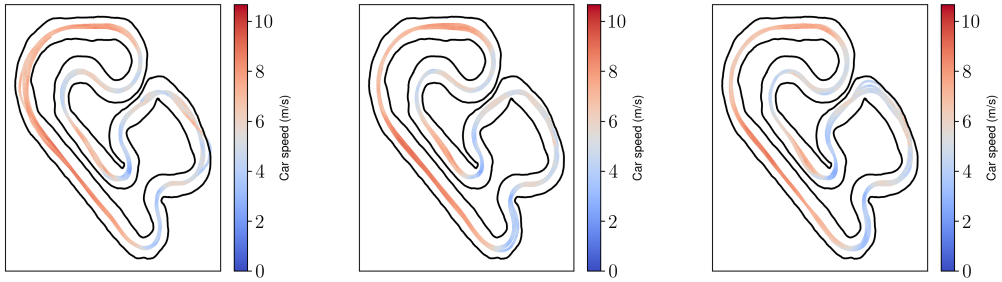
Samples K	Step size α	Lap time (s)	Avg. speed ($\mathrm{m/s}$)	Max speed ($\mathrm{m/s}$)
64	1	31.05 ± 0.67	5.80 ± 0.26	10.17 ± 0.30
	0.6	30.30 ± 0.56	5.98 ± 0.15	10.30 ± 0.05

each with a different target speed: one at $9\ \mathrm{m/s}$ and the other at $11\ \mathrm{m/s}$.

For the first set of experiments (target of $9\ \mathrm{m/s}$), we used the following configurations: each of 1920 and 64 samples, and each of step sizes 1 (corresponding to MPPI), 0.8, and 0.6.[11] Overall (Table 5.2), there was a mild degradation in performance when decreasing the step size at $K = 1920$ samples, due to the car taking a longer path on the track (Fig. 5.12a vs. Fig. 5.12c). With $K = 64$ samples, the results seem unaffected by the step size. This could be because, despite the noisiness of the DMD-MPC update, the set-point controller in the car's steering servo acts as a filter, smoothing out the control signal and allowing the car to drive on a consistent path (Fig. 5.13). Videos of this experiment can be found at `https://youtu.be/vZST3v0_S9w`.

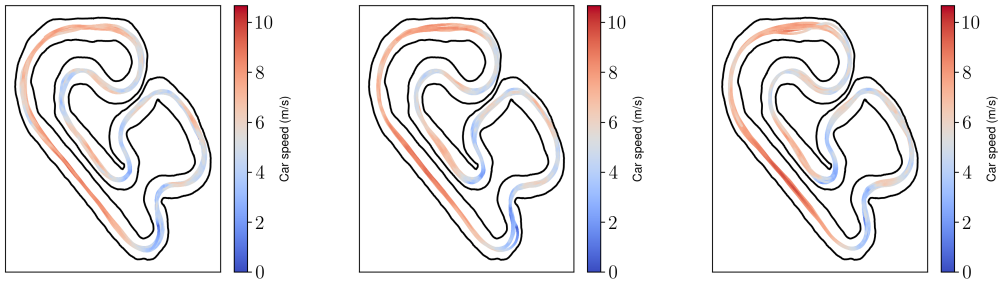
For the second set of experiments (target of $11\ \mathrm{m/s}$), we fixed the number of samples K at 64 and used step sizes of 1 (corresponding to MPPI) and 0.6. The statistics slightly improved with a decreased step size (Table 5.3), but qualitatively there was a larger difference between the step sizes. With a step size of 1, the car often wobbled while driving, turns around at one point, and crashes in one of the trials (Fig. 5.14a). On the other hand, with a step size of 0.6, the car drove much more smoothly and succeeded at the aggressive

[11] Due to weaker batteries used with 64 samples, results should not be compared across number of samples.



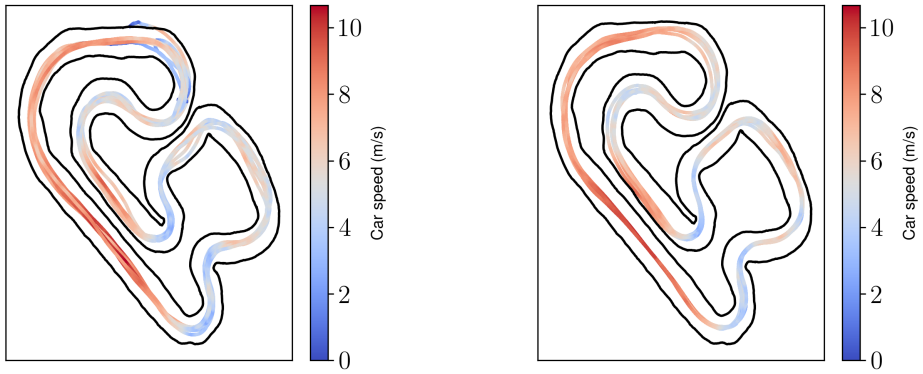
(a) $\alpha = 1$ (b) $\alpha = 0.8$ (c) $\alpha = 0.6$

Figure 5.12: Real-world car speeds with $K = 1920$ samples and target of $9\ \mathrm{m/s}$.



(a) $\alpha = 1$ (b) $\alpha = 0.8$ (c) $\alpha = 0.6$

Figure 5.13: Real-world car speeds with $K = 64$ samples and target of $9\ \mathrm{m/s}$.



(a) $\alpha = 1$ (b) $\alpha = 0.6$

Figure 5.14: Real-world car speeds with $K = 64$ samples and target of $11\ \mathrm{m/s}$. In Fig. 5.14a, note the crash and U-turn at the top of the plot as well as the wider spread of the paths throughout the whole track. By contrast, in Fig. 5.14b, the resulting paths are more consistent, and there are no failure points.

driving task with no issues (Fig. 5.14b). Despite the smoothing effect of the low-level controllers in the car, the more stringent costs associated with the larger target speed caused the noisiness of the DMD-MPC update to manifest in the car's performance when using a step size of 1. A smaller step size mitigated this noisiness. Videos of this experiment are available at `https://youtu.be/MhuqiHo2t98`.

5.6 Conclusion

We presented a connection between model predictive control and online learning. From this connection, we proposed an algorithm based on dynamic mirror descent that can work for a wide variety of settings and cost functions. We also discussed the choice of loss function within this online learning framework and the sort of preference each loss function imposes. From this general algorithm and assortment of loss functions, we show several well known algorithms are special cases and presented a general update for members of the exponential family.

We empirically validated our algorithm on continuous and discrete simulated problems and on a real-world aggressive driving task. In the process, we also studied the parameter choices within the framework, finding, for example, that in our framework a smaller number of rollout samples can be compensated for by varying other parameters like the step size.

We hope that the online learning and stochastic optimization viewpoints of MPC presented in this chapter opens up new possibilities for using tools from these domains, such as alternative efficient sampling techniques (Bellman and Casti, 1971) and accelerated optimization methods (Miyashita et al., 2018; Okada and Taniguchi, 2018), to derive new MPC algorithms that perform well in practice.

Part II

Reinforcement Learning

CHAPTER 6

PRELIMINARIES

This section builds upon the material presented in Section 2.1 and Section 2.1.1.

Reinforcement learning (Sutton and Barto, 2018) seeks to have an agent to learn good behaviors via trial-and-error interactions with an environment. In particular, it seeks a policy π that maximizes the expected sum of discounted rewards in the environment (Eq. (2.2)).

There are several quantities related to π that are useful for policy optimization. The value function V^π of policy π:

$$V^\pi(s) \triangleq \mathbb{E}_{\rho^\pi(\tau|s_0=s)}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right].$$

The state-action value function Q^π of policy π is:

$$Q^\pi(s,a) \triangleq \mathbb{E}_{\rho^\pi(\tau|s_0=s,a_0=a)}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right]$$

One consequence of these definitions is that $V^\pi(s) = \mathbb{E}_{\pi(a|s)}[Q^\pi(s,a)]$. We also define the advantage function A^π of policy π as:

$$A^\pi(s,a) \triangleq Q^\pi(s,a) - V^\pi(s).$$

Let $\pi^\star$ be an optimal policy of $\mathcal{M}$ (i.e., a policy that solves Eq. (2.2)). We then define the optimal value function $V^\star$ and optimal state-action value function $Q^\star$ as $V^\star = V^{\pi^\star}$ and $Q^\star = Q^{\pi^\star}$, respectively.

We define the state visitation distribution d^π under π as:

$$d^\pi(s) \triangleq (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\rho^\pi(s_t=s).$$

We similarly define the state-action visitation distribution as $d^\pi(s,a) \triangleq d^\pi(s)\pi(a|s)$.

6.1 Proximal Policy Optimization (PPO)

In the following chapters, we will use the proximal policy optimization (PPO) algorithm (Schulman, Wolski, et al., 2017) to optimize the policy π. PPO is an on-policy algorithm that alternates between rolling out π and updating π using the collected data. We represent the policy with a neural network π_θ (with parameters θ) and the value function with a neural network V_ϕ (with parameters ϕ). At some iteration k, we have data collection policy π_{θ_k}. Given some state s and action a, the PPO objective function consists of three terms:

- Policy optimization term:

$$\ell_k^\pi(\theta, s, a) = -\min\left\{ \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \hat{A}(s,a),\ \text{clip}\left\{ \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\varepsilon \right\} \hat{A}(s,a) \right\}$$

  Here, ε is a clipping hyperparameter, and $\hat{A}(s,a)$ is an estimate of the advantage function of π_{θ_k}. The estimate is usually found by generalized advantage estimation (Schulman, Moritz, et al., 2016), a temporal difference method that uses the learned value function V_ϕ and empirical returns from rollouts. This optimization term updates π_θ towards good actions (as measured by $\hat{A}$), while the clipping keeps π_θ from deviating too far from the collection policy π_{θ_k}.

- Value function term:

$$\ell_k^V(\phi, s) = (V_\phi(s) - \hat{V}_k(s))^2$$

  This term updates the value estimator V_ϕ to match the empirical value $\hat{V}_k$ of π_{θ_k} calculated from the rollouts.

- Policy entropy term:

$$\ell_k^{\mathcal{H}}(\theta, s) = -\mathcal{H}(\pi_\theta(a|s)) = \mathbb{E}_{\pi_\theta(a|s)}[\log \pi_\theta(a|s)]$$

This term increases the entropy of the policy π_θ to prevent the policy from becoming deterministic too quickly, which helps maintain exploration.

Putting these all together, the PPO objective function at iteration k is:

$$\ell_k(\theta, \phi) = \mathbb{E}_{d^{\pi_{\theta_k}}(s,a)}[\ell_k^\pi(\theta, s, a) + w_V \ell_k^V(\phi, s) + w_\mathcal{H} \ell_k^\mathcal{H}(\theta, s)],$$

where w_V and $w_\mathcal{H}$ are weighting hyperparameters. We use the Adam optimizer (Kingma and Ba, 2015) for this objective. Because we use this optimizer and have separate networks for the policy and value function, we can arbitrarily set $w_V = 1$ and not affect the learning dynamics.

6.2 Chance-Constrained Markov Decision Processes

Many robotic tasks partition the state space $\mathcal{S}$ into a set $\mathcal{S}_{\text{unsafe}}$ of undesired states and its complement $\mathcal{S}_{\text{safe}}$. We assume $\mathcal{S}_{\text{unsafe}}$ is absorbing. For example, for a robot we may define $\mathcal{S}_{\text{unsafe}}$ as the states where the robot is fallen over. We assume for simplicity of presentation that $\mathcal{S}_{\text{unsafe}}$ is a set of two states $\{s_\triangleright, s_\circ\}$ which follows the dynamics visualized in Fig. 6.1. That is, we assume that when going from $\mathcal{S}_{\text{safe}}$ to $\mathcal{S}_{\text{unsafe}}$ we first visit the "violation" state $s_\triangleright$ and then transition to an absorbing state $s_\circ$.

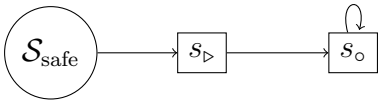


Figure 6.1: Absorbing property of $\mathcal{S}_{\text{unsafe}}$

Naturally, we would like our policy π to rarely visit the unsafe set. Defining $c(s, a) = \mathbf{1}\{s = s_\triangleright\}$ as a cost function that indicates whether we visit the violation state and δ as the tolerated violation probability, we want to solve the following chance-constrained policy

optimization problem (Altman, 1999):

$$\max_{\pi} \quad \mathbb{E}_{\rho^{\pi}(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] \quad \text{subject to} \quad \mathbb{E}_{\rho^{\pi}(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)\right] \leq \delta. \tag{6.1}$$

6.2.1 Safe Reinforcement Learning

Despite the ability of reinforcement learning to eventually produce good policies, we may desire to avoid going to undesirable states as much as possible during training. In the context of the chance-constrained problem in Eq. (6.1), we want to satisfy the safety constraint even during training, despite the objective only specifying the constraint for the returned policy π.

Safe reinforcement learning (García and Fernández, 2015; Amodei et al., 2016) studies the problem of designing learning agents with this constraint in mind. Many safe RL approaches tackle the safety requirement in one of two ways:

- *Constrained RL*: These approaches directly solve the constrained optimization problem in Eq. (6.1), e.g., by introducing a Lagrange multiplier w for the constraint and solving the resultant minimax problem:

$$\max_{\pi} \min_{w \geq 0} \quad \mathbb{E}_{\rho^{\pi}(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t) - w c(s_t, a_t))\right] + w\delta. \tag{6.2}$$

  Though these approaches can ultimately learn a policy that satisfies the safety constraint, there are usually no guarantees on safety during training.

- *Preemptive intervention*: Roughly speaking, a safety layer is wrapped around the RL policy π, whereupon it filters out any proposed actions that are deemed unsafe. Thus, we (ideally) avoid the unsafe set during the exploration process. The safety layer can be quite general, ranging from simple heuristics (e.g., if a bipedal robot is tilted too far forward, a safety gantry lifts the robot off the ground) to control-theoretic techniques like control barrier functions.

Model-free and model-based techniques may be used in either approach. We discuss examples of both approaches in Section 7.4.

CHAPTER 7
SAFE REINFORCEMENT LEARNING USING ADVANTAGE-BASED INTERVENTION

Many sequential decision problems involve finding a policy that maximizes total reward while obeying safety constraints. Although much recent research has focused on the development of safe reinforcement learning (RL) algorithms that produce a safe policy after training, ensuring safety *during* training as well remains an open problem. A fundamental challenge is performing exploration while still satisfying constraints in an unknown Markov decision process (MDP). In this chapter, we address this problem for the chance-constrained setting. We propose a new algorithm, SAILR, that uses an intervention mechanism based on advantage functions to keep the agent safe throughout training and optimizes the agent's policy using off-the-shelf RL algorithms designed for unconstrained MDPs. Our method comes with strong guarantees on safety during *both* training and deployment (i.e., after training and without the intervention mechanism) and policy performance compared to the optimal safety-constrained policy. In our experiments, we show that SAILR violates constraints far less frequently during training than standard safe RL and constrained MDP approaches and converges to a well-performing policy that can be deployed safely without intervention. Our code is available at `https://github.com/nolanwagener/safe_rl`.

7.1 Introduction

Reinforcement learning (RL) (Sutton and Barto, 2018) enables an agent to learn good behaviors with high returns through interactions with an environment of interest. However, in many settings, we want the agent not only to find a high-return policy but also avoid undesirable states as much as possible, even during training. For example, in a bipedal locomotion task, we do not want the robot to fall over and risk damaging itself either during

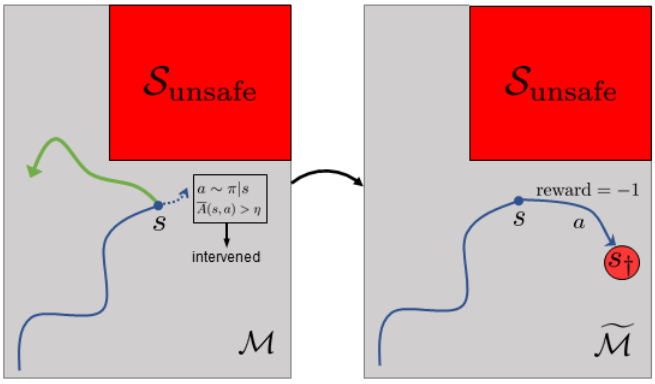


Figure 7.1: Advantage-based intervention of SAILR and construction of the surrogate MDP $\tilde{\mathcal{M}}$. In $\mathcal{M}$, whenever the policy π proposes an action a which is disadvantageous (w.r.t. a backup policy μ) in terms of safety, μ intervenes and guides the agent to safety (green path). From the perspective of π, it transitions to an absorbing state $s_\dagger$ and receives a penalizing reward of -1.

training or deployment. Maintaining safety while exploring an unknown environment is challenging, because venturing into new regions of the state space may carry a chance of a costly failure.

Safe reinforcement learning (García and Fernández, 2015; Amodei et al., 2016) studies the problem of designing learning agents for sequential decision-making with this challenge in mind. Most safe RL approaches tackle the safety requirement either by framing the problem as a constrained Markov decision process (CMDP) (Altman, 1999) or by using control-theoretic tools to restrict the actions that the learner can take. However, due to the natural conflict between learning, maximizing long-term reward, and satisfying safety constraints, these approaches make different performance trade-offs.

CMDP-based approaches (Borkar, 2005; Achiam et al., 2017; Le et al., 2019) take inspiration from existing constrained optimization algorithms for non-sequential problems, notably the Lagrangian method (Bertsekas, 2014). The most prominent examples (Chow, Ghavamzadeh, et al., 2017; Tessler et al., 2018) rely on first-order primal-dual optimization to solve a stochastic nonconvex saddle-point problem. Though they eventually produce a safe policy, such approaches provide no guarantees on policy safety during training.

Other safe RL approaches (Achiam et al., 2017; Le et al., 2019; Bharadhwaj et al., 2021) conservatively enforce safety constraints on every policy iterate by solving a constrained optimization problem, but they can be difficult to scale due to their high computational complexity. All of the above methods suffer from numerical instability originating in solving the stochastic nonconvex saddle-point problems (Facchinei and Pang, 2007; Lin et al., 2020); consequently, they are less robust than typical unconstrained RL algorithms.

Control-theoretic approaches to safe RL use interventions, projections, or planning (Hans et al., 2008; Wabersich and Zeilinger, 2021; Dalal et al., 2018; Berkenkamp et al., 2017) to enforce safe interactions between the agent and the environment, independent of the policy the agent uses. The idea is to use domain-specific heuristics to decide whether an action proposed by the agent's policy can be safely executed. However, some of these algorithms do not allow the agent to *learn* to be safe after training (Wabersich and Zeilinger, 2021; Hans et al., 2008; Polo and Rebollo, 2011), so they may not be applicable in scenarios where the control mechanism relies on resouces only available during training (such as computationally demanding online planning). It is also often unclear how these policies perform compared to the optimal policy in the CMDP-based approach.

In this chapter, we propose a new algorithm, SAILR (*Safe Advantage-based Intervention for Learning policies with Reinforcement*), that uses a novel advantage-based intervention rule to enable safe and stable RL for *general* MDPs. Our method comes with strong guarantees on safety during *both* training and deployment (i.e., after training and without the intervention mechanism) and has good on-policy performance compared to the optimal safety-constrained policy. Specifically, SAILR trains the agent's policy by calling an off-the-shelf RL algorithm designed for standard *unconstrained* MDPs. In each iteration, SAILR:

1. queries the base RL algorithm to get a data-collection policy,
2. runs the policy in the MDP while utilizing the advantage-based intervention rule to ensure safe interactions (and executes a backup policy upon intervention to ensure

safety),

3. transforms the collected data into experiences in a new unconstrained MDP that penalizes any visits of intervened state-actions (visualized in Fig. 7.1), and

4. gives the transformed data to the base RL algorithm to perform policy optimization.

Under very mild assumptions on the MDP and the safety of the backup policy used during the intervention,[1] we prove that running SAILR with *any* RL algorithm for unconstrained MDPs can safely learn a policy that has good performance in the safety-constrained MDP (with a bias propotional to how often the true optimal policy would be overridden by our intervention mechanism). Compared with existing work, SAILR is easier to implement and runs more reliably than the CMDP-based approaches. In addition, since we only rely on estimated advantage functions, our approach is also more generic than the aforementioned control-theoretic approaches which make assumptions on smoothness or ergodicity of the problem.

We also empirically validate our theory by comparing SAILR with several standard safe RL algorithms in simulated robotics tasks. The encouraging experimental results strongly support the theory: SAILR can learn safe policies with competitive performance using a standard unconstrained RL algorithm, PPO (Schulman, Wolski, et al., 2017), while incurring only a small fraction of unsafe training rollouts compared to the baselines.

7.2 Preliminaries

7.2.1 Notation

Here, we adopt the notation introduced in Chapter 6. We assume the reward function $r(s, a)$ lies in $[0, 1]$ and that $\mathcal{S}$ and $\mathcal{A}$ can be either discrete or continuous. We use the following overloaded notation: For a state distribution $d \in \Delta(\mathcal{S})$ and a function $f : \mathcal{S} \to \mathbb{R}$, we define

[1] We only assume that the unsafe states are absorbing and that the backup policy is safe from the initial state with high probability. We do *not* assume that the backup policy can achieve high rewards.

$f(d) \triangleq \mathbb{E}_{d(s)}[f(s)]$; similarly, for a policy π and a function $g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we define $g(s, \pi) \triangleq \mathbb{E}_{\pi(a|s)}[g(s, a)]$. Finally, later in the chapter we will consider multiple variants of an MDP (specifically, $\mathcal{M}$, $\bar{\mathcal{M}}$, and $\tilde{\mathcal{M}}$) and will use the decorative symbol on the MDP notation to distinguish similar objects from different MDPs (e.g., V^π and $\bar{V}^\pi$ will denote the state value functions of π in $\mathcal{M}$ and $\bar{\mathcal{M}}$, respectively).

7.2.2 Safe Reinforcment Learning

We consider safe RL in a γ-discounted infinite horizon MDP $\mathcal{M}$, where safety means that the probability of the agent entering an unsafe subset $\mathcal{S}_{\text{unsafe}} \subset \mathcal{S}$ is low. We assume that we know the unsafe subset $\mathcal{S}_{\text{unsafe}}$ and the safe subset $\mathcal{S}_{\text{safe}} \triangleq \mathcal{S} \setminus \mathcal{S}_{\text{unsafe}}$. However, we make no assumption on the knowledge of the reward r and the dynamics p, except that the reward r is zero on $\mathcal{S}_{\text{unsafe}}$ and that $\mathcal{S}_{\text{unsafe}}$ is absorbing: once the agent enters $\mathcal{S}_{\text{unsafe}}$ in a rollout, it cannot travel back to $\mathcal{S}_{\text{safe}}$ and stays in $\mathcal{S}_{\text{unsafe}}$ for the rest of the rollout.

Objective Our goal is to find a policy π that is safe and has a high return in $\mathcal{M}$, and to do so via a safe data collection process. Specifically, while keeping the agent safe during exploration, we want to solve the following chance-constrained policy optimization problem:

$$\begin{aligned} \max_{\pi} \quad & V^\pi(d_0) \\ \text{subject to} \quad & (1-\gamma) \sum_{h=0}^{\infty} \gamma^h \, \mathbb{P}_{\rho^\pi(\tau)}(\tau^h \subset \mathcal{S}_{\text{safe}}) \geq 1 - \delta, \end{aligned} \tag{7.1}$$

where $\delta \in [0, 1]$ is the tolerated failure probability, $\tau^h = (s_0, a_0, \ldots, s_{h-1}, a_{h-1})$ denotes an h-step trajectory segment, and $\mathbb{P}_{\rho^\pi(\tau)}(\tau^h \subset \mathcal{S}_{\text{safe}})$ denotes the probability of τ^h being safe (i.e., not entering $\mathcal{S}_{\text{unsafe}}$ from time step 0 to $h-1$) under the trajectory distribution ρ^π of π on $\mathcal{M}$.[2]

We desire the agent to provide anytime safety during *both* training and deployment. During training, the agent can interact with the unknown MDP $\mathcal{M}$ to collect data under

[2]We abuse the notation $\tau^h \subset \mathcal{S}_{\text{safe}}$ to mean that $s_t \in \mathcal{S}_{\text{safe}}$ for each s_t in $\tau^h = (s_0, a_0, \ldots, s_{h-1}, a_{h-1})$.

a training budget, such as the maximum number of environment interactions or allowed unsafe trajectories the agent can generate. Once the budget is used up, training stops, and an approximate solution of Eq. (7.1) needs to be returned.

The constraint in Eq. (7.1) is known as a *chance constraint*. The definition here accords to an exponentially weighted average (based on the discount factor γ) of trajectory safety probabilities of different horizons. This weighted average concept arises naturally in γ-discounted MDPs, because the objective in Eq. (7.1) can also be written as a weighted average of undiscounted expected returns, i.e., $V^\pi(d_0) = (1-\gamma)\sum_{h=0}^{\infty}\gamma^h U_h^\pi(d_0)$, where $U_h^\pi(d_0) \triangleq \mathbb{E}_{\rho^\pi(\tau)}[\sum_{t=0}^{h} r(s_t, a_t)]$.

CMDP Formulation The chance-constrained policy optimization problem in Eq. (7.1) can be formulated as a constrained Markov decision process (CMDP) problem (Altman, 1999; Chow, Ghavamzadeh, et al., 2017). For the mathematical convenience of defining and analyzing the equivalence between Eq. (7.1) and a CMDP, instead of treating $\mathcal{S}_{\text{unsafe}}$ as a single meta-absorbing state, without loss of generality we define $\mathcal{S}_{\text{unsafe}} \triangleq \{s_\triangleright, s_\circ\}$. The semantics of this set is that when an agent leaves $\mathcal{S}_{\text{safe}}$ and enters $\mathcal{S}_{\text{unsafe}}$, it first goes to $s_\triangleright$ and, regardless of which action it takes at $s_\triangleright$, it then goes to the absorbing state $s_\circ$ and stays there forever. We can view $s_\triangleright$ as a meta-state that summarizes the unsafe region in a given RL application (e.g., a biped robot falling on the ground) and $s_\circ$ as a fictitious state that captures the absorbing property of $\mathcal{S}_{\text{unsafe}}$.

For an MDP $\mathcal{M}$ with an unsafe set $\mathcal{S}_{\text{unsafe}} \triangleq \{s_\triangleright, s_\circ\}$, define the cost $c(s,a) \triangleq \mathbf{1}\{s = s_\triangleright\}$, where $\mathbf{1}$ denotes the indicator function. Then we can define a CMDP $(\mathcal{S}, \mathcal{A}, p, r, c, \gamma)$ using a reward-based MDP $\mathcal{M} \triangleq (\mathcal{S}, \mathcal{A}, p, r, \gamma)$ and a cost-based MDP $\bar{\mathcal{M}} \triangleq (\mathcal{S}, \mathcal{A}, p, c, \gamma)$. Using these new definitions, we can write the chance-constrained policy optimization in Eq. (7.1) as a CMDP problem:

$$\max_\pi \quad V^\pi(d_0) \quad \text{subject to} \quad \bar{V}^\pi(d_0) \le \delta. \tag{7.2}$$

For completeness, we include a proof of this equivalence in Section B.1.2, which follows from the fact that the unsafe probability can be represented as the expected cumulative cost, i.e., $\mathbb{P}_{\rho^\pi(\tau)}(s_\triangleright \in \tau^h) = \mathbb{E}_{\rho^\pi(\tau)}[\sum_{t=0}^{h-1} c(s_t, a_t)]$. In other words, the chance-constrained policy optimization problem is a CMDP problem that aims to find a policy that has a high cumulative reward $V^\pi(d_0)$ with cumulative cost $\bar{V}^\pi(d_0)$ below the allowed failure probability δ.

Challenges This CMDP formulation has been commonly studied to design RL algorithms to find good polices that can be deployed safely (Chow, Ghavamzadeh, et al., 2017; Achiam et al., 2017; Tessler et al., 2018; Efroni et al., 2020). However, as mentioned in the introduction, these algorithms do not necessarily ensure safety during training and can be numerically unstable. At a high level, this instability stems from the lack of off-the-shelf computationally reliable and efficient solvers for large-scale constrained stochastic optimization.

While several control-theoretic techniques have been proposed to ensure safe data collection (Dalal et al., 2018; Wabersich and Zeilinger, 2021; Perkins and Barto, 2002; Chow, Nachum, Duenez-Guzman, et al., 2018; Chow, Nachum, Faust, et al., 2019; Berkenkamp et al., 2017; Fisac et al., 2018) and in some cases prevent the need for solving a constrained problem, it is unclear how the learned policy performs in terms of the objective $V^\pi(d_0)$ in Eq. (7.2) (i.e., without any interventions). Most of these algorithms also require stronger assumptions on the environment than approaches based on CMDPs (e.g., smoothness or ergodicity).

As we will show, our proposed approach retains the best of both approaches, ensuring safe data collection via interventions while guaranteeing good performance and safety when deployed without the intervention mechanism.

7.3 Method

Our safe RL approach, SAILR, finds an approximate solution to the CMDP problem in Eq. (7.2) by using an advantage-based intervention rule for safe data collection and an off-the-shelf RL algorithm for policy optimization. As we will see, SAILR can ensure safety for *both* training and deployment, when

1. the intervention rule belongs to an "admissible class" (see Definition 1 in Section 7.3.1); and

2. the base RL algorithm finds a nearly optimal policy for a new *unconstrained* problem of a surrogate MDP $\tilde{\mathcal{M}}$ constructed by the *intervention rule* together with $\mathcal{M}$.

Moreover, because SAILR can reuse existing RL algorithms for unconstrained MDPs to optimize policies, it is easier to implement and is more stable than typical CMDP approaches based on constrained optimization.

Specifically, SAILR optimizes policies iteratively as outlined in Algorithm 6. As input, it takes an RL algorithm $\mathcal{F}$ for unconstrained MDPs and an intervention rule $\mathcal{G} : \pi \mapsto \mathcal{G}(\pi)$, where $\pi' = \mathcal{G}(\pi)$ is a *shielded policy* such that π' runs a *backup policy* $\mu : \mathcal{S} \to \Delta(\mathcal{A})$ instead of π when π proposes "unsafe actions." In every iteration, it takes the policy π of interest and uses the intervention rule $\mathcal{G}$ to modify π into π' (3) such that running π' in the original MDP $\mathcal{M}$ can be safe with high probability while effectively simulating execution of π in the surrogate $\tilde{\mathcal{M}}$. Next, it collects the data $\mathcal{D}$ by running π' in $\mathcal{M}$ and then transforms it into new data $\tilde{\mathcal{D}}$ of π in $\tilde{\mathcal{M}}$ (4). It then feeds $\tilde{\mathcal{D}}$ to the base RL algorithm $\mathcal{F}$ for policy optimization (5), and optionally uses $\mathcal{D}$ to refine the intervention rule $\mathcal{G}$ (6). The process above is repeated until the training budget is used up. When this happens, SAILR terminates and returns the best policy $\hat{\pi}^\star$ the base algorithm $\mathcal{F}$ can produce for $\tilde{\mathcal{M}}$ so far (8).

We provide the following informal guarantee for SAILR, which is a corollary of our main result in Theorem 1 presented in Section 7.3.3.

Algorithm 6: SAILR

Require: MDP $\mathcal{M}$, RL algorithm $\mathcal{F}$, Intervention rule $\mathcal{G}$
Ensure: Optimized safe policy $\hat{\pi}^\star$
1: Initialize π arbitrarily
2: **while** training budget available **do**
3: Set $\pi' = \mathcal{G}(\pi)$
4: Run π' in $\mathcal{M}$ and collect datasets $\mathcal{D}$ and $\tilde{\mathcal{D}}$
5: Update π using algorithm $\mathcal{F}$ and dataset $\tilde{\mathcal{D}}$
6: Update intervention rule $\mathcal{G}$ using dataset $\mathcal{D}$ *(Optional)*
7: **end while**
8: Set $\hat{\pi}^\star = \pi$

Proposition 1 (Informal Guarantee)**.** *For* SAILR*, if the intervention rule $\mathcal{G}$ is admissible (Definition 1 in Section 7.3.1) and the RL algorithm $\mathcal{F}$ learns an ε-suboptimal policy $\hat{\pi}$ for $\tilde{\mathcal{M}}$, then, for any comparator policy $\pi^\star$, $\hat{\pi}$ has the following performance and safety guarantees in $\mathcal{M}$:*

$$V^{\pi^\star}(d_0) - V^{\hat{\pi}}(d_0) \leq \frac{2}{1-\gamma} P_{\mathcal{G}}(\pi^\star) + \varepsilon$$
$$\bar{V}^{\hat{\pi}}(d_0) \leq \bar{V}^{\mu}(d_0) + \varepsilon,$$

where μ is the backup policy in $\mathcal{G}$ and $P_{\mathcal{G}}(\pi^\star)$ is the probability that $\pi^\star$ visits the intervention set of $\mathcal{G}$ in $\mathcal{M}$.

In other words, if the base algorithm $\mathcal{F}$ used by SAILR can find an ε-suboptimal policy for the surrogate, unconstrained MDP $\tilde{\mathcal{M}}$, then the policy returned by SAILR is roughly ε-suboptimal in the original MDP $\mathcal{M}$, up to an additional error proportional to the probabilty that the comparator policy $\pi^\star$ would be overridden by the intervention rule $\mathcal{G}$ at some point while running in $\mathcal{M}$. Furthermore, the returned policy $\hat{\pi}$ is as safe as the backup policy μ of the intervention rule $\mathcal{G}$, up to an additional unsafe probability ε arising from the suboptimality in solving $\tilde{\mathcal{M}}$ with $\mathcal{F}$.

We point out the results above hold without any assumption on the MDP (other than that the unsafe subset $\mathcal{S}_{\text{unsafe}}$ is absorbing and the reward is zero on $\mathcal{S}_{\text{unsafe}}$). To learn a safe

policy, SAILR only needs a good *unconstrained* RL algorithm $\mathcal{F}$, a backup policy μ that is safe starting at the *initial state* (not globally), and an advantage function estimate of μ, as we explain later in this section.

The price we pay for keeping the agent safe using an intervention rule $\mathcal{G}$ is a performance bias proportional to $P_{\mathcal{G}}(\pi^\star)/(1-\gamma)$. This happens because employing an intervention rule during data collection limits where the agent can explore in $\mathcal{M}$. Thus, if the comparator policy $\pi^\star$ goes to high-reward states which would be cut off by the intervention rule, SAILR (and any other intervention-based algorithm) will suffer in proportion to the intervention probability. Despite the dependency on $P_{\mathcal{G}}(\pi^\star)$, we argue that SAILR provides a resonable trade-off for safe RL thanks to its training safety and numerical stability. Moreover, we will discuss how to use data to improve the intervention rule $\mathcal{G}$ to reduce this performance bias.

In the following, we first discuss the design of our advantage-based intervention rules (Section 7.3.1) and provide details of the new MDP $\tilde{\mathcal{M}}$ (Section 7.3.2). Then, we state and prove the main result Theorem 1 (Section 7.3.3). The omitted proofs for the results in this section can be found in Section B.1.

7.3.1 Advantage-Based Intervention

We propose a family of intervention rules based on *advantage functions*. Each intervention rule $\mathcal{G}$ here is specified by a 3-tuple $(\bar{Q}, \mu, \eta)$, where $\bar{Q} : \mathcal{S}_{\text{safe}} \times \mathcal{A} \to [0,1]$ is a state-action value estimator, $\mu \in \Pi$ is a backup policy, and $\eta \in [0,1]$ is a threshold. Given an arbitrary policy π, $\mathcal{G} = (\bar{Q}, \mu, \eta)$ constructs a new *shielded* policy π' based on an intervention set $\mathcal{I}$ defined by the advantage-like function $\bar{A}(s,a) \triangleq \bar{Q}(s,a) - \bar{Q}(s,\mu)$:

$$\mathcal{I} \triangleq \{(s,a) \in \mathcal{S}_{\text{safe}} \times \mathcal{A} \ : \ \bar{A}(s,a) > \eta\}. \tag{7.3}$$

When sampling a from $\pi'(\cdot|s)$ at some $s \in \mathcal{S}_{\text{safe}}$, we first sample a_- from $\pi(\cdot|s)$. If $(s, a_-) \notin \mathcal{I}$, we execute $a = a_-$. Otherwise, we sample a according to $\mu(\cdot|s)$. Mathematically, π' is described by the conditional distribution

$$\pi'(a|s) \triangleq \pi(a|s)\,\mathbf{1}\{(s,a) \notin \mathcal{I}\} + \mu(a|s)w(s), \tag{7.4}$$

where $w(s) \triangleq 1 - \sum_{\tilde{a}:(s,\tilde{a})\in\mathcal{I}} \pi(\tilde{a}|s)$. Note that π' may still take actions in $\mathcal{I}$ when μ has non-zero probability assigned to those actions.

By running the shielded policy contructed by the advantage function $\bar{A}$, SAILR controls the safety relative to the backup policy μ with respect to d_0. As we will show later, if the relative safety for each time step (i.e., advantage) is close to zero, then the relative safety overall is also close to zero (i.e. $\bar{V}^{\pi'}(d_0) \leq \delta$). Note that the sheilded policy π', while satisfying $\bar{V}^{\pi'}(d_0) \leq \delta$, can generally visit (with low probability) the states where $\bar{V}^{\mu}(s) > 0$ (e.g., states where $\bar{V}^{\mu}(s) = 1$). At these places where μ is useless for safety, we need an intervention rule that naturally deactivates and lets the learner explore. Our advantage-based rule does exactly that. On the contrary, designing an intervention rule directly based on Q-based functions (Bharadhwaj et al., 2021; Thananjeyan et al., 2021; Eysenbach et al., 2018; Srinivasan et al., 2020) can be overly conservative in this scenario.

Motivating Example

Let us use an example to explain why the advantage-based rule works. Suppose we have a baseline policy μ that is safe starting at the intial state of the MDP $\mathcal{M}$ (i.e., $\bar{V}^{\mu}(d_0)$ is small). We can use μ as the backup policy and construct an intervention rule $\mathcal{G} = (\bar{Q}^{\mu}, \mu, 0)$, where we recall $\bar{Q}^{\mu}$ denotes the state-action value of μ for the cost-based MDP $\bar{\mathcal{M}}$. Because the intervention set in Eq. (7.3) only allows actions that are no more unsafe than the backup policy μ during execution, intuitively we see that the intervened policy π' will be at least as safe as the baseline policy μ. Indeed, we can quickly verify this by the performance

difference lemma (Lemma 3): $\bar{V}^{\pi'}(d_0) = \bar{V}^{\mu}(d_0) + \frac{1}{1-\gamma}\mathbb{E}_{d^{\pi'}(s,a)}[\bar{A}^{\mu}(s,a)] \leq \bar{V}^{\mu}(d_0)$. Importantly, in this example, we see that the safety of π' is ensured without requiring $\bar{V}^{\mu}(s)$ to be small for any $s \in \mathcal{S}$, but only starting from states sampled from d_0.

General Rules

We now generalize the above motivating example to a class of *admissible* intervention rules.

Definition 1 (σ-Admissible Intervention Rule)**.** *We say that an intervention rule $\mathcal{G} = (\bar{Q}, \mu, \eta)$ is* σ-admissible *if, for some $\sigma \geq 0$, the following holds for all $s \in \mathcal{S}_{\text{safe}}$ and $a \in \mathcal{A}$:*

$$\bar{Q}(s,a) \in [0, \gamma] \tag{7.5}$$

$$\bar{Q}(s,a) + \sigma \geq c(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}[\bar{Q}(s',\mu)], \tag{7.6}$$

where we recall that $c(s,a) = \mathbf{1}\{s = s_{\triangleright}\}$. If the above holds with $\sigma = 0$, we say that $\mathcal{G}$ is admissible.

One can verify that in the previous example $\mathcal{G} = (\bar{Q}^{\mu}, \mu, 0)$ is admissible. But more generally, an admissible intervention rule with a backup policy μ can use $\bar{Q} \neq \bar{Q}^{\mu}$. In a sense, admissibility (with $\sigma = 0$) only needs $\bar{Q}$ to be a conservative version of $\bar{Q}^{\mu}$, because $\bar{Q}^{\mu}(s,a) = c(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}[\bar{Q}^{\mu}(s,\mu)]$ and Eq. (7.6) uses an upper bound; the σ term is a slack variable to allow for non-conservative $\bar{Q}$. More precisely, we have the following relationship.

Proposition 2. *If $\mathcal{G} = (\bar{Q}, \mu, \eta)$ is σ-admissible, then $\bar{Q}^{\mu}(s,a) \leq \bar{Q}(s,a) + \frac{\sigma}{1-\gamma}$ for all $s \in \mathcal{S}_{\text{safe}}$ and $a \in \mathcal{A}$.*

The condition in Eq. (7.6) is also closely related to the concept and theory of improvable heuristics considered by Cheng, Kolobov, and Swaminathan (2021) (i.e., we can view the

$\bar{Q}(s, \mu)$ as a heurisitic for safety), where the authors show such $\bar{Q}$ can be constructed by pessimistic offline RL methods.

Examples We discuss several ways to construct admissible intervention rules. From Definition 1, it is clear that if $\mathcal{G} = (\bar{Q}, \mu, \eta)$ is σ-admissible, then $\mathcal{G}$ is also σ'-admissible for any $\sigma' \geq \sigma$ (in particular, $(\bar{Q}, \mu, \eta)$ is γ-admissible if $\bar{Q}(s, a) \in [0, \gamma]$). So we only discuss the minimal σ.

Proposition 3 (Intervention Rules)**.** *The following are true.*

1. ***Baseline policy**: Given a baseline policy μ of $\mathcal{M}$, $\mathcal{G} = (\bar{Q}^\mu, \mu, \eta)$ or $\mathcal{G} = (\bar{Q}^\mu, \mu^+, \eta)$ is admissible, where μ^+ is the greedy policy that treats $\bar{Q}^\mu$ as a cost.*

2. ***Composite intervention**: Given K intervention rules $\{\mathcal{G}_k\}_{k=1}^K$, where each $\mathcal{G}_k = (\bar{Q}_k, \mu_k, \eta)$ is σ_k-admissible. Define $\bar{Q}_{\min}(s, a) = \min_k \bar{Q}_k(s, a)$ and let $\mu_{\min}$ be the greedy policy w.r.t. $\bar{Q}_{\min}$, and $\sigma_{\max} = \max_k \sigma_k$. Then, $\mathcal{G} = (\bar{Q}_{\min}, \mu_{\min}, \eta)$ is $\sigma_{\max}$-admissible.*

3. ***Value iteration**: Define $\bar{\mathcal{T}}$ as $\bar{\mathcal{T}}Q(s, a) \triangleq c(s, a) + \gamma \mathbb{E}_{p(s'|s,a)}[\min_{a'} Q(s', a')]$. If $\mathcal{G} = (\bar{Q}, \mu, \eta)$ is σ-admissible, then $\mathcal{G}^k = (\bar{\mathcal{T}}^k \bar{Q}, \mu^k, \eta)$ is $\gamma^k\sigma$-admissible, where μ^k is the greedy policy that treats $\bar{\mathcal{T}}^k \bar{Q}$ as a cost.*

4. ***Optimal intervention**: Let $\bar{\pi}^\star$ be an optimal policy for $\bar{\mathcal{M}}$, and let $\bar{Q}^\star$ be the corresponding state-action value function. Then $\mathcal{G}^\star = (\bar{Q}^\star, \bar{\pi}^\star, \eta)$ is admissible.*

5. ***Approximation**: For σ-admissible $\mathcal{G} = (\bar{Q}, \mu, \eta)$, consider $\hat{Q}$ such that $\hat{Q}(s, a) \in [0, \gamma]$ for all $s \in \mathcal{S}_{\text{safe}}$ and $a \in \mathcal{A}$. If $\|\hat{Q} - \bar{Q}\|_\infty \leq \delta$, then $\hat{\mathcal{G}} = (\hat{Q}, \mu, \eta)$ is $(\sigma + (1 + \gamma)\delta)$-admissible.*

Proposition 3 provides recipes for constructing σ-admissible intervention rules for safe RL, such as leveraging existing baseline policies in a system (Examples 1 and 2) and performing short-horizon planning (Example 3; namely model-predictive control (Bertsekas,

2017)). Moreover, Proposition 3 hints that we can treat designing intervention rules as finding the optimal state-action value function $\bar{Q}^\star$ in the cost-based MDP $\bar{\mathcal{M}}$ (Example 4). Later, in Section 7.3.3, we prove that this intuition is indeed correct: among all intervention rules that provide optimal safety, the rule $\mathcal{G}^\star = (\bar{Q}^\star, \bar{\pi}^\star, 0)$ provides the largest free space for data collection (i.e., small $P_{\mathcal{G}}(\pi^\star)$ in Proposition 1) among the safest intervention rules. Finally, Proposition 3 shows that an approximation of any σ-admissible intervention rule, such as one learned from data or inferred from an inaccurate model (Cheng, Kolobov, and Swaminathan, 2021), is also a reasonable intervention rule (Example 5). As learning continues in SAILR, we can use the newly collected data from $\mathcal{M}$ to refine our estimate of the ideal $\bar{Q}$, such as by performing additional policy evaluation for μ or policy optimization to find $\bar{Q}^\star$ of the cost-based MDP $\bar{\mathcal{M}}$.

General Backup Policies To conclude this section, we briefly discuss how to extend the above results to work with general backup policies that may take actions outside $\mathcal{A}$ (i.e., the actions the learner policy aims to use), similar to Turchetta et al. (2020). For example, such a backup policy can be implemented through an external "kill switch" in a robotics system. For SAILR's theoretical guarantees to hold in this case, we require one extra assumption: for all $(s, a) \in \mathcal{I}$ that can be reached from d_0 with some policy, there must be some $a' \in \mathcal{A}$ such that $\bar{A}(s, a') = \bar{Q}(s, a') - \bar{Q}(s, \mu) \leq \eta$. In other words, for every state-action we can reach from d_0 that will be overridden, there must an alternative action *in the agent's action space* $\mathcal{A}$ that keeps the agent's policy from being intervened. This condition is a generalization of Definition 2 introduced later for our analysis (a condition we call *partial*), which is essential to the unconstrained policy optimization reduction in SAILR (Section 7.3.3). Note that while this condition holds trivially when the backup policy μ takes only actions in $\mathcal{A}$, generally the validity of this condition depends on the details of μ and transition dynamics p.

7.3.2 Absorbing MDP

SAILR performs policy optimization by running a base RL algorithm $\mathcal{F}$ to solve a new unconstrained MDP $\tilde{\mathcal{M}}$. In this section, we define $\tilde{\mathcal{M}}$ and discuss how to simulate experiences of π in $\tilde{\mathcal{M}}$ by running the shielded policy $\pi' = \mathcal{G}(\pi)$ in the original MDP $\mathcal{M}$.

Given the MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$ and the intervention set $\mathcal{I}$ in Eq. (7.3), we define $\tilde{\mathcal{M}} = (\tilde{\mathcal{S}}, \mathcal{A}, \tilde{p}, \tilde{r}, \gamma)$ as follows: Let $s_\dagger$ denote an absorbing state and $c_\dagger \geq 0$ be some problem-independent constant. The new MDP $\tilde{\mathcal{M}}$ has the state space $\tilde{\mathcal{S}} = \mathcal{S} \cup \{s_\dagger\}$ and modified dynamics and reward,

$$\tilde{r}(s,a) = \begin{cases} -c_\dagger, & (s,a) \in \mathcal{I} \\ 0, & s = s_\dagger \\ r(s,a), & \text{otherwise} \end{cases} \tag{7.7}$$

$$\tilde{p}(s'|s,a) = \begin{cases} \mathbf{1}\{s' = s_\dagger\}, & (s,a) \in \mathcal{I} \text{ or } s = s_\dagger \\ p(s'|s,a), & \text{otherwise.} \end{cases} \tag{7.8}$$

Since $s_\dagger$ is absorbing, given a policy π defined on $\mathcal{M}$, without loss of generality we extend its definition on $\tilde{\mathcal{M}}$ by setting $\pi(a|s_\dagger)$ to be the uniform distribution over $\mathcal{A}$. A simple example of this construction is shown in Fig. 7.2.

Compared with the original $\mathcal{M}$, the new MDP $\tilde{\mathcal{M}}$ has more absorbing state-action pairs and assigns lower rewards to them. When the agent takes some $(s,a) \in \mathcal{I}$ in $\tilde{\mathcal{M}}$, it goes to an absorbing state $s_\dagger$ and receives a *non-positive* reward. Thus, the new MDP $\tilde{\mathcal{M}}$ gives larger penalties for taking intervened state-actions than for going into $\mathcal{S}_{\text{unsafe}}$, where we only receive zero reward. This design ensures that any nearly-optimal policy of $\tilde{\mathcal{M}}$ will (when run in $\mathcal{M}$) have high reward and low probability of visiting intervened state-actions. As we will see, as long as $\mathcal{G}$ provides *safe* shielded polices, solving $\tilde{\mathcal{M}}$ will lead to a safe policy with potentially good performance in the original MDP $\mathcal{M}$ even after we lift the

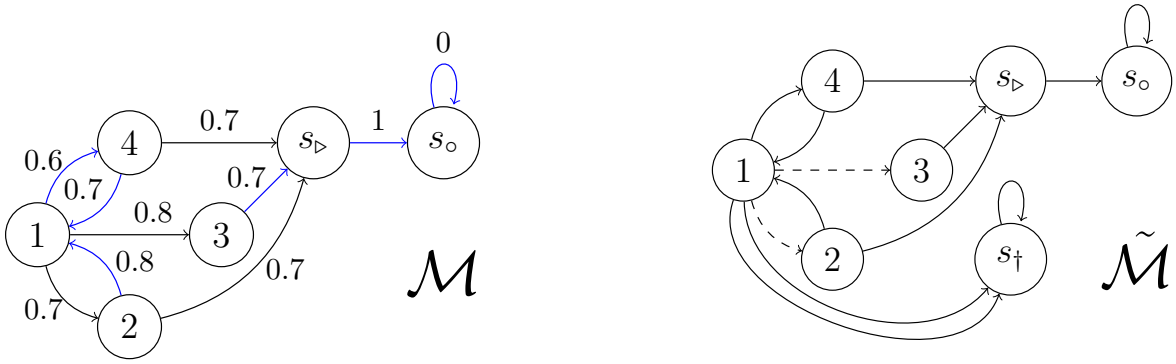


Figure 7.2: A simple example of the construction of $\tilde{\mathcal{M}}$ from $\mathcal{M}$ using advantage-based intervention given by some $\mathcal{G} = (\bar{Q}, \mu, \eta)$. In $\mathcal{M}$, the transitions are deterministic, and the blue arrows correspond to actions given by μ. The edge weights correspond to $\bar{Q}$, and $\mathcal{G}$ can be verified to be σ-admissible when $\sigma = 0.25$ and $\gamma = 0.9$. The surrogate MDP $\tilde{\mathcal{M}}$ is formed upon intervention with $\eta = 0.05$. The transitions $1 \to 2$ and $1 \to 3$ are replaced with transitions to the absorbing state $s_\dagger$.

intervention.

To simulate experiences of a policy π in $\tilde{\mathcal{M}}$, we simply run $\pi' = \mathcal{G}(\pi)$ in the original MDP $\mathcal{M}$ and collect samples until the intervention triggers (if at all). Specifically, suppose running π' in $\mathcal{M}$ generates a trajectory $\tau = (s_0, a_0, \ldots, s_T, a'_T, \ldots)$, where T is the time step of intervention and a'_T is the first action given by the backup policy μ. Let a_T be the corresponding action from π that was overridden. We construct the trajectory $\tilde{\tau}$ that would be generated by running π in $\tilde{\mathcal{M}}$ by setting $\tilde{\tau} = (s_0, a_0, \ldots, s_T, a_T, \tilde{s}_{T+1}, \tilde{a}_{T+1}, \ldots)$, where $\tilde{s}_{t'} = s_\dagger$ and $\tilde{a}_{t'}$ is arbitrary for any $t' \geq t + 1$. This is valid since the two MDPs $\mathcal{M}$ and $\tilde{\mathcal{M}}$ share the same dynamics until the intervention happens at time step T.

7.3.3 Theoretical Analysis

We state the main theoretical result of SAILR, which includes the informal Proposition 1 as a special case.

Theorem 1 (Performance and Safety Guarantee at Deployment)**.** *Let $c_\dagger = 1$ and $\mathcal{G}$ be σ-admissible. If $\hat{\pi}$ is an ε-suboptimal policy for $\tilde{\mathcal{M}}$, then, for any comparator policy $\pi^\star$, the*

following performance and safety guarantees hold for $\hat{\pi}$ in $\mathcal{M}$:

$$V^{\pi^\star}(d_0) - V^{\hat{\pi}}(d_0) \leq \frac{2}{1-\gamma} P_{\mathcal{G}}(\pi^\star) + \varepsilon$$
$$\bar{V}^{\hat{\pi}}(d_0) \leq \bar{Q}(d_0, \mu) + \frac{\min\{\sigma + \eta, 2\gamma\}}{1-\gamma} + \varepsilon,$$

where $P_{\mathcal{G}}(\pi^\star) \triangleq (1-\gamma)\sum_{h=0}^{\infty} \gamma^h \, \mathbb{P}_{\rho^{\pi^\star}(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing)$ is the probability that $\pi^\star$ visits $\mathcal{I}$ in $\mathcal{M}$.

Theorem 1 shows that, when the base RL algorithm $\mathcal{F}$ finds an ε-suboptimal policy $\hat{\pi}$ in $\tilde{\mathcal{M}}$, this policy $\hat{\pi}$ is also close to ε-suboptimal in the CMDP in Eq. (7.2), as long as running the comparator policy $\pi^\star$ in $\mathcal{M}$ will result in low probability of visiting state-actions that would be intervened by $\mathcal{G}$ (i.e., $P_{\mathcal{G}}(\pi^\star)$ is small). In addition, the policy $\hat{\pi}$ is almost as safe as the backup policy μ, since $\bar{Q}(d_0, \mu)$ can be viewed as an upper bound of $\bar{Q}^{\mu}(d_0, \mu)$. The safety deterioration can be made small when the suboptimality ε, intervention threshold η, and imperfect admissibility σ of $\mathcal{G}$ are small. The proof of Theorem 1 follows directly from Theorem 2 and Proposition 7 below, which are main properties of the advantage-based intervention rules and the absorbing MDPs in SAILR. We now discuss these properties in more detail.

Intervention Rules

First, we show that the shielded policy π' produced by a σ-admissible intervention rule $\mathcal{G} = (\bar{Q}, \mu, \eta)$ has a small unsafe cost if backup policy μ has a small cost.

Theorem 2 (Safety of Shielded Policy)**.** *Let $\mathcal{G} = (\bar{Q}, \mu, \eta)$ be σ-admissible as per Definition 1. For any policy π, let $\pi' = \mathcal{G}(\pi)$. Then,*

$$\bar{V}^{\pi'}(d_0) \leq \bar{Q}(d_0, \mu) + \frac{\min\{\sigma + \eta, 2\gamma\}}{1-\gamma}. \tag{7.9}$$

Next we provide a formal statement that $\mathcal{G}^\star = (\bar{Q}^\star, \bar{\pi}^\star, 0)$ is the optimal intervention

rule that gives the largest free space for policy optimization, among the safest intervention rules. The size of the free space provided $\mathcal{G}^\star$ is captured as $\mathrm{Supp}_{\mathcal{S}\times\mathcal{A}}(\tilde{d}^{\star,\pi})$, which can be interpreted as the state-actions that $\mathcal{G}^\star(\pi)$ can explore before any intervention is triggered.

Proposition 4. *Let $\bar{\pi}^\star$ be an optimal policy for $\bar{\mathcal{M}}$, $\bar{Q}^\star$ be its state-action value function, and $\bar{V}^\star$ be its state value function. Let $G_0 = \{(\bar{Q}, \mu, 0) : (\bar{Q}, \mu, 0)$ is admissible, $\bar{Q}(d_0, \mu) = \bar{V}^\star(d_0)\}$. Let $\mathcal{G}^\star = (\bar{Q}^\star, \bar{\pi}^\star, 0) \in G_0$. Consider arbitrary $\mathcal{G} \in G_0$ and policy π. Let $\tilde{\mathcal{M}}$ and $\tilde{\mathcal{M}}^\star$ be the absorbing MDPs induced by $\mathcal{G}$ and $\mathcal{G}^\star$, respectively, and let $\tilde{d}^\pi$ and $\tilde{d}^{\star,\pi}$ be their respective state-action distributions. Then,*

$$\mathrm{Supp}_{\mathcal{S}\times\mathcal{A}}(\tilde{d}^\pi) \subseteq \mathrm{Supp}_{\mathcal{S}\times\mathcal{A}}(\tilde{d}^{\star,\pi}),$$

where $\mathrm{Supp}_{\mathcal{S}\times\mathcal{A}}(d)$ denotes the support of a distribution d when restricted on $\mathcal{S}\times\mathcal{A}$.

Finally, we highlight a property of the intervention set $\mathcal{I}$ of our advantage-based rules, which is crucial for the unconstrained MDP reduction described in the next section.

Definition 2. *A set $\mathcal{X} \subset \mathcal{S}_{\text{safe}} \times \mathcal{A}$ is called* partial *if for every $(s, a) \in \mathcal{X}$, there is some $a' \in \mathcal{A}$ such that $(s, a') \notin \mathcal{X}$.*

Proposition 5. *If $\eta \geq 0$, then $\mathcal{I}$ in Eq.* (7.3) *is partial.*

Proof. For $(s, a) \in \mathcal{I}$, define $a' = \arg\min_{a''\in\mathcal{A}} \bar{Q}(s, a'')$. Because $\bar{A}(s, a') = \bar{Q}(s, a') - \bar{Q}(s, \mu) \leq 0 \leq \eta$, we conclude that $(s, a') \notin \mathcal{I}$. □

Abosrbing MDP

As discussed in Section 7.3.2, the new MDP $\tilde{\mathcal{M}}$ provides a pessimistic value estimate of $\mathcal{M}$ by penalizing trajectories that trigger the intervention rule $\mathcal{G}$. Precisely, we can show that the amount of pessimism introduced on a policy π is proportional to $P_{\mathcal{G}}(\pi)$ (the probability of triggering the intervention rule $\mathcal{G}$ when running π in $\mathcal{M}$).

Lemma 1. *For every policy π, it holds that*

$$c_\dagger P_{\mathcal{G}}(\pi) \leq V^\pi(d_0) - \tilde{V}^\pi(d_0) \leq \left(c_\dagger + \frac{1}{1-\gamma}\right) P_{\mathcal{G}}(\pi).$$

As a result, one would intuitively imagine that an optimal policy of $\tilde{\mathcal{M}}$ would never visit the intervention set $\mathcal{I}$ at all. Below we show that this intuition is correct. Importantly, we highlight that this property holds *only* because the intervention set $\mathcal{I}$ used here is *partial* (Proposition 5). If we were to construct an absorbing MDP $\tilde{\mathcal{M}}'$ described in Section 7.3.2 using an arbitrary non-partial subset $\mathcal{I}' \subseteq \mathcal{S}_{\text{safe}} \times \mathcal{A}$, then the optimal policy of $\tilde{\mathcal{M}}'$ can still enter $\mathcal{I}'$ for any $c_\dagger \geq 0$, because an optimal policy of $\tilde{\mathcal{M}}'$ can use earlier rewards to mitigate penalties incurred in $\mathcal{I}'$ (Section B.2.1).

Proposition 6. *If $c_\dagger$ is positive and $\mathcal{G}$ induces a partial $\mathcal{I}$, then every optimal policy $\tilde{\pi}^\star$ of $\tilde{\mathcal{M}}$ satisfies $P_{\mathcal{G}}(\tilde{\pi}^\star) = 0$.*

The partial property of $\mathcal{I}$ enables our unconstrained MDP reduction, which relates the performance and safety of a policy π in the original MDP $\mathcal{M}$ to the suboptimality in the new MDP $\tilde{\mathcal{M}}$ and the safety of $\pi' = \mathcal{G}(\pi)$.

Proposition 7 (Suboptimality in $\tilde{\mathcal{M}}$ to Suboptimality and Safety in $\mathcal{M}$)**.** *Let $c_\dagger$ be positive. For some policy π, let π' be the shielded policy defined in Eq.* (7.4)*. Suppose π is ε-suboptimal for $\tilde{\mathcal{M}}$. Then, for any comparator policy $\pi^\star$, the following performance and safety guarantees hold for π in $\mathcal{M}$:*

$$V^{\pi^\star}(d_0) - V^\pi(d_0) \leq \left(c_\dagger + \frac{1}{1-\gamma}\right) P_{\mathcal{G}}(\pi^\star) + \varepsilon$$
$$\bar{V}^\pi(d_0) \leq \bar{V}^{\pi'}(d_0) + \frac{\varepsilon}{c_\dagger}.$$

7.4 Related Work

CMDPs (Altman, 1999) have been a popular framework for safe RL as it side-steps the reward design problem for ensuring safety in a standard MDP (Geibel and Wysotzki, 2005; Shalev-Shwartz et al., 2016). Most existing CMDP-based safe RL algorithms closely follow algorithms in the constrained optimization literature (Bertsekas, 2014). They can be classified into either online or offline schemes. Online schemes learn by coupling the iteration of a numerical optimization algorithm (notably primal-dual gradient updates) with data collection (Borkar, 2005; Chow, Ghavamzadeh, et al., 2017; Tessler et al., 2018; Bohez, Abdolmaleki, et al., 2019), and these algorithms have also been studied in the exploration context (Ding et al., 2021; Qiu et al., 2020; Efroni et al., 2020). However, they have no guarantees on policy safety during training. Offline schemes (Achiam et al., 2017; Bharadhwaj et al., 2021; Le et al., 2019; Efroni et al., 2020), on the other hand, separate optimization and data collection. They conservatively enforce safety constraints on every policy iterate but are more difficult to scale up. Many of these constrained algorithms for CMDPs, however, have worse numerical stability compared with typical RL algorithms for MDPs, because of the nonconvex saddle-point of the CMDP (J. D. Lee et al., 2019; Chow, Nachum, Duenez-Guzman, et al., 2018).

Another line of safe RL research uses control-theoretic techniques to enforce safe exploration, though only few provide guarantees with respect to the CMDP in Eq. (7.2). These methods include restricting the agent to take actions that lead to next-state safety (Dalal et al., 2018; Wabersich and Zeilinger, 2021) or states where a safe backup exists (Hans et al., 2008; Polo and Rebollo, 2011; S. Li and Bastani, 2020). Other works consider more structured shielding approaches, including those with temporal logic safety rules and backup policies (Alshiekh et al., 2018) and neurosymbolic policies (Anderson et al., 2020) whose safety can be checked easily. Many of these approaches require strong assumptions on the MDP (e.g., taking an action to ensure the next state's safety being sufficient to imply

all future states will continue to have such safe actions available). Algorithms based on Lyapunov functions and reachability (Perkins and Barto, 2002; Chow, Nachum, Duenez-Guzman, et al., 2018; Chow, Nachum, Faust, et al., 2019; Berkenkamp et al., 2017; Fisac et al., 2018) address the long-term feasibility issue, but they are more complicated than common RL algorithms. We note that our admissible intervention rules in Eq. (7.6) can be viewed as a state-action Lyapunov function.

To the best of our knowledge, SAILR is the first unconstrained method that provides formal guarantees with respect to the CMDP objective. The closest work to ours is that of Turchetta et al. (2020), who also use the idea of intervention for training safety and trains the agent in a new MDP that discourages visiting intervened state-actions. However, their algorithm, CISR, is still based on calling CMDP subroutines (Le et al., 2019). They neither specify how the intervention rules can be constructed nor provide performance guarantees. By comparsion, we provide a general recipe of intervention rules and obtain the properties desired by Turchetta et al. (2020) by simply unconstrained RL.

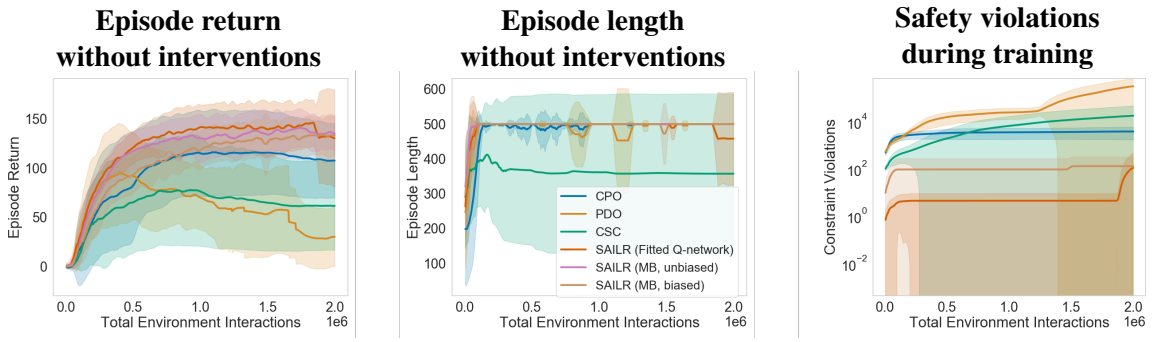
7.5 Experiments

We conducted experiments to corroborate our theoretical analysis of SAILR. We aimed to verify whether a properly designed intervention mechanism can drastically reduce the amount of unsafe trajectories generated in training while still resulting in good safety and performance in deployment.

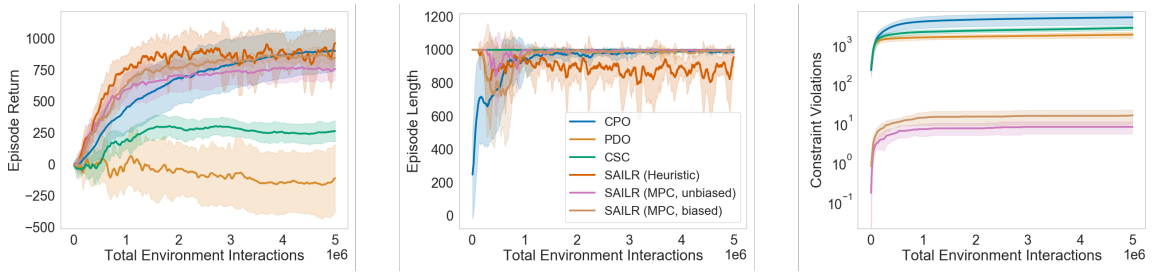
Our experiments consider two different tasks:

1. A simple point robot inspired from Achiam et al. (2017) that gets reward for following a circular path at high speed, but is constrained to stay in a region smaller than the target circle.

2. A half-cheetah that gets reward equal to its forward velocity, with one of its links constrained to remain in a given height range, outside of which the robot is deemed



(a) Results for point robot



(b) Results for half-cheetah

Figure 7.3: Results of SAILR and baseline CMDP-based methods. Overall, SAILR dramatically reduces the amount of safety constraint violations while still having large returns at deployment. Plots in a row share the same legend. All error bars are ± 1 standard deviation over 10 (point robot) or 8 (half-cheetah) random seeds. Any curve not plotted in the third column corresponds to zero safety violations.

to be unsafe.

In all experiments, when computing $\bar{Q}$, we opted to use a *shaped* cost function in place of the original sparse indicator cost function to make our intervention mechanism more conservative (and hence the training process safer). In particular, this shaped cost function is a function of the distance to the unsafe set and is an upper bound of the original sparse cost. Section B.4 in the Appendix includes some additional experiments where the original sparse cost is used.

We implemented SAILR by using PPO (Section 6.1) as the RL subroutine. We also compare our approach to two CMDP-based approaches: constrained policy optimization (CPO) (Achiam et al., 2017) and a primal-dual optimization (PDO) algorithm (Chow, Ghavamzadeh, et al., 2017). For the PDO algorithm, we used PPO as the policy optimization subroutine and dual gradient ascent as the Lagrange multiplier update. We also consider a variant of PDO,

called "conservative safety critic" (CSC), where a learned conservative critic is used to filter unsafe actions (Bharadhwaj et al., 2021).

7.5.1 Point Robot

Here SAILR used the intervention rule $\mathcal{G} = (\mu, \bar{Q}, \eta)$ where the baseline policy μ aims to stop the robot by deceleration. The function $\bar{Q}$ was estimated by either querying a fitted Q-network or by rolling out μ on a dynamical model (denoted "MB" in Fig. 7.3a) of the point robot and querying a shaped cost function. We consider both biased and unbiased models (details in Section B.3.1). Fig. 7.3a shows the main experimental results, with all three instances of SAILR outperforming the baselines on all three metrics. For SAILR, the shielding prevented many safety violations, and the unconstrained approach allowed for reliable convergence as opposed to the baselines which rely on elaborate constrained approaches.

7.5.2 Half-Cheetah

We considered two intervention rules in SAILR: a reset backup policy μ with a simple heuristic $\bar{Q}$ based on the predicted height of the link after taking a proposed action, and a reset backup policy μ based on a sampling-based model predictive control (MPC) algorithm (Williams, Wagener, et al., 2017; Bhardwaj, Choudhury, et al., 2021) with a model-based value estimate (i.e., $\bar{Q} \approx \bar{Q}^{\mu}$). The simple heuristic uses a slightly smaller height range for intervention to attempt to construct a *partial* intervention set (Section 7.3.3). The MPC algorithm optimized a control sequence over the same cost function. The function $\bar{Q}$ was computed by rolling out this control sequence on the dynamical model and querying the cost function. We also considered model bias in the MPC experiments (details in Section B.3.2).

As with the point environment, SAILR incurred orders of magnitude fewer safety violations than the baselines (right plot of Fig. 7.3b), with all three instances having compa-

rable deployment performance to that of CPO. Though the heuristic intervention violated no constraints in training, it was consistently unsafe in deployment (middle plot), likely because the resulting intervention set is not partial. On the other hand, MPC-based approaches were consistently safe in deployment, owing to its multi-step lookahead yielding an intervention rule that is likely to be σ-admissible (and therefore give an intervention set that is partial).

7.6 Conclusion

We presented an intervention-based method for safe reinforcement learning. By utilizing advantage functions for intervention and penalizing an agent for taking intervened actions, we can use unconstrained RL algorithms in the safe learning domain. Our analysis shows that using advantage functions for the intervention decision gives strong guarantees for safety during training and deployment, with the performance only limited by how often the true optimal policy would be intervened. We also discussed ways of synthesizing good intervention rules, such as using value iteration techniques. Finally, our experiments showed that the shielded policy violates few, if any, constraints during training while the corresponding deployed policy enjoys convergence to a large return.

CHAPTER 8

MoCapAct: A MULTI-TASK DATASET FOR SIMULATED HUMANOID CONTROL

Simulated humanoids are an appealing research domain due to their physical capabilities offering a wide range of flexibility in tasks and behaviors they can achieve. Nonetheless, they are also challenging to control, as a policy must drive an unstable, discontinuous, and high-dimensional physical system. One widely studied approach is to utilize motion capture (MoCap) data to teach the humanoid agent low-level skills (e.g., standing, walking, and running) which can then be re-used to synthesize high-level behaviors. However, even with MoCap data, controlling simulated humanoids remains very hard, as MoCap data offers only kinematic information. Finding physical control inputs to realize the demonstrated motions requires computationally intensive methods like reinforcement learning. Thus, despite the publicly available MoCap data, its utility has been limited to institutions with large-scale compute. In this chapter, we dramatically lower the barrier for productive research on this topic by training and releasing high-quality agents that can track over three hours of MoCap data for a simulated humanoid in the `dm_control` physics-based environment. We release *MoCapAct*, a dataset of these expert agents and their rollouts, which contain proprioceptive observations and actions. We demonstrate the utility of MoCapAct by using it to train a *single* hierarchical policy capable of tracking the *entire* MoCap dataset within `dm_control` and show the learned low-level component can be re-used to efficiently learn downstream high-level tasks. Finally, we use MoCapAct to train an autoregressive GPT model and show that it can control a simulated humanoid to perform natural motion completion given a motion prompt. Videos of the results and links to the code and dataset are available at the project website (`microsoft.github.io/MoCapAct`).

8.1 Introduction

The wide range of human physical capabilities makes simulated humanoids a compelling platform for studying motor intelligence. Learning and utilization of motor skills is a prominent research topic in machine learning, with advances ranging from emergence of learned locomotion skills in traversing an obstacle course (Heess et al., 2017) to the picking up and carrying of objects to desired locations (Merel, Tunyasuvunakool, et al., 2020; Peng, Chang, et al., 2019) to team coordination in simulated soccer (Liu et al., 2022). Producing natural and physically plausible human motion animation (Harvey et al., 2020; Kania et al., 2021; Yuan and Kitani, 2020) is an active research topic in the game and movie industries. However, while physical simulation of human capabilities is a useful research domain, it is also very challenging from a control perspective. A controller must contend with an unstable, discontinuous, and high-dimensional system that requires a high degree of coordination to execute a desired motion.

Tabula rasa learning of complex humanoid behaviors (e.g., navigating through an obstacle field) is extremely difficult for all known learning approaches. In light of this challenge, motion capture (MoCap) data has become an increasingly common aid in humanoid control research (Merel, Tassa, et al., 2017; Peng, Abbeel, et al., 2018). MoCap trajectories contain *kinematic* information about motion: they are sequences of configurations and poses that the human body assumes throughout the motion in question. This data can alleviate the difficulty of training sophisticated control policies by enabling a simulated humanoid to learn *low-level* motor skills from MoCap demonstrations. The low-level skills can then be re-used for learning advanced, higher-level motions. Datasets such as CMU MoCap (CMU, 2003), Human3.6M (Ionescu et al., 2013), and LaFAN1 (Harvey et al., 2020) offer hours of recorded human motion, ranging from simple locomotion demonstrations to interactions with other humans and objects.

However, since MoCap data only offers kinematic information, utilizing it in a physics

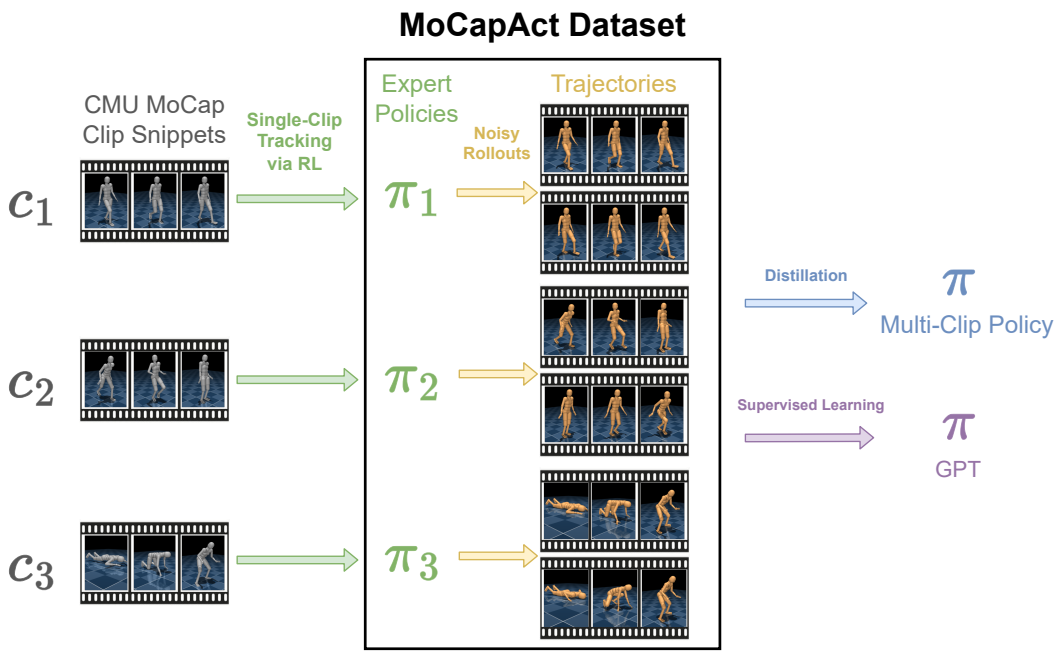


Figure 8.1: The MoCapAct dataset includes expert policies that are trained to track individual clips. A dataset of noise-injected rollouts (containing observations and actions) is then collected from each expert. These rollouts can subsequently be used to, for instance, train a multi-clip or GPT policy.

simulator requires recovering the actions (e.g., joint torques) that induce the sequence of kinematic poses in a given MoCap trajectory (i.e., track the trajectory). While easier than *tabula rasa* learning of a high-level task, finding an action sequence that makes a humanoid track a MoCap sequence is still non-trivial. For instance, this problem has been tackled with reinforcement learning (Chentanez et al., 2018; Merel, Hasenclever, et al., 2019; Peng, Abbeel, et al., 2018) and adversarial learning (Merel, Tassa, et al., 2017; Z. Wang et al., 2017). The computational burden of finding these actions scales with the amount of MoCap data, and training agents to recreate hours of MoCap data requires significant compute. *As a result, despite the broad availability of MoCap datasets, their utility—and their potential for enabling research progress on learning-based humanoid control—has been limited to institutions with large compute budgets.*

To remove this obstacle and facilitate the use of MoCap data in humanoid control research, we introduce **MoCapAct** (Motion Capture with Actions, Fig. 8.1), a dataset of

high-quality MoCap-tracking policies for a MuJoCo-based (Todorov et al., 2012) simulated humanoid as well as a collection of rollouts from these expert policies. The policies from MoCapAct can track 3.5 hours of recorded motion from CMU MoCap (CMU, 2003), one of the largest publicly available MoCap datasets. We analyze the expert policies of MoCapAct and, to illustrate MoCapAct's usefulness for learning diverse motions, use the expert rollouts to train a *single* hierarchical policy which is capable of tracking *all* of the considered MoCap clips. We then re-use the low-level component of the policy to efficiently learn downstream tasks via reinforcement learning. Finally, we use the dataset for generative motion completion by training a GPT network (Karpathy, 2020) to produce a motion in the MuJoCo simulator given a motion prompt.

8.2 Related Work

MoCap Data Of the existing datasets featuring motion capture of humans, the largest and most cited are CMU MoCap (CMU, 2003) and Human3.6M (Ionescu et al., 2013). These datasets feature tens of hours of human motion capture arranged as a collection of clips recorded at 30-120Hz. They demonstrate a wide range of motions, including locomotion (e.g., walking, running, jumping, and turning), physical activities (e.g., dancing, boxing, and gymnastics), and interactions with other humans and objects.

MoCap Tracking via Reinforcement Learning To make use of MoCap data for downstream tasks, much of prior work first learns individual clip-tracking policies. For instance, Peng, Abbeel, et al. (2018), Merel, Ahuja, et al. (2019), Merel, Hasenclever, et al. (2019), and Merel, Tunyasuvunakool, et al. (2020) use reinforcement learning (RL) to learn the clip-tracking policies, whereas Merel, Tassa, et al. (2017) use adversarial imitation learning. Upon learning the tracking policies, there are a variety of ways to utilize them. Peng, Abbeel, et al. (2018) and Merel, Tassa, et al. (2017) and Merel, Ahuja, et al. (2019) learn a skill-selecting policy to dynamically choose a clip-tracking policy to achieve new tasks.

Merel, Hasenclever, et al. (2019) and Merel, Tunyasuvunakool, et al. (2020) instead opt for a distillation approach, whereby they collect rollouts from the clip-tracking policies and then train a hierarchical multi-clip policy via supervised learning on the rollouts. The low-level policy is then re-used to aid in learning new high-level tasks.

Alternatively, large-scale RL may be used to learn a single policy that covers the MoCap dataset. Hasenclever et al. (2020) use a distributed RL setup for the MuJoCo simulator (Todorov et al., 2012), while Peng, Guo, et al. (2022) use the GPU-based Isaac simulator (Makoviychuk et al., 2021) to perform RL on a single machine.

While some prior work has released source code to train individual clip-tracking policies (Peng, Abbeel, et al., 2018; Yuan and Kitani, 2020), their included catalog of policies is small, and the resources needed to train per-clip policies scale linearly with the number of MoCap clips. In the process of our work, we found that we needed about *50 years* of wall-clock time to train the policies to track our MoCap corpus using a similar approach to Peng, Abbeel, et al. (2018).

Motion Completion Outside of the constraints of a physics simulator, learning natural completions of MoCap trajectories (i.e., producing a trajectory given a prompt trajectory) is the subject of many research papers (Mourot et al., 2022), typically motivated by the challenging and labor-intensive process of creating realistic animations for video games and films. Prior work (Aksan et al., 2021; Harvey et al., 2020; Kania et al., 2021; Mao et al., 2019; Tevet et al., 2022; B. Wang et al., 2019) typically trains a model to replicate the kinematic motion found in a MoCap dataset, which is then evaluated according to how well the model can predict or synthesize motions given some initial prompt on held-out trajectories.

The more difficult task of performing motion completion *within a physics simulator* is not widely studied. Yuan and Kitani (2020) jointly learn a kinematic policy and a tracking policy, where the kinematic policy predicts future kinematic poses given a recent history

Figure 8.2: The humanoid displaying a variety of motions from the CMU MoCap dataset.

of observations and the tracking policy outputs a low-level action to track the predicted poses.

8.3 The `dm_control` Humanoid Environment

Our simulated humanoid of interest is the "CMU Humanoid" (Fig. 8.2) from the `dm_control` package (Tunyasuvunakool et al., 2020), which contains 56 joints and is designed to be similar to an average human body. The humanoid contains a rich and customizable observation space, from proprioceptive observations like joint positions and velocities, actuator states, and touch sensor measurements to high-dimensional observations like images from an egocentric camera. The action a is the desired joint angles of the humanoid, which are then converted to joint torques via some pre-defined PD controllers. The humanoid operates in the MuJoCo simulator (Todorov et al., 2012).

The `dm_control` package contains a variety of tools for the humanoid. The package comes with pre-defined tasks like navigation through an obstacle field (Heess et al., 2017), maze navigation (Merel, Ahuja, et al., 2019), and soccer (Liu et al., 2022), and a user may create custom tasks with the package's API. The `dm_control` package also integrates 3.5 hours of motion sequences from the CMU Motion Capture Dataset (CMU, 2003), including clips of locomotion (standing, walking, turning, running, jumping, etc.), acrobatics, and arm movements. Each clip C is a reference state sequence $(\hat{s}_0^C, \hat{s}_1^C, \ldots, \hat{s}_{T_C-1}^C)$, where T_C is the clip length and each $\hat{s}_t^C$ contains kinematic information like joint angles, joint velocities, and humanoid pose.

As discussed in Section 8.2, training a control policy to work on all of the included clips requires large-scale solutions. For example, Hasenclever et al. (2020) rely on a distributed RL approach that uses about ten billion environment interactions collected by 4000 parallel actor processes running for multiple days. To our knowledge, there are no agents publicly available that can track all the MoCap data within `dm_control`. We address this gap by releasing a dataset of high-quality experts and their rollouts for the "CMU Humanoid" in the `dm_control` package.

8.4 MoCapAct Dataset

The MoCapAct dataset (Fig. 8.1) consists of:

- Experts, each trained to track an individual snippet from the MoCap dataset (Section 8.4.1) and
- HDF5 files containing rollouts from the aforementioned experts (Section 8.4.2).

We include documentation of the MoCapAct dataset in Section C.1.

8.4.1 Clip Snippet Experts

Our expert training scheme largely followed that of Merel, Ahuja, et al. (2019), Merel, Hasenclever, et al. (2019), and Peng, Abbeel, et al. (2018), which we now summarize.

Training We split each clip in the MoCap dataset into 4–6 second snippets with 1-second overlaps. With 836 clips in the MoCap dataset, this clip splitting resulted in 2589 snippets. For each clip snippet c, we trained a time-indexed Gaussian policy $\pi_c(a|s,t)$ to track the snippet. We used the same clip-tracking reward function $r_c(s,t)$ as Hasenclever et al. (2020), which encourages matching the MoCap clip's joint angles and velocities, positions of various body parts, and joint orientations. This reward function lies in the interval $[0, 1.45]$. To speed up training, we used the same early episode termination condition

Table 8.1: Snippet expert results on the MoCap snippets within `dm_control`. We disable the Gaussian noise for π_c when computing these results.

	Mean	Standard deviation	Median	Minimum	Maximum
Avg. normalized episode reward	0.816	0.153	0.777	0.217	1.233
Avg. normalized episode length	0.997	0.022	1.000	0.424	1.000

as Hasenclever et al. (2020), which activates if the humanoid deviates too far from the snippet. To help exploration, the initial state of an episode was generated by randomly sampling a time step from the given snippet. The Gaussian policy π_c uses a mean parameterized by a neural network as well as a fixed standard deviation of 0.1 for each action to induce robustness and to prepare for the noisy rollouts (Section 8.4.2). We used the Stable-Baselines3 (Raffin et al., 2021) implementation of PPO (Section 6.1) to train the experts. Our training took about *50 years* of wall-clock time. We give hyperparameters and training details in Section C.2.1.

Results To account for the snippets having different lengths and for the episode initialization scheme used in training, we report our evaluations in a length-normalized fashion.[1] For a snippet c (with length T_c) and some policy π, recall that we initialize the humanoid at some randomly chosen time step t_0 from c and then generate the trajectory τ by rolling out π from t_0 until either the end of the snippet or early termination. Let $R(\tau)$ and $L(\tau)$ denote the accumulated reward and the length of the trajectory τ, respectively. We define the normalized episode reward and normalized episode length of τ as $\frac{R(\tau)}{T_c - t_0}$ and $\frac{L(\tau)}{T_c - t_0}$, respectively. One consequence of this definition is that trajectories that are terminated early in a snippet yield smaller normalized episode rewards and lengths. Next, we define the average normalized episode reward and average normalized episode length of policy π on snippet c as $\hat{R}_c(\pi) = \mathbb{E}_{t_0 \sim c}\, \mathbb{E}_{\tau \sim \pi | t_0}\left[\frac{R(\tau)}{T_c - t_0}\right]$ and $\hat{L}_c(\pi) = \mathbb{E}_{t_0 \sim c}\, \mathbb{E}_{\tau \sim \pi | t_0}\left[\frac{L(\tau)}{T_c - t_0}\right]$, respectively. For example, if π always successfully tracks some MoCap snippet from any t_0 to the end of the snippet, π has an average normalized episode length of 1 on snippet c.

[1] We point out that PPO uses the original unnormalized reward for policy optimization.

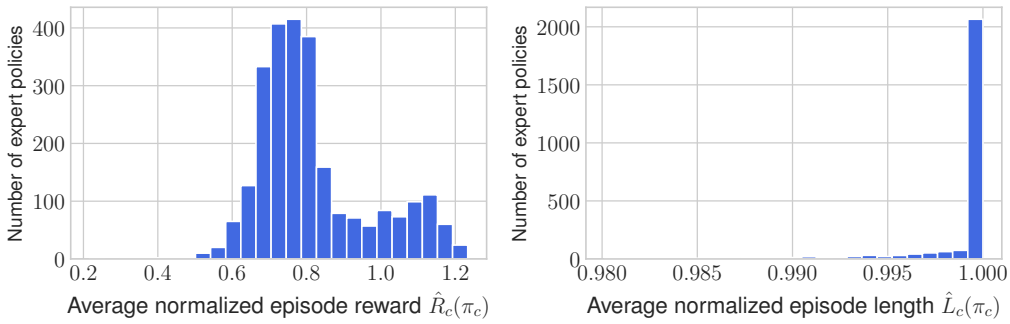


Figure 8.3: Clip expert results on the MoCap snippets within `dm_control`.

Overall, the clip experts reliably track the overwhelming majority of the MoCap snippets (Table 8.1 and Fig. 8.3). Averaged over all the snippets, the experts have a per-joint mean angle error of 0.062 radians. We find that 80% of the trained experts have an average normalized episode length of at least 0.999. We also observe there is a bimodal structure to the reward distribution in Fig. 8.3, which is due to many clips having artifacts like jittery limbs and extremities clipping through the ground. These artifacts limit the extent to which the humanoid can track the clip. Among the handful of experts with very low reward (between 0.2 and 0.5), we find that the corresponding clips are erroneously sped up, making them impossible to track in the simulator.

The experts produce motion that is generally indistinguishable from the MoCap reference (Fig. 8.4), from simple walking behaviors seen in the top row to highly coordinated motions like the cartwheel in the middle row. On some clips, the expert deviates from the clip because the demonstrated motion is too highly dynamic, such as the 360-degree jump in the bottom row. Instead, the expert typically learns some other behavior that keeps the episode from terminating early, which in this case is jumping without spinning. We also point out that, in these failure modes, the humanoid still tracks some portions of the reference, such as hand positions and orientations. Yuan and Kitani (2020) rectify similar tracking issues by augmenting the action space with external forces on certain parts of the humanoid body, but we do not explore this avenue since the issue only affects a small number of clips. We encourage the reader to visit the project web-

site (`microsoft.github.io/MoCapAct`) to see videos of the clip experts.

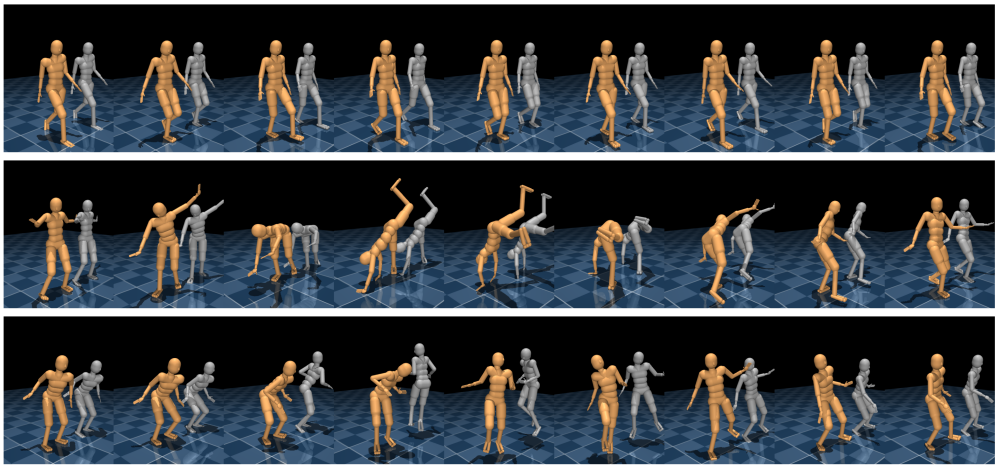
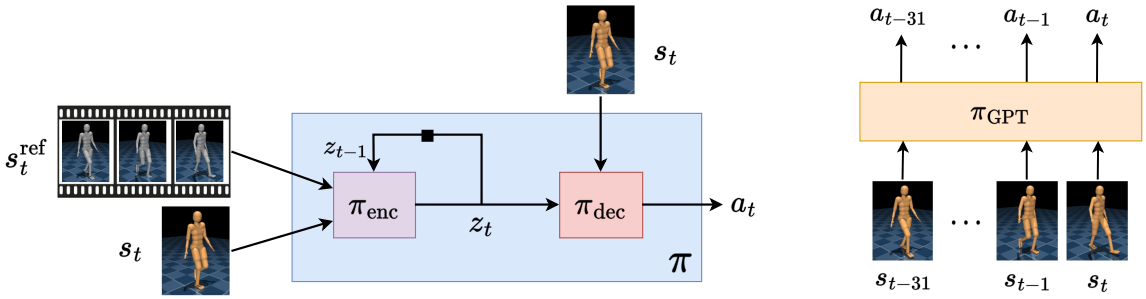


Figure 8.4: Visualizations of clip experts. The top two rows show episodes (first: walking, second: cartwheel) where the expert (bronze humanoid) closely tracks the corresponding MoCap clip (grey humanoid). The bottom row shows a clip where the expert and MoCap clip differ in behavior. The MoCap clip demonstrates a 360-degree jump, whereas the expert jumps without spinning.

8.4.2 Expert Rollouts

Following Merel, Hasenclever, et al. (2019), we rolled out the experts on their respective snippets and collected data from the rollouts into a dataset $\mathcal{D}$. In order to obtain a broad state coverage from the experts, we repeatedly rolled out the *stochastic* experts (i.e., with Gaussian noise injected into the actions) starting from different initial states. This injected noise helps the dataset cover states that a policy learned by imitating the dataset would visit, therefore mitigating the distribution shift issue for the learned policy (Laskey et al., 2017; Merel, Hasenclever, et al., 2019).

For each clip snippet c, we denote the corresponding expert policy as $\pi_c(a|s,t) = \mathcal{N}(a; \mu_c(s,t), 0.1^2 I)$, where $\mu_c(s,t)$ is the mean of the expert's action distribution. We initialized the humanoid at some point in the snippet c (half of the time at the beginning of the snippet and otherwise at some random point in the snippet). We then rolled out π_c until either the end of the snippet or early termination using the scheme from Section 8.4.1. At every time step t in the rollout, we logged the humanoid state s_t, the target



(a) Multi-clip tracking policy.

(b) GPT policy.

Figure 8.5: Policies used in the applications.

reference poses $s_t^{\text{ref}} = (\hat{s}_{t+1}^c, \ldots, \hat{s}_{t+5}^c)$ from the next five steps of the MoCap snippet, the expert's sampled action a_t, the expert's mean action $\bar{a}_t = \mu_c(s_t, t)$, the observed snippet reward $r_c(s_t, t)$, the estimated value $\hat{V}^{\pi_c}(s_t)$, and the estimated advantage $\hat{A}^{\pi_c}(s_t, a_t)$ into HDF5 files.

We release two versions of the rollout dataset:

- A "large" 600-gigabyte collection at 200 rollouts per snippet with a total of 67 million environment transitions (corresponding to 620 hours in the simulator) and
- A "small" 50-gigabyte collection at 20 rollouts per snippet with a total of 5.5 million environment transitions (corresponding to 51 hours in the simulator).

In our application of MoCapAct (Section 8.5), we used the "large" version of the dataset. We do observe, though, that the multi-clip policy results (Section 8.5.1) are similar when using either dataset.

8.5 Applications

We trained two policies (Fig. 8.5) using our dataset:

1. A hierarchical policy which can track all the MoCap snippets and be re-used for learning new high-level tasks (Section 8.5.1).

2. An autoregressive GPT model which generates motion from a given prompt (Section 8.5.2).

8.5.1 Multi-Clip Tracking Policy

We first show that the MoCapAct dataset can reproduce the results in Merel, Hasenclever, et al. (2019) by learning a single policy that tracks the entire MoCap dataset within `dm_control`. Our policy architecture (Fig. 8.5a) follows the same encoder-decoder scheme as Merel, Hasenclever, et al. (2019), who introduce a "motor intention" z_t that acts as a low-dimensional embedding of the MoCap reference s_t^{ref}. The intention z_t is then decoded into an action a_t. In other words, the policy π is factored into an encoder π_{enc} and a decoder π_{dec}. The encoder $\pi_{\text{enc}}(z_t|s_t, s_t^{\text{ref}}, z_{t-1})$ compresses the MoCap reference s_t^{ref} into an intention z_t and may use the current humanoid state s_t and previous intention z_{t-1} in predicting the current intention. Furthermore, the encoder outputs an intention that is *stochastic*, which models ambiguity in the MoCap reference and allows for the high-level behavior to be specified more coarsely. The decoder $\pi_{\text{dec}}(a_t|s_t, z_t)$ translates the sampled intention z_t into an action a_t with the aid of the state s_t as an additional input.

Training

In our implementation, the encoder outputs the mean and diagonal covariance of a Gaussian distribution over a 60-dimensional motor intention z_t. The decoder outputs the mean of a Gaussian distribution over actions with a standard deviation of 0.1 for each action. In training, we maximized a variant of the multi-step imitation learning objective from Merel, Hasenclever, et al. (2019):

$$\mathbb{E}_{\substack{(s_{1:T}, s_{1:T}^{\text{ref}}, a_{1:T}, c)\sim\mathcal{D},\\ z_{0:T}\sim\pi_{\text{enc}}}}\left[\sum_{t=1}^{T}\left[w_c(s_t,a_t)\log\pi_{\text{dec}}(a_t|s_t,z_t) - \beta\,\mathrm{KL}(\pi_{\text{enc}}(z_t|s_t,s_t^{\text{ref}},z_{t-1})\,\|\,p(z_t|z_{t-1}))\right]\right],$$

where T is the sequence length, w_c is a clip-dependent data-weighting function, $p(z_t|z_{t-1})$ is an autoregressive prior, and β is a hyperparameter.

The weighting function w_c allows for some data points to be considered more heavily,

which may be useful given the spectrum of expert performance. Letting λ be a hyperparameter, we consider the following four weighting schemes:

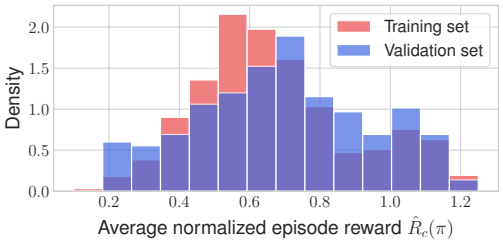
- Behavioral cloning (BC): $w_c(s,a) = 1$. This scheme is commonly used in imitation learning and treats every data point equally.
- Clip-weighted regression (CWR): $w_c(s,a) = \exp(\hat{R}_c(\pi_c)/\lambda)$. This scheme upweights data from snippets where the experts have higher average normalized rewards.
- Advantage-weighted regression (AWR) (Peng, Aviral Kumar, et al., 2019): $w_c(s,a) = \exp(\hat{A}^{\pi_c}(s,a)/\lambda)$. This scheme upweights actions that perform better than the expert's average return.
- Reward-weighted regression (RWR) (Peters and Schaal, 2007): $w_c(s,a) = \exp(\hat{Q}^{\pi_c}(s,a)/\lambda)$, where $\hat{Q}^{\pi_c}(s,a) = \hat{V}^{\pi_c}(s) + \hat{A}^{\pi_c}(s,a)$. This scheme upweights state-actions which have higher returns, which typically happens with good experts at earlier time steps in the corresponding snippet.

The KL divergence term encourages the decoder to follow a simple random walk. In this case, the prior has the form $p(z_t|z_{t-1}) = \mathcal{N}(z_t; \alpha z_{t-1}, \sigma^2 I)$, where $\alpha \in [0,1]$ is a hyperparameter and $\sigma = \sqrt{1-\alpha^2}$. This prior in turn encourages the marginals to be a spherical Gaussian, i.e., $p(z_t) = \mathcal{N}(z_t; 0, I)$. Furthermore, the regularization introduces a bottleneck (Alemi et al., 2017) that limits the information the intention z_t can provide about the state s_t and MoCap reference s_t^{ref}. This forces the encoder to only encode high-level information about the reference (e.g., direction of motion of leg) while excluding fine-grained details (e.g., precise velocity of each joint in leg).

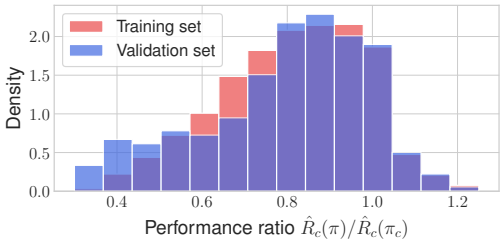
In our experiments, we found that the training takes about three hours on a single-GPU machine. More training details are available in Section C.2.2.

Table 8.2: Multi-clip results on the MoCap snippets, showing the mean and standard deviation over three seeds. For evaluation, we disable the Gaussian noise for π_{dec} but keep the stochasticity for π_{enc}.

	BC	CWR	AWR	RWR
Avg. normalized episode reward	0.654 ± 0.005	0.671 ± 0.003	0.661 ± 0.003	$\mathbf{0.688 \pm 0.002}$
Avg. normalized episode length	0.855 ± 0.004	0.858 ± 0.003	0.861 ± 0.001	$\mathbf{0.868 \pm 0.002}$



(a) Multi-clip policy's performance on training and validation sets.



(b) Performance of multi-clip policy relative to expert policies.

Figure 8.6: Performance of RWR-trained multi-clip policy.

Results All four regression approaches yielded broadly good results (Table 8.2), achieving 80% to 84% of the experts' performance on the MoCap dataset (cf. Table 8.1). We also see that every weighted regression scheme gave some improvement over the unweighted approach. AWR only gave 1% improvement over BC, likely because the experts are already near-optimal and the dataset lacks sufficient state-action coverage to reliably contain advantageous actions. CWR gave a 3% improvement over BC, which arises from the objective placing more emphasis on data coming from high-reward clips. Finally, RWR gave a 5% improvement over BC, which comes from increased weight on earlier time steps in high-reward clips. This is a sensible weighting scheme since executing a skill requires taking correct actions at earlier time steps before completing the skill at later time steps. As a point of comparison to prior work, the RWR-trained policy achieved an average reward-per-step (i.e., $\mathbb{E}[R(\tau)/L(\tau)]$) of 0.67 on the "Locomotion" subset of the MoCap data, which is 96% of the reward-per-step achieved by the large-scale RL approach of Hasenclever et al. (2020). We also find that the RWR-trained policy had a per-joint mean angle error of 0.085 radians.

To assess the generalization of the multi-clip policy, we trained the policy using RWR

on a subset of MoCapAct covering 90% of the MoCap clips. We treated the remaining 10% of the clips as a validation set when evaluating the multi-clip policy. We found that the multi-clip policy performs similarly on the training set and validation set clips (Fig. 8.6a), with the validation set performance even being slightly higher than the training set performance (mean of 0.699 vs. 0.674). This was likely because the clips in the validation set are slightly easier.

To account for the reward scale of the clips, we also report the multi-clip policy's performance relative to the clip experts (Fig. 8.6b). Again, the training set and validation set relative performances are very similar, though now the multi-clip policy has a small relative performance drop in the validation set (mean of 0.797 vs. 0.815). We also observe that the multi-clip policy outperforms the clip experts on 13% of the MoCap snippets.

We encourage the reader to visit the project website (`microsoft.github.io/MoCapAct`) to see videos of the multi-clip policy.

Re-Use for Reinforcement Learning

We re-used the decoder π_{dec} from an RWR-trained multi-clip policy for reinforcement learning to constrain the behaviors of the humanoid and speed up learning. In particular, we studied two tasks that require adept locomotion skills:

1. A sparse-reward go-to-target task, where the agent receives a non-zero reward only if the humanoid is sufficiently close to the target. The target relocates once the humanoid stands on it for a few time steps.

2. A velocity control task, where shaped rewards encourage the humanoid to go at a given speed in a given direction. The desired speed and direction change randomly every few seconds.

We treat π_{dec} as part of the environment and the motor intention z as the action. We thus learn a new high-level policy $\pi_{\text{task}}(z|s)$ that steers the low-level policy to maximize the

Table 8.3: Returns for the transfer tasks, showing the mean and standard deviation over five seeds.

	General low-level policy	Locomotion low-level policy	No low-level policy
Go-to-target	96.3 ± 2.8	66.1 ± 32.8	7.5 ± 1.1
Velocity control	1074 ± 55	884 ± 81	1157 ± 89

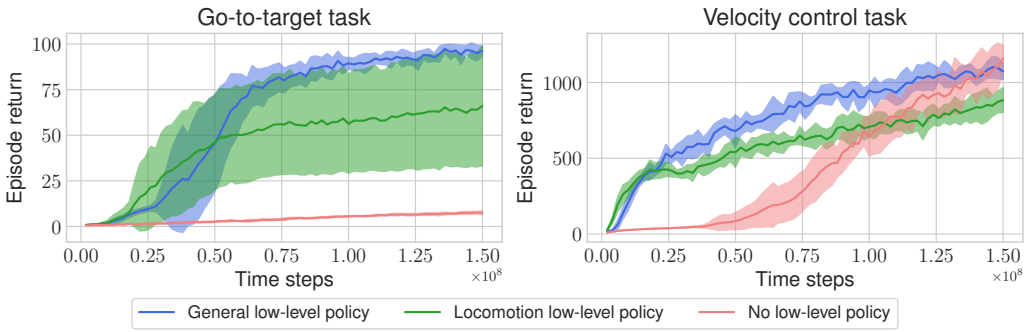


Figure 8.7: Training curves for transfer tasks. All experiments use five seeds.

task reward.

Given the tasks are locomotion-driven, we also considered a more specialized decoder with a 20-dimensional intention which was trained solely on locomotion clips from MoCapAct (called the "Locomotion" subset) to see if further restricting the learned skills offers any more speedup. As a baseline, we also performed RL without a low-level policy.

We find that re-using a low-level policy drastically sped up learning and usually produced higher returns (Table 8.3 and Fig. 8.7). For the go-to-target task, the locomotion-based low-level policy induced faster training than the more general low-level policy, though it did converge to lesser performance and on one out of five seeds converged to a very low reward. This performance gap was likely a combination of the lower dimensionality of the locomotion policy restricting the degree of control by the high-level policy and the "Locomotion" subset excluding some useful behaviors, a result also found by Hasenclever et al. (2020). The baseline without the low-level policy failed to learn the task. For the velocity control task, the locomotion-based policy induced slightly faster learning than the general policy but again resulted in lower reward. The baseline without the low-level policy learned the task more slowly, though it did achieve high reward eventually.

In both tasks, we find that including a pre-trained low-level policy produced much more realistic gaits. The humanoid efficiently ran from target to target in the go-to-target task and smoothly changed speeds and direction of motion in the velocity control task. On the other hand, the baseline approach produced unusual motions. In the go-to-target task, the humanoid convulsed and contorted itself towards the first target before falling to the ground. In the velocity control task, the humanoid rapidly tapped the feet to propel the body at the desired velocity. We encourage the reader to visit the project website (`microsoft.github.io/MoCapAct`) to see videos of the RL results.

8.5.2 Motion Completion with GPT

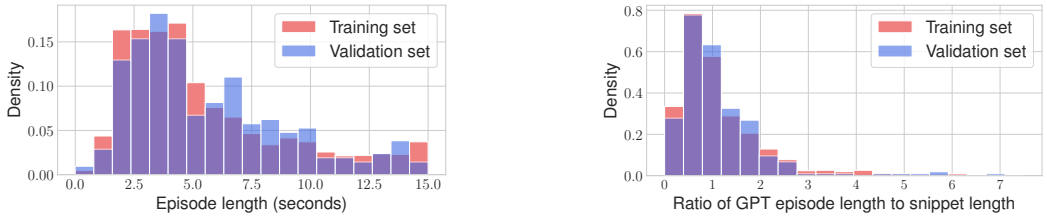
We also trained a GPT model (Radford, Narasimhan, et al., 2018; Radford, Wu, et al., 2019) based on the minGPT implementation (Karpathy, 2020) to generate motion. Starting with a motion prompt (sequence of humanoid observations generated by a clip expert), the GPT policy (Fig. 8.5b) autoregressively predicts actions from the context of recent humanoid observations. We trained the GPT by sampling 32-step sequences (corresponding to 1 second of motion) of humanoid observations $s_{(t-31):t}$ and expert's mean actions $\bar{a}_{(t-31):t}$ from the MoCapAct dataset $\mathcal{D}$ and performing supervised learning using the mean squared error loss on the predicted action sequence.

To roll out the policy, we provide the GPT policy with a 32-step prompt from a clip expert and let GPT roll out thereafter. The episode either terminates after 500 steps (about 15 seconds) or if a body part other than the feet touches the ground (e.g., humanoid falling over). On many clip snippets, the GPT model was able to control the humanoid for several seconds past the end of the prompt (Table 8.4 and Fig. 8.8a), with similar lengths on the training set and a held-out validation set of prompts. We also observed that on many clips the GPT can control the humanoid for several times longer than the length of the corresponding clip snippet (Table 8.4 and Fig. 8.8b).

To visualize the rollouts, we performed principal component analysis (PCA) on action

Table 8.4: Motion completion statistics on the MoCap snippets.

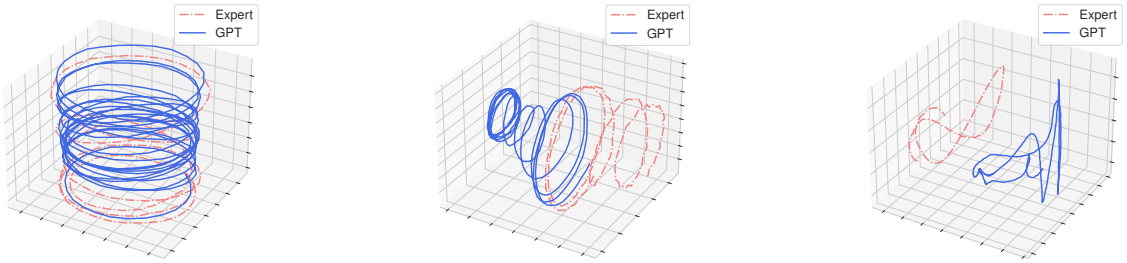
	Mean	Standard deviation	Median	Minimum	Maximum
Episode length (seconds)	5.47	3.47	4.38	0.23	15.00
Relative episode length	1.15	0.94	0.87	0.05	7.63



(a) Absolute episode lengths of GPT.

(b) Relative episode lengths of GPT.

Figure 8.8: Episode lengths of GPT on MoCap snippets.



(a) Locomotion clip where behaviors align.

(b) Locomotion clip where behaviors differ.

(c) Non-locomotion clip where behaviors differ.

Figure 8.9: PCA projections of action sequences of length 32 from experts and GPT.

sequences of length 32 applied by GPT and the snippet expert used to generate the motion prompt (Fig. 8.9). Qualitatively, we find that GPT usually repeated motions demonstrated in locomotion prompts, such as the running motion corresponding to Fig. 8.9a. Occasionally, GPT produced a different motion than the underlying clip, usually due to ambiguity in the prompt. For example, in Fig. 8.9b, GPT had the humanoid repeatedly step backwards, whereas the expert took repeated side steps. In Fig. 8.9c, the GPT policy performed an entirely different arm-waving motion than that of the expert. We encourage the reader to visit the project website (`microsoft.github.io/MoCapAct`) to see videos of GPT motion completion.

8.6 Discussion

We presented a dataset of high-quality MoCap-tracking policies and their rollouts for the `dm_control` humanoid environment. From these rollouts, we trained multi-clip tracking policies that can be re-used for new high-level tasks and GPT policies which can generate humanoid motion when given a prompt. We have open sourced our dataset, models, and code under permissive licenses.

We do point out that our models and data are only applicable to the `dm_control` environment, which uses MuJoCo as the backend simulator. We also point out that all considered clips only occur on flat ground and do not include any human or object interaction. Though this seems to limit the environments and tasks where this dataset is applicable, the `dm_control` package (Tunyasuvunakool et al., 2020) has tools to change the terrain, add more MoCap clips, and add objects (e.g., balls) to the environment. Indeed, prior work has used custom clips which include extra objects (Merel, Tunyasuvunakool, et al., 2020; Liu et al., 2022). While the dataset and domain may raise concerns on automation, we believe the considered simulated domain is limited enough to not be of ethical import.

This work significantly lowers the barrier of entry for simulated humanoid control, which promises to be a rich field for studying multi-task learning and motor intelligence. In addition to the showcases presented, we believe this dataset can be used in training other policy architectures like decision and trajectory transformers (Chen et al., 2021; Janner et al., 2021) or in setups like offline reinforcement learning (Fu et al., 2020; Levine, Aviral Kumar, et al., 2020) as the dataset allows research groups to bypass the time- and energy-consuming process of learning low-level motor skills from MoCap data.

CHAPTER 9
DISCUSSION

We have explored structured ways we can use data and learning in robotic control. We focused on two paradigms: model predictive control (MPC) and reinforcement learning (RL). Within MPC, we showed how neural networks could be used in place of traditional system identification techniques to represent dynamics models (Chapter 4). We showed that they could accurately learn complex phenomena and be used as part of an MPC framework for performant control. We also re-examined MPC from an online learning perspective (Chapter 5) and propose a general MPC algorithm based on dynamic mirror descent. This general algorithm contains various well-known MPC algorithms as special cases and exposes various shared hyperparameters. Within RL, we showed that data can be used as side information to guide the learning process. In the safe reinforcement learning setting, we showed that safety interventions can be used as a reward signal to teach an agent to become safe (Chapter 7). We proposed a safe RL algorithm based on this idea and showed that this algorithm comes with strong theoretical guarantees on the trained agent's safety and performance when safety interventions are disabled. In the simulated humanoid control setting, we leveraged hours of human motion capture (MoCap) data to teach agents natural and agile motions (Chapter 8). We generated an embodied dataset of these motions and used it to bootstrap behaviors for use in downstream RL and motion completion.

This thesis has only studied learning for control in contexts where the controller observes low-dimensional physical states. In our experiments, this corresponded to either ground-truth states when done in simulation or with high-quality state estimators when done in the real world. An interesting line of future work would be to expand the purview of learning to include perception so as to map directly from high-dimensional observations like images to actions. Indeed, this is an active area of research in the community and has

yielded impressive results on a wide variety of robotic platforms (Ananye Agarwal et al., 2023; Levine, Pastor, et al., 2018; Pan et al., 2020; Pinto and Gupta, 2016; Stachowicz et al., 2023; Zhuang et al., 2023). That said, though end-to-end policies offer the chance for a policy to learn to make up for perceptional defects (Levine, Finn, et al., 2016), it remains challenging to train end-to-end policies that have the same general capabilities as hierarchical and engineered robotic statcks (including those augmented with learning) (Roy et al., 2021; Sünderhauf et al., 2018).

One emerging trend in machine learning is to rely on extremely large and diverse datasets (Kirillov et al., 2023; Radford, Wu, et al., 2019) to train foundation models (Bommasani et al., 2021) that can be transferred to downstream tasks. Such models have been pivotal in the language domain (Brown et al., 2020; OpenAI, 2023). Another interesting line of future work would be to see if large-scale robotic data is sufficient to train capable robotic agents in lieu of traditional robotic methods. Similar efforts to curate large and diverse datasets and train corresponding models are being made in the robotics domain (Brohan et al., 2022; Collaboration et al., 2023; Reed et al., 2022; Zitkovich et al., 2023). However, it's not clear if robot data alone is sufficient to have a learned model deal with issues of embodiment or whether scaling laws that hold for language models (Henighan et al., 2020; Kaplan et al., 2020) transfer to learned robotic models.

Besides access to large amounts of data, another driving factor in modern machine learning successes has been proper architectural structure in neural nets, such as convolutional layers (Fukushima, 1980; Krizhevsky et al., 2012; LeCun, Bottou, et al., 1998) for vision and attention layers (Vaswani et al., 2017) for language. It is unclear if a different kind of structure is needed for the embodied domain, such as physics-informed (Greydanus et al., 2019) or planning layers (Amos et al., 2018; Srinivas et al., 2018; Tamar et al., 2016). Given the reliance of structure in robotics, it would be interesting to see if any such structure could be incorporated into a neural network for embodied agents that can both provide enough flexibility to maximally learn from data while giving enough inductive biases to

have the agent generalize to new scenarios.

Some Lessons Learned

- *Sampling-based MPC is surprisingly effective.* Sampling-based MPC essentially relies on a coarse strategy: Sample many different control trajectories, and pick the one with minimum cost. Despite throwing away all structure in the dynamics and cost by treating them as a black-box, this sampling-based strategy can be highly effective at solving robotic tasks. We've already showed that it can solve high-speed, off-road driving (Chapters 4 and 5), but it has also been successfully applied to quadruped locomotion (Yang et al., 2020), humanoids (Howell et al., 2022), and manipulation (Lowrey et al., 2019; Bhardwaj, Sundaralingam, et al., 2021). Also, while it may be tempting to say that it needs thousands of samples to solve tasks, prior work has shown that sampling-based MPC can solve high-dimensional (and potentially unstable) problems with 10 to 100 samples (Howell et al., 2022; Lowrey et al., 2019; Bhardwaj, Handa, et al., 2020). We have also shown that a smaller step size can mitigate the noisiness arising from few samples (Section 5.5.2). Since sampling-based MPC works most efficiently on GPUs and more GPU-accelerated physics simulators are becoming availabile (Freeman et al., 2021; Makoviychuk et al., 2021), sampling-based MPC may become a more widely adopted tool in robotic control.

- *Transformers can be effective at learning from data for control.* Taking MoCapAct (Chapter 8) as a case study, transformers like GPT can quite competently control high-dimensional systems like humanoids if trained on enough data. This can include reliably repeating motions conveyed in a prompt for several times the duration of the corresponding MoCap clip (video link). The transformer's high learning capacity can allow it to imitate motions with low data coverage (like cartwheels) that other architectures like recurrent networks struggle with (GPT video link, recurrent network video link).

Furthermore, like in language models (Radford, Wu, et al., 2019), transformers appear to benefit from learning on *diverse* data for the control setting. As a preliminary experiment to demonstrate this, I trained a GPT policy on the 200 rollouts that track a single jogging clip (video link) from the MoCapAct dataset. This specialized jogging policy consistently fails at reproducing the jogging motion (video link), while the "generalist" GPT policy trained on the entire MoCapAct dataset can reliably achieve the jogging motion (video link). This particular jogging motion is not conveyed in any other clip from MoCapAct, so the generalist GPT policy must be taking advantage of information contained in the other demonstrations, such as how the joints in a leg generally move together, in making its decision.

Nonetheless, transformers can still be brittle in the control setting. Prompts that are sufficiently out-of-distribution (even if not dramatically different than prompts in the training set) can cause the policy to immediately fail (video link 1, video link 2). Also, the policy tends to repeat motions indefinitely rather than transitioning between motions, limiting its capabilities. It would be interesting to investigate how to make the policy more robust (e.g., through RL-fine-tuning) and its overall behavior more controllable via interaction (e.g., through extra inputs given by a human user).

- *Good reward functions are hard to design.* Usually, reward functions are manually designed by the user. Because of this, an improperly designed reward function can be exploited by an optimized policy to produce ultimately undesired behavior (Clark and Amodei, 2016; Amodei et al., 2016). Furthermore, the weights of the reward function are typically set by hand in an iterative process. This makes reward design and tuning a laborious and often opaque process.

  As a working example, we consider the task of having a quadruped robot track user-given velocity commands (J. Lee et al., 2020). Typically, hand-designed reward terms are included to regularize the motion (J. Lee et al., 2020; Ashish Kumar et al., 2021;

Margolis et al., 2022; Rudin et al., 2022). The terms corresponding to the gait (e.g., foot contact forces, joint angle ranges) tend to affect the gait in an indirect and uninterpretable manner. This also makes the learned gait very sensitive to other hyperparameters such as choice of observation space (gait when policy has access to domain randomization parameters, gait when policy does not have access to domain randomization parameters) or PD gain settings (learned gait with default PD gains, learned gait with larger PD gains).

An alternative approach is to rely on *data* as a form of regularization. Usually, the data comes in the form of kinematic trajectories to track and can either come from motion capture of dogs (Peng, Coumans, et al., 2020) or trajectory optimization (Fuchioka et al., 2023). The perscribed motion can be tracked with either a hand-crafted reward (Peng, Coumans, et al., 2020; Fuchioka et al., 2023; Smith et al., 2022) or an adversarial approach that automatically learns the tracking reward (Escontrela et al., 2022; Y. Wang et al., 2023). One appealing outcome of this approach is that tracking the perscribed motions tends to automatically satisfy the hand-crafted regularization that would normally be included (e.g., low joint toruqes, proper base height). Thus, the motion imitation reward term can be used in place of hand-crafted regularization terms. Furthermore, since the perscribed motion acts as a strong learning signal, the learned motion is much more robust to hyperparameters like choice of joint PD gains (learned gait with default PD gains, learned gait with larger PD gains). However, this approach requires access to high-quality data with enough information to properly ground the motion. This may not be applicable if the data is hard to obtain or if it is misaligned with the task (e.g., morphological differences between data source and the robot).

It is not clear if there is a good middle ground between hand-designed rewards and incorporating data into the reward function. C. Li et al. (2022) show that rough demonstrations combined with a smaller number of hand-crafted regularization terms are

sufficient achieve maneuvers such as backflips. A compelling alternative to structuring the reward process would be to rely on human feedback via preferences (Christiano et al., 2017; Ouyang et al., 2022) to automatically shape the reward or to use large language models (Ma et al., 2023) to form a curriculum over reward functions.

Appendices

APPENDIX A
SYSTEM DESCRIPTIONS FOR PART I

A.1 Cartpole

The state is $x = (p, \varphi, \dot{p}, \dot{\varphi})$, where p is the cart position, φ is the pole's angle, $\dot{p}$ and $\dot{\varphi}$ are the corresponding velocities, and the control u is the force applied to the cart. The cart has mass m_c, and the pole has mass m_p and length ℓ. The equations of motion for the cartpole are:

$$\ddot{p} = \frac{1}{m_c + m_p \sin^2 \varphi}\big(u + m_p \sin\varphi\,(\ell\dot{\varphi}^2 + g\cos\varphi)\big)$$
$$\ddot{\varphi} = \frac{1}{\ell(m_c + m_p \sin^2 \varphi)}\big(-u\cos\varphi - m_p\ell\dot{\varphi}^2 \sin\varphi\cos\varphi - (m_c + m_p)g\sin\varphi\big),$$

where $g = 9.81\ \mathrm{m/s^2}$ is the gravitational acceleration. We give the system parameters for the cartpole in Table A.1. The modeled dynamics $\hat{p}$ in Section 5.5.1 uses a length of $0.346\ \mathrm{m}$.

Each time step is modeled using an Euler discretization of 0.02 seconds. Each episode of the problem lasts 500 time steps (i.e, 10 seconds) and has episode cost equal to the sum of encountered instantaneous costs. Both the true system and the model apply Gaussian additive noise to the commanded control with zero mean and a standard deviation of 5 newtons. For the continuous system, the commanded control is clamped to ± 25 newtons. For the discrete system, the controller can either command 10 newtons to the left, 10 newtons to the right, or 0 newtons.

Both the discrete and continuous controller use a planning horizon of 50 time steps (i.e., 1 second). For the continuous controller, we keep the standard deviation of the Gaussian distribution fixed at 2 newtons for each time step in the planning horizon. When applying

Table A.1: Parameters for cartpole

	m_c (kg)	m_p (kg)	ℓ (m)
Section 4.5.1	1	0.01	0.25
Section 5.5.1	0.711	0.209	0.326

a control u_t on the real cartpole, we choose the mode of π_{θ_t} rather than sample from the distribution.

We use two forms of the cost functions:

$$\text{Section 4.5.1} \quad c(x,u) = c_{\text{term}}(x) = 10p^2 + 500(\cos\varphi + 1)^2 + \dot{p}^2 + 15\dot{\varphi}^2$$

$$\text{Section 5.5.1} \quad c(x,u) = c_{\text{term}}(x) = 10p^2 + 500(\varphi - \pi)^2 + \dot{p}^2 + 15\dot{\varphi}^2 + 1000 \cdot \mathbf{1}\{|\varphi - \pi| \geq \Delta\},$$

where Δ is some threshold. For our experiments, we set $\Delta = 12^\circ = 0.21$ rad.

A.2 Quadrotor

We use the same system definition as that given by Appendix E of Williams, Aldrich, et al. (2017). We use the quadrotor model from Michael et al. (2010), but we treat body frame angular velocity rates and net thrust as control inputs. The cost function has the form:

$$c(x,u) = c_{\text{term}}(x) = (x - x^\star)^\mathsf{T} Q (x - x^\star) + 10^5 \cdot \mathbf{1}_{\text{crash}}(x)$$
$$x^\star = (50, 50, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$
$$Q = \text{diag}(1, 1, 1, 25, 25, 25, 1, 1, 1, 1, 1, 1)$$

Here, $(50, 50, 5)$ indicates the position target, and the indicator variable $\mathbf{1}_{\text{crash}}(x)$ which is 1 if the quadrotor crashes into an obstacle or the ground and 0 otherwise.

A.3 AutoRally

The state of the vehicle is $x = (p_x, p_y, \varphi, r, v_x, v_y, \dot{\varphi})$, where (p_x, p_y) is the position of the car in the global frame, φ and r are the yaw and roll angles, v_x and v_y are the longitudinal and lateral velocities in the car frame, and $\dot{\varphi}$ is the yaw rate. The control u we apply is the throttle and steering angle. For some weights $w_s, w_M, w_{\text{slip}}, w_{\text{crash}}$, the cost function is

$$c(x, u) = w_s|s - s^\star|^k + w_M M(p_x, p_y) + w_{\text{slip}}\, \mathbf{1}_{\text{slip}}(x)$$
$$c_{\text{term}}(x) = w_{\text{crash}}\, \mathbf{1}_{\text{crash}}(x).$$

Here, s and $s^\star$ are the current and target speed of the car, respectively. Note the speed is calculated as $s = \sqrt{v_x^2 + v_y^2}$. $M(p_x, p_y)$ is the positional cost of the car (low cost in center of track, high cost at edge of track), $\mathbf{1}_{\text{slip}}(x)$ is an indicator variable which activates if the slip angle[1] exceeds a certain threshold, and $\mathbf{1}_{\text{crash}}(x)$ is an indicator function which activates if the car leaves the track at all in the trajectory. Note that the terminal cost depends on the trajectory instead of the terminal state. Each time step represents 0.02 seconds for every experiment except the real-world experiment with a target of $11\ \text{m/s}$ where each time step represents 0.025 seconds. The length of the planning trajectory is 100 time steps (i.e., either 2 seconds or 2.5 seconds depending on the length of the time step). The values for the cost function parameters are given in Table A.2.

The control space for each of the throttle and steering angle is normalized to the range $[-1, 1]$. For our experiments, we clamp the throttle to $[-1, 0.65]$. In simulated experiments, the standard deviations of the throttle and steering angle distributions were 0.3 and 0.275, respectively. In the real world experiments, they were both set to 0.3. When applying a control u_t on the car, we chose the mean of π_{θ_t} rather than sampling from the distribution.

In simulation, the environment (Fig. 5.7) is an elliptical track approximately 3 meters

[1]The slip angle is defined as $-\arctan \frac{v_y}{|v_x|}$, which gives the angle between the direction the car is pointing and the direction in which it is actually traveling.

Table A.2: Cost function settings for AutoRally experiments.

	$s^\star$ (m/s)	k	w_s	w_M	w_{slip}	w_{crash}	Slip angle threshold (rad)
Gazebo simulator	11	1	30	250	10	10^5	0.275
Real world (Section 4.5.2)	9–13	2	2.5	100	50	10^5	0.275–0.375
Real world (Section 5.5.2)	9 or 11	2	4.25	200	100	10^5	0.9

wide and 30 meters across at its furthest point. The real-world dirt track is about 5 meters wide and and has a track length of 170 meters. All reported results for simulated experiments were gathered using 30 consecutive laps in the counter-clockwise direction for each parameter setting. For real-world experiments, results were gathered using ten laps for each parameter setting when the target speed is $9\ \mathrm{m/s}$ and five laps for $11\ \mathrm{m/s}$.

APPENDIX B

SAFE REINFORCEMENT LEARNING USING ADVANTAGE-BASED INTERVENTION

B.1 Missing Proofs

B.1.1 Useful Lemmas

Lemma 2. *For any γ-discounted MDP with reward function r, the identity $V^\pi(d_0) = (1-\gamma)\sum_{h=0}^{\infty}\gamma^h U_h^\pi(d_0)$ holds, where $U_h^\pi(d_0) = \mathbb{E}_{\rho^\pi(\tau)}[\sum_{t=0}^{h} r(s_t,a_t)]$ is the undiscounted h-step return.*

Proof. The proof follows from exchanging the order of summations:

$$\begin{aligned}
(1-\gamma)\sum_{h=0}^{\infty}\gamma^h U_h^\pi(d_0) &= (1-\gamma)\sum_{h=0}^{\infty}\gamma^h\,\mathbb{E}_{\rho^\pi(\tau)}\left[\sum_{t=0}^{h} r(s_t,a_t)\right] \\
&= (1-\gamma)\,\mathbb{E}_{\rho^\pi(\tau)}\left[\sum_{t=0}^{\infty} r(s_t,a_t)\sum_{h=t}^{\infty}\gamma^h\right] \\
&= \mathbb{E}_{\rho^\pi(\tau)}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right] \\
&= V^\pi(d_0)
\end{aligned}$$

□

Lemma 3 (Performance Difference Lemma (Kakade and Langford, 2002; Cheng, Kolobov, and Alekh Agarwal, 2020))**.** *Let $\mathcal{M}$ be an MDP and π be a policy. For any function $f : \mathcal{S} \to \mathbb{R}$ and any initial state distribution d_0, it holds that*

$$V^\pi(d_0) - f(d_0) = \frac{1}{1-\gamma}\,\mathbb{E}_{d^\pi(s,a)}[r(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}[f(s')] - f(s)]$$

Corollary 1. *Let $\mathcal{M}$ and $\hat{\mathcal{M}}$ be MDPs with common state and action spaces. For any policy π, the difference in value functions in $\mathcal{M}$ and $\hat{\mathcal{M}}$ satisfies*

$$V^\pi(d_0) - \hat{V}^\pi(d_0) = \frac{1}{1-\gamma}\,\mathbb{E}_{d^\pi(s,a)}[(\mathcal{D}^\pi \hat{Q}^\pi)(s,a)]$$

where $\mathcal{D}^\pi$ is the temporal-difference operator of $\mathcal{M}$:

$$(\mathcal{D}^\pi Q)(s,a) \triangleq (\mathcal{B}^\pi Q)(s,a) - Q(s,a),$$

and $\mathcal{B}^\pi$ is the Bellman operator of $\mathcal{M}$:

$$(\mathcal{B}^\pi Q)(s,a) \triangleq r(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}[Q(s',\pi)].$$

Proof. Set $f = \hat{V}^\pi$ and observe that $\hat{V}^\pi(s) = \hat{Q}^\pi(s,\pi)$. □

B.1.2 Proof of Equivalent CMDP Formulation in Section 3.2

We show that Eq. (7.1) and Eq. (7.2) are the same by proving the equivalence

$$(1-\gamma)\sum_{h=0}^{\infty}\gamma^h\,\mathbb{P}_{\rho^\pi(\tau)}(\tau^h \subset \mathcal{S}_{\text{safe}}) \geq 1-\delta \quad\iff\quad \bar{V}^\pi(d_0) \leq \delta. \tag{B.1}$$

By the definition of the cost function $c(s,a) = \mathbf{1}\{s = s_\triangleright\}$ and absorbing property of $\mathcal{S}_{\text{unsafe}} = \{s_\triangleright, s_\circ\}$, we can write

$$1 - \mathbb{P}_{\rho^\pi(\tau)}(\tau^h \subset \mathcal{S}_{\text{safe}}) = \mathbb{P}_{\rho^\pi(\tau)}(s_\triangleright \in \tau^h) = \mathbb{E}_{\rho^\pi(\tau)}\left[\sum_{t=0}^{h} c(s_t,a_t)\right] \tag{B.2}$$

since $s_\triangleright$ can only appear at most once within τ^h. Substituting this equality into the negation of the chance constraint,

$$\begin{aligned}1-(1-\gamma)\sum_{h=0}^{\infty}\gamma^h\,\mathbb{P}_{\rho^\pi(\tau)}(\tau^h\subset\mathcal{S}_{\text{safe}}) &= (1-\gamma)\sum_{h=0}^{\infty}\gamma^h\,\mathbb{E}_{\rho^\pi(\tau)}\left[\sum_{t=0}^{h}c(s_t,a_t)\right]\\ &= \mathbb{E}_{\rho^\pi(\tau)}\left[\sum_{t=0}^{\infty}\gamma^t c(s_t,a_t)\right]\\ &= \bar{V}^\pi(d_0),\end{aligned}$$

where the second equality follows from Lemma 2. Therefore, Eq. (B.1) holds.

B.1.3 Proof for Intervention Rules in Section 3.3

Admissible Rules and Pessimism

Proposition 2. *If $\mathcal{G}=(\bar{Q},\mu,\eta)$ is σ-admissible, then $\bar{Q}^\mu(s,a)\le\bar{Q}(s,a)+\frac{\sigma}{1-\gamma}$ for all $s\in\mathcal{S}_{\text{safe}}$ and $a\in\mathcal{A}$.*

Proof. The proof follows by repeating the inequality of $\bar{Q}$.

$$\begin{aligned}\bar{Q}(s,a) &\ge c(s,a)+\gamma\,\mathbb{E}_{p(s'|s,a)}[\bar{Q}(s',\mu)]-\sigma\\ &\ge c(s,a)+\gamma\,\mathbb{E}_{p(s'|s,a)}\left[c(s',\mu)+\gamma\,\mathbb{E}_{p(s''|s',\mu)}[\bar{Q}^\mu(s'',\mu)]\right]-(1+\gamma)\sigma\\ &\;\;\vdots\\ &\ge \bar{Q}^\mu(s',\mu)-\frac{\sigma}{1-\gamma}.\end{aligned}$$

□

Example Intervention Rules

Proposition 3 (Intervention Rules)**.** *The following are true.*

1. ***Baseline policy**: Given a baseline policy μ of $\mathcal{M}$, $\mathcal{G}=(\bar{Q}^\mu,\mu,\eta)$ or $\mathcal{G}=(\bar{Q}^\mu,\mu^+,\eta)$ is admissible, where μ^+ is the greedy policy that treats $\bar{Q}^\mu$ as a cost.*

2. ***Composite intervention**: Given K intervention rules $\{\mathcal{G}_k\}_{k=1}^K$, where each $\mathcal{G}_k = (\bar{Q}_k, \mu_k, \eta)$ is σ_k-admissible. Define $\bar{Q}_{\min}(s,a) = \min_k \bar{Q}_k(s,a)$ and let $\mu_{\min}$ be the greedy policy w.r.t. $\bar{Q}_{\min}$, and $\sigma_{\max} = \max_k \sigma_k$. Then, $\mathcal{G} = (\bar{Q}_{\min}, \mu_{\min}, \eta)$ is $\sigma_{\max}$-admissible.*

3. ***Value iteration**: Define $\bar{\mathcal{T}}$ as $\bar{\mathcal{T}}Q(s,a) \triangleq c(s,a) + \gamma\, \mathbb{E}_{p(s'|s,a)}[\min_{a'} Q(s',a')]$. If $\mathcal{G} = (\bar{Q}, \mu, \eta)$ is σ-admissible, then $\mathcal{G}^k = (\bar{\mathcal{T}}^k\bar{Q}, \mu^k, \eta)$ is $\gamma^k\sigma$-admissible, where μ^k is the greedy policy that treats $\bar{\mathcal{T}}^k\bar{Q}$ as a cost.*

4. ***Optimal intervention**: Let $\bar{\pi}^\star$ be an optimal policy for $\bar{\mathcal{M}}$, and let $\bar{Q}^\star$ be the corresponding state-action value function. Then $\mathcal{G}^\star = (\bar{Q}^\star, \bar{\pi}^\star, \eta)$ is admissible.*

5. ***Approximation**: For σ-admissible $\mathcal{G} = (\bar{Q}, \mu, \eta)$, consider $\hat{Q}$ such that $\hat{Q}(s,a) \in [0,\gamma]$ for all $s \in \mathcal{S}_{\text{safe}}$ and $a \in \mathcal{A}$. If $\|\hat{Q} - \bar{Q}\|_\infty \leq \delta$, then $\hat{\mathcal{G}} = (\hat{Q}, \mu, \eta)$ is $(\sigma + (1+\gamma)\delta)$-admissible.*

Proof. We show each intervention rule $\mathcal{G} = (\bar{Q}, \mu, \eta)$ below satisfies the admissibility condition

$$\bar{Q}(s,a) + \sigma \geq c(s,a) + \gamma\, \mathbb{E}_{p(s'|s,a)}[\bar{Q}(s',\mu)].$$

For convenience, we define the Bellman operator $\bar{\mathcal{B}}^\mu$ as $(\bar{\mathcal{B}}^\mu Q)(s,a) \triangleq c(s,a) + \gamma\, \mathbb{E}_{p(s'|s,a)}[Q(s,\mu)]$. Then, the admissibility condition can be written as $\bar{Q}(s,a) + \sigma \geq (\bar{\mathcal{B}}^\mu\bar{Q})(s,a)$ for any $s \in \mathcal{S}_{\text{safe}}$ and $a \in \mathcal{A}$. Also, we write $\bar{Q} \in [0,\gamma]$ on $\mathcal{S}_{\text{safe}}$ if $\bar{Q}(s,a) \in [0,\gamma]$ for all $s \in \mathcal{S}_{\text{safe}}$ and $a \in \mathcal{A}$.

1. **Baseline policy**: We know $\mathcal{G} = (\bar{Q}^\mu, \mu, \eta)$ is admissible since $\bar{Q}^\mu = \bar{\mathcal{B}}^\mu\bar{Q}^\mu$. For $\mathcal{G} = (\bar{Q}^\mu, \mu^+, \eta)$, we have $\bar{Q}^\mu \geq \bar{\mathcal{B}}^{\mu^+}\bar{Q}^\mu$ since μ^+ is greedy with respect to $\bar{Q}^\mu$. Also, by the definition of the cost c and transition dynamics P, we know that $\bar{Q}^\mu(s,a) \in [0,1]$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Furthermore, when $s \in \mathcal{S}_{\text{safe}}$, we have $c(s,a)$ and therefore $\bar{Q}^\mu(s,a) = \gamma\, \mathbb{E}_{p(s'|s,a)}[\bar{Q}^\mu(s',\mu)] \in [0,\gamma]$.

2. **Composite intervention**: For any $k \in \{1, \ldots, K\}$, the following bound holds:

$$\begin{aligned}
(\bar{\mathcal{B}}^{\mu_{\min}} \bar{Q}_{\min})(s,a) &= c(s,a) + \gamma\, \mathbb{E}_{p(s'|s,a)}[\bar{Q}_{\min}(s', \mu_{\min})] \\
&\leq c(s,a) + \gamma\, \mathbb{E}_{p(s'|s,a)}[\bar{Q}_{\min}(s', \mu_k)] \\
&\leq c(s,a) + \gamma\, \mathbb{E}_{p(s'|s,a)}[\bar{Q}_k(s', \mu_k)] \\
&\leq \bar{Q}_k(s,a) + \sigma_k \\
&\leq \bar{Q}_k(s,a) + \sigma_{\max},
\end{aligned}$$

where the first inequality comes from $\mu_{\min}$ being a minimizer of $\bar{Q}_{\min}$, and the second inequality from $\bar{Q}_{\min}$ being a pointwise minimum of $\{\bar{Q}_k\}_{k=1}^K$. Since this holds for every k, we conclude:

$$\begin{aligned}
(\bar{\mathcal{B}}^{\mu_{\min}} \bar{Q}_{\min})(s,a) &\leq \min_k [\bar{Q}_k(s,a) + \sigma_{\max}] \\
&= \min_k \bar{Q}_k(s,a) + \sigma_{\max} \\
&= \bar{Q}_{\min}(s,a) + \sigma_{\max},
\end{aligned}$$

which establishes the Bellman bound holds. Finally, since each $\bar{Q}_k$ satisfies $\bar{Q}_k \in [0, \gamma]$ on $\mathcal{S}_{\text{safe}}$, we conclude that $\bar{Q}_{\min}$ has the same range. Therefore, $\mathcal{G}$ is $\sigma_{\max}$-admissible.

3. **Value iteration**: Define the shortcuts $\bar{Q}_k \triangleq \bar{\mathcal{T}}^k \bar{Q}$, where $\bar{Q}_0 = \bar{Q}$.

We first show that, by policy improvement, we have $\bar{Q}_k(s,a) \leq \bar{Q}_{k-1}(s,a) + \gamma^{k-1}\sigma$

on $\mathcal{S}_{\text{safe}} \times \mathcal{A}$. We do this by induction. First, we see that:

$$\begin{aligned}
\bar{Q}_1(s,a) &= \bar{\mathcal{T}}\,\bar{Q}_0(s,a) \\
&= c(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}\left[\min_{a'} \bar{Q}_0(s',a')\right] \\
&= c(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}\left[\min_{a'} \bar{Q}(s',a')\right] \\
&\leq c(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}[\bar{Q}(s',\mu)] \\
&\leq \bar{Q}(s,a) + \sigma \\
&= \bar{Q}_0(s,a) + \sigma.
\end{aligned}$$

Now suppose $\bar{Q}_\kappa(s,a) \leq \bar{Q}_{\kappa-1}(s,a) + \gamma^{\kappa-1}\sigma$ holds on $\mathcal{S}_{\text{safe}} \times \mathcal{A}$ for some κ. Therefore,

$$\begin{aligned}
\bar{Q}_{\kappa+1}(s,a) &= \bar{\mathcal{T}}\,\bar{Q}_\kappa(s,a) \\
&= c(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}\left[\min_{a'} \bar{Q}_\kappa(s',a')\right] \\
&\leq c(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}\left[\min_{a'} \bar{Q}_{\kappa-1}(s',a')\right] + \gamma^\kappa\sigma \\
&= \bar{\mathcal{T}}\,\bar{Q}_{\kappa-1}(s,a) + \gamma^\kappa\sigma \\
&= \bar{Q}_\kappa(s,a) + \gamma^\kappa\sigma.
\end{aligned}$$

Using this inequality, we now show that $\mathcal{G}^k = (\bar{Q}_k, \mu^k, \eta)$ is indeed $\gamma^k\sigma$-admissible:

$$\begin{aligned}
\bar{Q}_k(s,a) &= \bar{\mathcal{T}}\,\bar{Q}_{k-1}(s,a) \\
&= c(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}\left[\min_{a'} \bar{Q}_{k-1}(s',a')\right] \\
&\geq c(s,a) + \gamma\,\mathbb{E}_{p(s'|s,a)}\left[\min_{a'} \bar{Q}_k(s',a')\right] - \gamma^k\sigma \\
&= \bar{\mathcal{T}}\,\bar{Q}_k(s,a) - \gamma^k\sigma \\
&= \bar{\mathcal{B}}^{\mu^k}\bar{Q}_k(s,a) - \gamma^k\sigma,
\end{aligned}$$

where the inequality was used in the third line. This establishes that the Bellman bound holds.

We prove that $\bar{Q}_k \in [0, \gamma]$ on $\mathcal{S}_{\text{safe}}$ by induction. Clearly, $\bar{Q}_0 = \bar{Q} \in [0, \gamma]$ on $\mathcal{S}_{\text{safe}}$ since $\mathcal{G}$ is σ-admissible. Now suppose $\bar{Q}_\kappa \in [0, \gamma]$ on $\mathcal{S}_{\text{safe}}$ for some κ. Then, for any $s \in \mathcal{S}_{\text{safe}}$ and $a \in \mathcal{A}$, we have $\bar{Q}_{\kappa+1}(s, a) = \gamma \, \mathbb{E}_{p(s'|s,a)}[\min_{a'} \bar{Q}_\kappa(s, a)] \in [0, \gamma]$. Therefore, $\mathcal{G}^k$ is $\gamma^k \sigma$-admissible.

4. **Optimal intervention**: This is a special case of case 1.

5. **Approximation**: The following holds on $\mathcal{S}_{\text{safe}} \times \mathcal{A}$:

$$\begin{aligned}
\hat{Q}(s, a) &= \hat{Q}(s, a) - \bar{Q}(s, a) + \bar{Q}(s, a) \\
&\geq -\delta + (\bar{\mathcal{B}}^\mu \bar{Q})(s, a) - \sigma \\
&= -\delta - \sigma + c(s, a) + \gamma \, \mathbb{E}_{p(s'|s,a)}[\bar{Q}(s', \mu)] \\
&\geq -\delta - \sigma + c(s, a) + \gamma \, \mathbb{E}_{p(s'|s,a)}[\hat{Q}(s', \mu) - \delta] \\
&= -\delta - \sigma - \gamma\delta + \bar{\mathcal{B}}^\mu \hat{Q}(s, a).
\end{aligned}$$

That is, $(\bar{\mathcal{B}}^\mu \hat{Q})(s, a) \leq \hat{Q}(s, a) + \sigma + (1 + \gamma)\delta$. Therefore, $\hat{\mathcal{G}} = (\hat{Q}, \mu, \eta)$ is $(\sigma + (1 + \gamma)\delta)$-admissible.

□

Safety Guarantee of Shielded Policy

Before proving Theorem 2, we prove two lemmas, one showing that the average advantage of a shielded policy satisfies the intervention threshold (Lemma 4) and the other stating that the cost-value function is equal to the expected occupancy of the unsafe set (Lemma 5).

Lemma 4. *For some policy π and intervention rule $\mathcal{G} = (\bar{Q}, \mu, \eta)$, let $\pi' \triangleq \mathcal{G}(\pi)$ and $\bar{A}(s, a) \triangleq \bar{Q}(s, a) - \bar{Q}(s, \mu)$. Then, $\bar{A}(s, \pi') \leq \eta$ for any $s \in \mathcal{S}_{\text{safe}}$.*

Proof. We use the definition of π' (in Eq. (7.4)), the facts that $\bar{A}(s,\mu) = 0$, and that $(s,a) \notin \mathcal{I}$ if and only if $\bar{A}(s,a) \leq \eta$. The following then holds:

$$\begin{aligned}
\bar{A}(s,\pi') &= \sum_{a\in\mathcal{A}} \pi'(a|s)\bar{A}(s,a) \\
&= \sum_{a:(s,a)\notin\mathcal{I}} \pi(a|s)\bar{A}(s,a) + w(s)\sum_{a\in\mathcal{A}} \mu(a|s)\bar{A}(s,a) \\
&\leq \eta \sum_{a:(s,a)\notin\mathcal{I}} \pi(a|s) + w(s)\bar{A}(s,\mu) \\
&\leq \eta \cdot 1 + w(s)\cdot 0 \\
&= \eta.
\end{aligned}$$

□

Lemma 5. *For any policy π,*

$$\mathbb{E}_{d^\pi(s)}[\mathbf{1}\{s \in \{s_\triangleright, s_\circ\}\}] = \bar{V}^\pi(d_0).$$

Proof. We know from the definition of the cost function that $\bar{V}^\pi(d_0) = \frac{1}{1-\gamma}\mathbb{E}_{d^\pi(s)}[\mathbf{1}\{s = s_\triangleright\}]$. From the absorbing property of $\mathcal{S}_{\text{unsafe}}$, we have $\mathbb{E}_{d^\pi(s)}[\mathbf{1}\{s = s_\circ\}] = \frac{\gamma}{1-\gamma}\mathbb{E}_{d^\pi(s)}[\mathbf{1}\{s = s_\triangleright\}]$. We can then derive

$$\begin{aligned}
\mathbb{E}_{d^\pi(s)}[\mathbf{1}\{s \in \{s_\triangleright, s_\circ\}\}] &= \mathbb{E}_{d^\pi(s)}[\mathbf{1}\{s = s_\triangleright\}] + \mathbb{E}_{d^\pi(s)}[\mathbf{1}\{s = s_\circ\}] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{d^\pi(s)}[\mathbf{1}\{s = s_\triangleright\}] \\
&= \bar{V}^\pi(d_0).
\end{aligned}$$

□

We now prove the safety guarantee of the shieled policy π'.

Theorem 2 (Safety of Shielded Policy)**.** *Let $\mathcal{G} = (\bar{Q}, \mu, \eta)$ be σ-admissible as per Definition 1. For any policy π, let $\pi' = \mathcal{G}(\pi)$. Then,*

$$\bar{V}^{\pi'}(d_0) \leq \bar{Q}(d_0, \mu) + \frac{\min\{\sigma + \eta, 2\gamma\}}{1 - \gamma}. \tag{7.9}$$

Proof.

$$\bar{Q}(s_{\triangleright}, a) = 1 \qquad \text{and} \qquad \bar{Q}(s_{\circ}, a) = 0 \qquad \text{for all } a \in \mathcal{A}.$$

Define $\bar{V}(s) \triangleq \bar{Q}(s, \mu)$. Since $c(s,a) + \gamma\, \mathbb{E}_{p(s'|s,a)}[\bar{V}(s')] = \bar{V}(s)$ when $s \in \{s_{\triangleright}, s_{\circ}\}$, we can use the performance difference lemma (Lemma 3) to derive

$$\begin{aligned}
\bar{V}^{\pi'}(d_0) - \bar{Q}(d_0, \mu) &= \frac{1}{1-\gamma}\, \mathbb{E}_{d^{\pi'}(s,a)}[c(s,a) + \gamma\, \mathbb{E}_{p(s'|s,a)}[\bar{V}(s')] - \bar{V}(s)] \\
&= \frac{1}{1-\gamma}\, \mathbb{E}_{d^{\pi'}(s,a)}[(c(s,a) + \gamma\, \mathbb{E}_{p(s'|s,a)}[\bar{V}(s')] - \bar{V}(s))\, \mathbf{1}\{s \notin \{s_{\triangleright}, s_{\circ}\}\}] \\
&\leq \frac{1}{1-\gamma}\, \mathbb{E}_{d^{\pi'}(s,a)}[(\min\{\sigma, \gamma\} + \bar{Q}(s,a) - \bar{V}(s))\, \mathbf{1}\{s \notin \{s_{\triangleright}, s_{\circ}\}\}] \\
&\leq \frac{\min\{\sigma, \gamma\} + \min\{\eta, \gamma\}}{1-\gamma}\, \mathbb{E}_{d^{\pi'}(s)}[\mathbf{1}\{s \notin \{s_{\triangleright}, s_{\circ}\}\}] \\
&= \frac{\min\{\sigma, \gamma\} + \min\{\eta, \gamma\}}{1-\gamma} \bar{V}^{\pi'}(d_0),
\end{aligned}$$

where the first inequality comes from $\bar{Q}$ being σ-admissible and γ-admissible, the second inequality from $\bar{A}(s, \pi') \leq \eta$ (Lemma 4) and $\bar{A}(s, \pi') \leq \gamma$ (Definition 1) for $s \notin \{s_{\triangleright}, s_{\circ}\}$, and the last equality from Lemma 5.

Therefore, after some algebraic rearrangement,

$$\begin{aligned}
\bar{V}^{\pi'}(d_0) &\leq \frac{(1-\gamma)\bar{Q}(d_0, \mu) + \min\{\sigma, \gamma\} + \min\{\eta, \gamma\}}{1 - \gamma + \min\{\sigma, \gamma\} + \min\{\eta, \gamma\}} \\
&\leq \bar{Q}(d_0, \mu) + \frac{\min\{\sigma, \gamma\} + \min\{\eta, \gamma\}}{1-\gamma} \\
&\leq \bar{Q}(d_0, \mu) + \frac{\min\{\sigma + \eta, 2\gamma\}}{1-\gamma}.
\end{aligned}$$

□

An Optimal Intervention Rule

First, we show that every state-action pair visited by π' will not have an advantage function lower than that of the optimal policy for $\bar{\mathcal{M}}$.

Lemma 6. *Let $\bar{\pi}^\star$ be an optimal policy for $\bar{\mathcal{M}}$, $\bar{Q}^\star$ be its state-action value function, and $\bar{V}^\star$ be its state value function. Let $G_0 = \{(\bar{Q}, \mu, 0) : (\bar{Q}, \mu, 0)$ is admissible, $\bar{Q}(d_0, \mu) = \bar{V}^\star(d_0)\}$ be a subset of admissible intervention rules with a threshold of zero and average $\bar{Q}$ that matches $\bar{V}^\star$. Define $\bar{A}^\star(s, a) = \bar{Q}^\star(s, a) - \bar{Q}^\star(s, \bar{\pi}^\star)$ as the advantage function of the optimal policy. For some intervention rule $\mathcal{G} \in G_0$ and policy π, let $\pi' = \mathcal{G}(\pi)$.*

Then, the inequality $\bar{A}(s, a) \geq \bar{A}^\star(s, a)$ holds for all $a \in \mathcal{A}$ almost surely over the distribution $d^{\pi'}(s)$.

Proof. First, we show by induction that running π' starting from d_0 results in the agent staying in the subset $\mathcal{S}_\mathcal{G} = \{s \in \mathcal{S} : \bar{Q}(s, \mu) = \bar{V}^\star(s)\}$.

For $t = 0$, consider some $s_0 \sim d_0$. We observe from admissibility of $\mathcal{G}$ and Proposition 2 that $\bar{Q}(s, a) \geq \bar{Q}^\mu(s, a) \geq \bar{V}^\star(s)$ on $\mathcal{S} \times \mathcal{A}$. Since $\bar{Q}(d_0, \mu) = \bar{V}^\star(d_0)$, we conclude that $\bar{Q}(s_0, \mu) = \bar{V}^\star(s_0)$. Therefore, $s_0 \in \mathcal{S}_\mathcal{G}$ almost surely over d_0.

Now suppose the agent is in $\mathcal{S}_\mathcal{G}$ with probability one at some time step t. Consider some $s_t \sim \rho^{\pi'}$ (observing that $s_t \in \mathcal{S}_\mathcal{G}$). We assume that $s_t \in \mathcal{S}_{\text{safe}}$ (otherwise, the below is trivially true as there is no intervention outside $\mathcal{S}_{\text{safe}}$). By Lemma 4 and admissibility, we

can derive:

$$
\begin{aligned}
0 = \eta \geq \bar{A}(s_t, \pi') & \\
&= \bar{Q}(s_t, \pi') - \bar{Q}(s_t, \mu) \\
&\geq c(s_t, \pi') + \gamma \, \mathbb{E}_{p(s_{t+1}, s_t, \pi')}[\bar{Q}(s_{t+1}, \mu)] - \bar{Q}(s_t, \mu) \\
&= \gamma \, \mathbb{E}_{p(s_{t+1}|s_t, \pi')}[\bar{Q}(s_{t+1}, \mu)] - \bar{Q}(s_t, \mu) \\
&= \gamma \, \mathbb{E}_{p(s_{t+1}|s_t, \pi')}[\bar{Q}(s_{t+1}, \mu)] - \bar{V}^{\star}(s_t) \\
&= \gamma \, \mathbb{E}_{p(s_{t+1}|s_t, \pi')}[\bar{Q}(s_{t+1}, \mu)] - \gamma \, \mathbb{E}_{p(s_{t+1}|s_t, \bar{\pi}^{\star})}[\bar{V}^{\star}(s_{t+1})],
\end{aligned}
$$

where the second and fourth equalities are due to $s_t \in \mathcal{S}_{\text{safe}}$, and the third equality is due to $s_t \in \mathcal{S}_{\mathcal{G}}$. Notice also, since $s_t \in \mathcal{S}_{\text{safe}}$, we have

$$
\gamma \, \mathbb{E}_{p(s_{t+1}|s_t, \pi')}[\bar{V}^{\star}(s_{t+1})] = \bar{Q}^{\star}(s_t, \pi') \geq \bar{Q}^{\star}(s_t, \bar{\pi}^{\star}) = \gamma \, \mathbb{E}_{p(s_{t+1}|s_t, \bar{\pi}^{\star})}[\bar{V}^{\star}(s_{t+1})].
$$

Therefore, combining the two inequalities above, we have

$$
\mathbb{E}_{p(s_{t+1}|s_t, \pi')}[\bar{V}^{\star}(s_{t+1})] \geq \mathbb{E}_{p(s_{t+1}|s_t, \pi')}[\bar{Q}(s_{t+1}, \mu)].
$$

Since $\bar{Q}(s, a) \geq \bar{V}^{\star}(s)$ on $\mathcal{S} \times \mathcal{A}$, by the same argument we made for s_0, we conclude $\bar{Q}(s_{t+1}, \mu) = \bar{V}^{\star}(s_{t+1})$ with probability one. Therefore, the agent stays in the subset $\mathcal{S}_{\mathcal{G}}$.

With this property in mind, let $s \sim d^{\pi'}$. Then the following holds for all $a \in \mathcal{A}$:

$$
\begin{aligned}
\bar{A}(s, a) &= \bar{Q}(s, a) - \bar{Q}(s, \mu) \\
&= \bar{Q}(s, a) - \bar{Q}^{\star}(s, \bar{\pi}^{\star}) \\
&\geq \bar{Q}^{\star}(s, a) - \bar{Q}^{\star}(s, \bar{\pi}^{\star}) = \bar{A}^{\star}(s, a),
\end{aligned}
$$

where the second equality is due to $\bar{Q}(s, \mu) = \bar{V}^{\star}(s) = \bar{Q}^{\star}(s, \bar{\pi}^{\star})$ on $\mathcal{S}_{\mathcal{G}}$. □

Proposition 4. *Let $\bar{\pi}^\star$ be an optimal policy for $\bar{\mathcal{M}}$, $\bar{Q}^\star$ be its state-action value function, and $\bar{V}^\star$ be its state value function. Let $G_0 = \{(\bar{Q}, \mu, 0) : (\bar{Q}, \mu, 0)$ is admissible, $\bar{Q}(d_0, \mu) = \bar{V}^\star(d_0)\}$. Let $\mathcal{G}^\star = (\bar{Q}^\star, \bar{\pi}^\star, 0) \in G_0$. Consider arbitrary $\mathcal{G} \in G_0$ and policy π. Let $\tilde{\mathcal{M}}$ and $\tilde{\mathcal{M}}^\star$ be the absorbing MDPs induced by $\mathcal{G}$ and $\mathcal{G}^\star$, respectively, and let $\tilde{d}^\pi$ and $\tilde{d}^{\star,\pi}$ be their respective state-action distributions. Then,*

$$\mathrm{Supp}_{\mathcal{S}\times\mathcal{A}}(\tilde{d}^\pi) \subseteq \mathrm{Supp}_{\mathcal{S}\times\mathcal{A}}(\tilde{d}^{\star,\pi}),$$

where $\mathrm{Supp}_{\mathcal{S}\times\mathcal{A}}(d)$ denotes the support of a distribution d when restricted on $\mathcal{S} \times \mathcal{A}$.

Proof. Let $\tau = (s_0, a_0, s_1, a_1, \dots)$ be any trajectory that has non-zero probabilty in the trajectory distribution $\tilde{\rho}^\pi$ of π on $\tilde{\mathcal{M}}$. Let $\mathcal{I}$ and $\mathcal{I}^\star$ be the intervention sets of $\mathcal{G}$ and $\mathcal{G}^\star$, respectively. Suppose for some t that $(s_t, a_t) \in \mathcal{I}$. We know for $t' \geq t+1$ that $s_{t'} = s_\dagger$. In addition, by Lemma 6, we have $\bar{A}^\star(s_{t'}, a_{t'}) \leq \bar{A}(s_{t'}, a_{t'}) \leq 0$ for any $t' \in \{0, \dots, t-1\}$, so $(s_{t'}, a_{t'}) \notin \mathcal{I}^\star$. Therefore, the sub-trajectory $(s_{t'}, a_{t'})$ with $t' \in \{0, \dots, t\}$ also has a non-zero probability in $\tilde{\mathcal{M}}^\star$. By this argument, every sub-trajectory in $\mathcal{S} \times \mathcal{A}$ with non-zero probability in $\tilde{\mathcal{M}}$ also has non-zero probability in $\tilde{\mathcal{M}}^\star$. The final thesis follows from defining the state-action distributions through averaging the trajectory distributions. □

B.1.4 Proof for Absorbing MDP in Section 3.3.3

We derive some properties of the Bellman operator of the absorbing MDP.

Lemma 7. *For a policy π, let $(\mathcal{B}^\pi Q)(s,a) \triangleq r(s,a) + \gamma\, \mathbb{E}_{p(s'|s,a)}[Q(s', \pi)]$ denote the Bellman operator of π in $\mathcal{M}$; similarly define $\tilde{\mathcal{B}}^\pi$ for $\tilde{\mathcal{M}}$. Let $Q : \tilde{\mathcal{S}} \times \mathcal{A} \to \mathbb{R}$ be some function satisfying $Q(s_\dagger, a) = 0$ for all $a \in \mathcal{A}$.*

1. *The Bellman operator in $\tilde{\mathcal{M}}$ can be written as*

$$(\tilde{\mathcal{B}}^\pi Q)(s,a) = \begin{cases} (\mathcal{B}^\pi Q)(s,a) \cdot \mathbf{1}\{(s,a) \notin \mathcal{I}\} - c_\dagger \cdot \mathbf{1}\{(s,a) \in \mathcal{I}\}, & (s,a) \in \mathcal{S} \times \mathcal{A} \\ 0, & s = s_\dagger. \end{cases} \tag{B.3}$$

2. *The following holds when the temporal-difference operator $\tilde{\mathcal{D}}^\pi$ for $\tilde{\mathcal{M}}$ is applied to the policy's state-action value function Q^π for $\mathcal{M}$:*

$$\left(-c_\dagger - \frac{1}{1-\gamma}\right) \mathbf{1}\{(s,a) \in \mathcal{I}\} \leq (\tilde{\mathcal{D}}^\pi Q^\pi)(s,a) \leq -c_\dagger \, \mathbf{1}\{(s,a) \in \mathcal{I}\} \quad \textit{for all } (s,a) \in \mathcal{S} \times \mathcal{A} \tag{B.4}$$

$$(\tilde{\mathcal{D}}^\pi Q^\pi)(s_\dagger, a) = 0, \tag{B.5}$$

where the definition of Q^π is extended to $s_\dagger$ as $Q^\pi(s_\dagger, a) = 0$.

Proof. For brevity, let $\Omega(s,a) = \mathbf{1}\{(s,a) \in \mathcal{I}\}$.

1. Since $Q(s_\dagger, \pi) = 0$, the following holds for any $(s,a) \in \mathcal{S} \times \mathcal{A}$:

$$\begin{aligned} (\tilde{\mathcal{B}}^\pi Q)(s,a) &= \tilde{r}(s,a) + \gamma \, \mathbb{E}_{p(s'|s,a)}[Q(s',\pi)] \\ &= (1 - \Omega(s,a))\big(r(s,a) + \gamma \, \mathbb{E}_{p(s'|s,a)}[Q(s',\pi)]\big) - \Omega(s,a) \cdot c_\dagger \\ &= (1 - \Omega(s,a)) \cdot (\mathcal{B}^\pi Q)(s,a) - \Omega(s,a) \cdot c_\dagger \end{aligned}$$

and

$$(\tilde{\mathcal{B}}^\pi Q)(s_\dagger, a) = 0 + \gamma Q(s_\dagger, \pi) = 0.$$

2. For Eq. (B.4), using the fact that $\mathcal{B}^\pi Q^\pi = Q^\pi$, the following applies on $\mathcal{S} \times \mathcal{A}$:

$$(\tilde{\mathcal{D}}^\pi Q^\pi)(s,a) = (\tilde{\mathcal{B}}^\pi Q^\pi)(s,a) - Q^\pi(s,a) = \Omega(s,a) \cdot (-c_\dagger - Q^\pi(s,a)).$$

Since $0 \le Q^\pi(s,a) \le \frac{1}{1-\gamma}$, we have

$$\left(-c_\dagger - \frac{1}{1-\gamma}\right)\Omega(s,a) \le (\tilde{\mathcal{D}}^\pi Q^\pi)(s,a) \le -c_\dagger\,\Omega(s,a).$$

For the absorbing state in Eq. (B.5), by the extended definition and the equality $(\tilde{\mathcal{B}}^\pi Q)(s_\dagger, a) = 0$, we have

$$(\tilde{\mathcal{D}}^\pi Q^\pi)(s_\dagger, a) = (\tilde{\mathcal{B}}^\pi Q^\pi)(s_\dagger, a) - Q^\pi(s_\dagger, a) = 0.$$

□

Lemma 8. *For any policy π, $P_{\mathcal{G}}(\pi) = \frac{1}{1-\gamma}\,\mathbb{E}_{\tilde{d}^\pi(s,a)}[\mathbf{1}\{(s,a) \in \mathcal{I}\}]$.*

Proof. Notice that for any h,

$$\begin{aligned}\mathbb{P}_{\rho^\pi(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing) &= \mathbb{P}_{\tilde{\rho}^\pi(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing) \\ &= \mathbb{E}_{\tilde{\rho}^\pi(\tau)}\left[\sum_{t=0}^{h-1} \mathbf{1}\{(s_t,a_t) \in \mathcal{I}\}\right].\end{aligned}$$

By Lemma 2,

$$\begin{aligned}\frac{1}{1-\gamma}\,\mathbb{E}_{\tilde{d}^\pi(s,a)}[\mathbf{1}\{(s,a) \in \mathcal{I}\}] &= \mathbb{E}_{\tilde{\rho}^\pi(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t\,\mathbf{1}\{(s_t,a_t) \in \mathcal{I}\}\right] \\ &= (1-\gamma)\sum_{h=0}^{\infty}\gamma^h\,\mathbb{E}_{\tilde{\rho}^\pi(\tau)}\left[\sum_{t=0}^{h-1} \mathbf{1}\{(s_t,a_t) \in \mathcal{I}\}\right] \\ &= (1-\gamma)\sum_{h=0}^{\infty}\gamma^h\,\mathbb{P}_{\rho^\pi(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing) \\ &= P_{\mathcal{G}}(\pi).\end{aligned}$$

□

Using the above results, we can bound the difference between the values of the original and the absorbing MDPs.

Lemma 1. *For every policy π, it holds that*

$$c_\dagger P_{\mathcal{G}}(\pi) \leq V^\pi(d_0) - \tilde{V}^\pi(d_0) \leq \left(c_\dagger + \frac{1}{1-\gamma}\right) P_{\mathcal{G}}(\pi).$$

Proof. First, extend the definition of Q^π to $s_\dagger$ as $Q^\pi(s_\dagger, a) = 0$ for any $a \in \mathcal{A}$. From Corollary 1, we have

$$\tilde{V}^\pi(d_0) - V^\pi(d_0) = \frac{1}{1-\gamma} \mathbb{E}_{\tilde{d}^\pi(s,a)}[(\tilde{\mathcal{D}}^\pi Q^\pi)(s,a)]$$

From Lemma 7, we can derive

$$\left(-c_\dagger - \frac{1}{1-\gamma}\right) \frac{\mathbb{E}_{\tilde{d}^\pi(s,a)}[\mathbf{1}\{(s,a) \in \mathcal{I}\}]}{1-\gamma} \leq \tilde{V}^\pi(d_0) - V^\pi(d_0) \leq -c_\dagger \frac{\mathbb{E}_{\tilde{d}^\pi(s,a)}[\mathbf{1}\{(s,a) \in \mathcal{I}\}]}{1-\gamma}.$$

Finally, substituting the equality from Lemma 8 and negating the inequality concludes the proof. □

Next, we derive some lemmas, which will be later to used to show that when the intervention set is partial, the unconstrained reduction is effective.

Lemma 9. *Let $\mathcal{I} \subset \mathcal{S}_{\text{safe}} \times \mathcal{A}$ be partial, and let $\mathcal{F} = (\mathcal{S}_{\text{safe}} \times \mathcal{A}) \setminus \mathcal{I}$ be the state-action pairs that are not intervened. For an arbitrary policy π, define*

$$\pi_f(a|s) \triangleq \pi(a|s)\, \mathbf{1}\{(s,a) \in \mathcal{F}\} + f(s,a), \tag{B.6}$$

where $f(s,a)$ is some arbitrary non-negative function which is zero on $\mathcal{I}$ and that ensures

$\sum_{a\in\mathcal{A}} \pi_f(a|s) = 1$ *for all* $s \in \mathcal{S}$. *Define*

$$\tilde{J}_+^\pi \triangleq \frac{1}{1-\gamma}\,\mathbb{E}_{\tilde{d}^\pi(s,a)}[r(s,a)\cdot \mathbf{1}\{(s,a)\in\mathcal{F}\}]$$
$$\tilde{J}_-^\pi \triangleq \frac{1}{1-\gamma}\,\mathbb{E}_{\tilde{d}^\pi(s,a)}[-c_\dagger\cdot \mathbf{1}\{(s,a)\in\mathcal{I}\}]$$

as the expected returns in $\mathcal{F}$ *and* $\mathcal{I}$*, respectively.*

The following are true:

1. $\tilde{V}^\pi(d_0) = \tilde{J}_+^\pi + \tilde{J}_-^\pi$.
2. $\tilde{d}^{\pi_f}(s,a) \geq \tilde{d}^\pi(s,a)$ *for all* $(s,a)\in\mathcal{F}$.
3. $\tilde{J}_+^{\pi_f} \geq \tilde{J}_+^\pi$.
4. $\mathbb{E}_{\tilde{d}^{\pi_f}(s,a)}[\mathbf{1}\{(s,a)\in\mathcal{I}\}] = 0$, *implying* $\tilde{J}_-^{\pi_f} = 0$.
5. $\tilde{V}^{\pi_f}(d_0) \geq \tilde{V}^\pi(d_0)$ *whenever* $c_\dagger \geq 0$. *Furthermore, if* $c_\dagger > 0$ *and* $\pi(a|s) > 0$ *for some* $(s,a)\in\mathcal{I}$*, then* $\tilde{V}^{\pi_f}(d_0) > \tilde{V}^\pi(d_0)$.

Proof. 1. This follows from the definition of $\tilde{r}$ in Eq. (7.7).

2. Recall $\tilde{d}^\pi(s,a) = (1-\gamma)\sum_{t=0}^\infty \gamma^t \tilde{d}_t^\pi(s,a)$. To show the desired result, we show by induction that $\tilde{d}_t^{\pi_f}(s,a) \geq \tilde{d}_t^\pi(s,a)$ for all $(s,a)\in\mathcal{F}$ and $t\geq 0$. For $t=0$, by construction of π_f, we have $\pi_f(a|s)\geq\pi(a|s)$ for all $(s,a)\in\mathcal{F}$ and therefore $\tilde{d}_0^{\pi_f}(s,a)\geq\tilde{d}_0^\pi(s,a)$ for all $(s,a)\in\mathcal{F}$.

   Now suppose that for some $t\geq 0$ the inequality $\tilde{d}_t^{\pi_f}(s,a)\geq\tilde{d}_t^\pi(s,a)$ holds for all

$(s, a) \in \mathcal{F}$. Then, for some $(s, a) \in \mathcal{F}$, we can derive

$$\begin{aligned}
\tilde{d}^{\pi_f}_{t+1}(s, a) &= \pi_f(a|s) \sum_{(s_t, a_t) \in \mathcal{S} \times \mathcal{A}} \tilde{p}(s|s_t, a_t) \tilde{d}^{\pi_f}_t(s_t, a_t) \\
&= \pi_f(a|s) \sum_{(s_t, a_t) \in \mathcal{F}} p(s|s_t, a_t) \tilde{d}^{\pi_f}_t(s_t, a_t) \\
&\geq \pi(a|s) \sum_{(s_t, a_t) \in \mathcal{F}} p(s|s_t, a_t) \tilde{d}^{\pi}_t(s_t, a_t) \\
&= \tilde{d}^{\pi}_{t+1}(s, a),
\end{aligned}$$

where we use the inductive hypothesis in the inequality. Thus, we have $\tilde{d}^{\pi_f}(s, a) \geq \tilde{d}^{\pi}(s, a)$ by summing over each time step.

3. From statement 2, definition of $\tilde{J}^{\pi}_{+}$, and non-negativity of the reward r, it follows that $\tilde{J}^{\pi_f}_{+} \geq \tilde{J}^{\pi}_{+}$.

4. This statement from the construction of π_f and induction. First, we have $\tilde{d}^{\pi_f}_0(s, a) = 0$ for all $(s, a) \in \mathcal{I}$. Now suppose for some $t \geq 0$ that $\tilde{d}^{\pi_f}_t(s, a) = 0$ for all $(s, a) \in \mathcal{I}$. We can see that $\tilde{d}^{\pi_f}_{t+1}(s, a) = 0$ for all $(s, a) \in \mathcal{I}$ since π_f never chooses actions such that $(s, a) \in \mathcal{I}$.

   Therefore, $\tilde{d}^{\pi_f}(s, a) = 0$ for all $(s, a) \in \mathcal{I}$. By definition of $\tilde{J}^{\pi}_{-}$, this allows us to conclude that $\tilde{J}^{\pi}_{-} = 0$.

5. Using statements 3 and 4, we conclude that

$$\tilde{V}^{\pi_f}(d_0) = \tilde{J}^{\pi_f}_{+} + \tilde{J}^{\pi_f}_{-} \geq \tilde{J}^{\pi}_{+} + \tilde{J}^{\pi}_{-} = \tilde{V}^{\pi}(d_0).$$

   The special case follows from observing that $\tilde{J}^{\pi}_{-} < 0$ whenever $\pi(a|s) > 0$ for some $(s, a) \in \mathcal{I}$.

□

Lemma 10. *Let $c_\dagger$ be non-negative and $\tilde{V}^\star$ denote the optimal value function for $\tilde{\mathcal{M}}$.*

1. *For any policy π,*

$$\tilde{V}^\star(d_0) \geq \tilde{J}_+^\pi.$$

2. *There is an optimal policy $\tilde{\pi}^\star$ of $\tilde{\mathcal{M}}$ satisfying*

$$\mathbb{E}_{\tilde{d}^{\tilde{\pi}^\star}(s,a)}[\mathbf{1}\{(s,a) \in \mathcal{I}\}] = 0. \tag{B.7}$$

3. *If $c_\dagger$ is positive, every optimal policy of $\tilde{\mathcal{M}}$ satisfies Eq.* (B.7)*.*

Proof. 1. Let the policy π be arbitrary, and define π_f using Eq. (B.6). The following then holds by Lemma 9:

$$\tilde{V}^\star(d_0) \geq \tilde{V}^{\pi_f}(d_0) = \tilde{J}_+^{\pi_f} \geq \tilde{J}_+^\pi.$$

2. Suppose that π is an optimal policy of $\tilde{\mathcal{M}}$, and define π_f using Eq. (B.6). Because $c_\dagger$ is non-negative, we know by Lemma 9 and optimality of π that $\tilde{V}^{\pi_f}(d_0) = \tilde{V}^{\pi}(d_0)$. Therefore, we can define an optimal policy as $\tilde{\pi}^\star = \pi_f$ and conclude by Lemma 9 that $\mathbb{E}_{\tilde{d}^{\tilde{\pi}^\star}(s,a)}[\mathbf{1}\{(s,a) \in \mathcal{I}\}] = 0$.

3. Suppose for the sake of contradiction there is an optimal policy $\tilde{\pi}^\star$ of $\tilde{\mathcal{M}}$ such that Eq. (B.7) does *not* hold (i.e., it may take some $(s,a) \in \mathcal{I}$). By Lemma 9, we can construct some policy π_f such that $\tilde{V}^{\pi_f}(d_0) > \tilde{V}^{\tilde{\pi}^\star}(d_0)$. This contradicts $\tilde{\pi}^\star$ being optimal, so every optimal policy of $\tilde{\mathcal{M}}$ must satisfy Eq. (B.7).

□

These results show that if the intervention set is partial and the penalty of being intervened is strict, then the optimal policy of the absorbing MDP would not be intervened.

Proposition 6. *If $c_\dagger$ is positive and $\mathcal{G}$ induces a partial $\mathcal{I}$, then every optimal policy $\tilde{\pi}^\star$ of $\tilde{\mathcal{M}}$ satisfies $P_{\mathcal{G}}(\tilde{\pi}^\star) = 0$.*

Proof. This directly follows from Lemmas 8 and 10. □

Below we derive some lemmas to show a near optimal policy of the absorbing MDP is safe. (We already proved above that the optimal policy of the absorbing MDP is safe).

Lemma 11. *Let $\mathcal{I} \subset \mathcal{S} \times \mathcal{A}$ be partial (Definition 2). Given some policy π, let π' be the corresponding shielded policy defined in Eq.* (7.4)*. Then, the following holds for any $h \geq 0$ in $\mathcal{M}$:*

$$\mathbb{P}_{\rho^{\pi}(\tau)}(s_{\triangleright} \in \tau^h) \leq \mathbb{P}_{\rho^{\pi'}(\tau)}(s_{\triangleright} \in \tau^h) + \mathbb{P}_{\rho^{\pi}(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing), \tag{B.8}$$

where $\tau^h = (s_0, a_0, \ldots, s_{h-1}, a_{h-1})$ is an h-step trajectory segment.

Proof. First, we notice that $\pi'(a|s) \geq \pi(a|s)$ when $(s, a) \notin \mathcal{I}$, because $\pi'(a|s) = \pi(a|s) + w(s)\mu(a|s) \geq \pi(a|s)$.

We bound the probability of π violating a constraint in $\mathcal{M}$ by introducing whether π visits the intervention set:

$$\begin{aligned}\mathbb{P}_{\rho^{\pi}(\tau)}(s_{\triangleright} \in \tau^h) &= \mathbb{P}_{\rho^{\pi}(\tau)}(s_{\triangleright} \in \tau^h,\ \tau^h \cap \mathcal{I} = \varnothing) + \mathbb{P}_{\rho^{\pi}(\tau)}(s_{\triangleright} \in \tau^h,\ \tau^h \cap \mathcal{I} \neq \varnothing) \\ &\leq \mathbb{P}_{\rho^{\pi}(\tau)}(s_{\triangleright} \in \tau^h,\ \tau^h \cap \mathcal{I} = \varnothing) + \mathbb{P}_{\rho^{\pi}(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing).\end{aligned}$$

We now bound the first term. Let τ^h satisfy the event "$s_{\triangleright} \in \tau^h,\ \tau^h \cap \mathcal{I} = \varnothing$", and let T be the time index such that $s_T = s_{\triangleright}$ in τ^h. Then, the probability of this trajectory under π and $\mathcal{M}$ is

$$d_0(s_0)\pi(a_0|s_0)p(s_1|s_0, a_0) \cdots \pi(a_{T-1}|s_{T-1})p(s_T|s_{T-1}, a_{T-1}).$$

Since each (s_t, a_t) is not in $\mathcal{I}$, we have $\pi(a_t|s_t) \leq \pi'(a_t|s_t)$ for each (s_t, a_t) in τ^h. Thus, the probability of this trajectory under π and $\mathcal{M}$ is upper bounded by its probability under π' and $\mathcal{M}$. Summing over each trajectory τ^h satisfying the event then yields:

$$\mathbb{P}_{\rho^{\pi}(\tau)}(s_{\triangleright} \in \tau^h,\ \tau^h \cap \mathcal{I} = \varnothing) \leq \mathbb{P}_{\rho^{\pi'}(\tau)}(s_{\triangleright} \in \tau^h,\ \tau^h \cap \mathcal{I} = \varnothing).$$

We now complete the original bound:

$$\begin{aligned}\mathbb{P}_{\rho^{\pi}(\tau)}(s_{\triangleright} \in \tau^h) &\leq \mathbb{P}_{\rho^{\pi}(\tau)}(s_{\triangleright} \in \tau^h,\ \tau^h \cap \mathcal{I} = \varnothing) + \mathbb{P}_{\rho^{\pi}(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing) \\ &\leq \mathbb{P}_{\rho^{\pi'}(\tau)}(s_{\triangleright} \in \tau^h,\ \tau^h \cap \mathcal{I} = \varnothing) + \mathbb{P}_{\rho^{\pi}(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing) \\ &\leq \mathbb{P}_{\rho^{\pi'}(\tau)}(s_{\triangleright} \in \tau^h) + \mathbb{P}_{\rho^{\pi}(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing).\end{aligned}$$

□

Lemma 12. *For any policy π and $\mathcal{I} \subset \mathcal{S} \times \mathcal{A}$ that is partial, let π' be the corresponding shielded policy. Then, the following safety bound holds:*

$$\bar{V}^{\pi}(d_0) \leq \bar{V}^{\pi'}(d_0) + \frac{1}{1-\gamma}\,\mathbb{E}_{\tilde{d}^{\pi}(s,a)}[\mathbf{1}\{(s,a) \in \mathcal{I}\}].$$

Proof. Using Eq. (B.8) from Lemma 11 and the fact that the probabilities can be expressed as expected sums of indicators:

$$\begin{aligned}\mathbb{P}_{\rho^{\pi}(\tau)}(s_{\triangleright} \in \tau^h) &= \mathbb{E}_{\rho^{\pi}(\tau)}\left[\sum_{t=0}^{h-1} \mathbf{1}\{s_t = s_{\triangleright}\}\right] \\ \mathbb{P}_{\rho^{\pi'}(\tau)}(s_{\triangleright} \in \tau^h) &= \mathbb{E}_{\rho^{\pi'}(\tau)}\left[\sum_{t=0}^{h-1} \mathbf{1}\{s_t = s_{\triangleright}\}\right] \\ \mathbb{P}_{\rho^{\pi}(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing) &= \mathbb{E}_{\tilde{\rho}^{\pi}(\tau)}\left[\sum_{t=0}^{h-1} \mathbf{1}\{(s_t, a_t) \in \mathcal{I}\}\right].\end{aligned}$$

Then, applying Lemma 2 results in the desired inequality. □

Proposition 7 (Suboptimality in $\tilde{\mathcal{M}}$ to Suboptimality and Safety in $\mathcal{M}$)**.** *Let $c_{\dagger}$ be positive. For some policy π, let π' be the shielded policy defined in Eq.* (7.4)*. Suppose π is ε-suboptimal for $\tilde{\mathcal{M}}$. Then, for any comparator policy $\pi^{\star}$, the following performance and*

safety guarantees hold for π in $\mathcal{M}$:

$$V^{\pi^\star}(d_0) - V^{\pi}(d_0) \leq \left(c_\dagger + \frac{1}{1-\gamma}\right) P_{\mathcal{G}}(\pi^\star) + \varepsilon$$
$$\bar{V}^{\pi}(d_0) \leq \bar{V}^{\pi'}(d_0) + \frac{\varepsilon}{c_\dagger}.$$

Proof. The performance bound follows from Lemma 1.

$$\begin{aligned}
V^{\pi^\star}(d_0) - V^{\pi}(d_0) &= V^{\pi^\star}(d_0) - \tilde{V}^{\pi^\star}(d_0) + \tilde{V}^{\pi^\star}(d_0) - \tilde{V}^{\pi}(d_0) + \tilde{V}^{\pi}(d_0) - V^{\pi}(d_0) \\
&\leq \left(c_\dagger + \frac{1}{1-\gamma}\right) P_{\mathcal{G}}(\pi^\star) + \tilde{V}^{\pi^\star}(d_0) - \tilde{V}^{\pi}(d_0) - c_\dagger\, P_{\mathcal{G}}(\pi) \\
&\leq \left(c_\dagger + \frac{1}{1-\gamma}\right) P_{\mathcal{G}}(\pi^\star) + \tilde{V}^{*}(d_0) - \tilde{V}^{\pi}(d_0) \\
&\leq \left(c_\dagger + \frac{1}{1-\gamma}\right) P_{\mathcal{G}}(\pi^\star) + \varepsilon.
\end{aligned}$$

For the safety bound, we start with Lemma 12:

$$\bar{V}^{\pi}(d_0) \leq \bar{V}^{\pi'}(d_0) + \frac{1}{1-\gamma}\, \mathbb{E}_{\tilde{d}^{\pi}(s,a)}[\mathbf{1}\{(s,a) \in \mathcal{I}\}]$$

We provide an upper bound on the second term on the right hand side above. Using the definition of $\tilde{J}^{\pi}_{-}$ in Lemma 9, we derive that

$$\begin{aligned}
\frac{\mathbb{E}_{\tilde{d}^{\pi}(s,a)}[\mathbf{1}\{(s,a) \in \mathcal{I}\}]}{1-\gamma} &= -\frac{\tilde{J}^{\pi}_{-}}{c_\dagger} \\
&= \frac{1}{c_\dagger}\Big(-\tilde{V}^{\pi}(d_0) + \tilde{V}^{\star}(d_0) + \tilde{J}^{\pi}_{+} - \tilde{V}^{\star}(d_0)\Big) \\
&\leq \frac{1}{c_\dagger}\Big(\tilde{V}^{\star}(d_0) - \tilde{V}^{\pi}(d_0)\Big) = \frac{\varepsilon}{c_\dagger},
\end{aligned}$$

where the inequality is due to Lemma 10.

Combine everything altogether:

$$\begin{aligned}\bar{V}^{\pi}(d_0) &\leq \bar{V}^{\pi'}(d_0) + \frac{\mathbb{E}_{\tilde{d}^{\pi}(s,a)}[\mathbf{1}\{(s,a) \in \mathcal{I}\}]}{1-\gamma} \\ &= \bar{V}^{\pi'}(d_0) + \frac{\varepsilon}{c_\dagger}.\end{aligned}$$

□

We now prove the main result of the chapter.

Theorem 1 (Performance and Safety Guarantee at Deployment)**.** *Let $c_\dagger = 1$ and $\mathcal{G}$ be σ-admissible. If $\hat{\pi}$ is an ε-suboptimal policy for $\tilde{\mathcal{M}}$, then, for any comparator policy $\pi^\star$, the following performance and safety guarantees hold for $\hat{\pi}$ in $\mathcal{M}$:*

$$\begin{aligned}V^{\pi^\star}(d_0) - V^{\hat{\pi}}(d_0) &\leq \frac{2}{1-\gamma} P_{\mathcal{G}}(\pi^\star) + \varepsilon \\ \bar{V}^{\hat{\pi}}(d_0) &\leq \bar{Q}(d_0, \mu) + \frac{\min\{\sigma + \eta, 2\gamma\}}{1-\gamma} + \varepsilon,\end{aligned}$$

where $P_{\mathcal{G}}(\pi^\star) \triangleq (1-\gamma)\sum_{h=0}^{\infty} \gamma^h \, \mathbb{P}_{\rho^{\pi^\star}(\tau)}(\tau^h \cap \mathcal{I} \neq \varnothing)$ is the probability that $\pi^\star$ visits $\mathcal{I}$ in $\mathcal{M}$.

Proof. This is a direct result of Proposition 7.

The performance suboptimality results from:

$$\begin{aligned}V^{\pi^\star}(d_0) - V^{\hat{\pi}}(d_0) &\leq \left(c_\dagger + \frac{1}{1-\gamma}\right) P_{\mathcal{G}}(\pi^\star) + \varepsilon \\ &\leq \left(1 + \frac{1}{1-\gamma}\right) P_{\mathcal{G}}(\pi^\star) + \varepsilon \\ &= \frac{2-\gamma}{1-\gamma} P_{\mathcal{G}}(\pi^\star) + \varepsilon \\ &\leq \frac{2}{1-\gamma} P_{\mathcal{G}}(\pi^\star) + \varepsilon.\end{aligned}$$

For the safety bound,

$$\begin{aligned}\bar{V}^{\hat{\pi}}(d_0) &\leq \bar{V}^{\mathcal{G}(\hat{\pi})}(d_0) + \varepsilon \\ &\leq \bar{Q}(d_0, \mu) + \frac{\min\{\sigma + \eta, 2\gamma\}}{1-\gamma} + \varepsilon,\end{aligned}$$

where the second inequality follows from Theorem 2 and ε-suboptimality of $\hat{\pi}$ in $\tilde{\mathcal{M}}$. □

B.2 Additional Discussion of SAILR

B.2.1 Necessity of the Partial Property

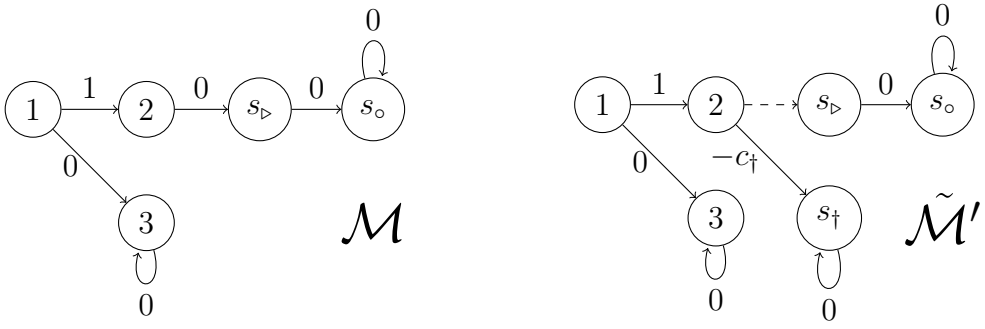


Figure B.1: A simple example illustrating a non-partial intervention. Edge weights correspond to rewards. If $c_\dagger < 1/\gamma$, the optimal policy in $\tilde{\mathcal{M}}'$ will always go into the intervention set.

We highlight that the subset $\mathcal{I}$ being *partial* (Definition 1) is crucial for the unconstrained MDP reduction behind SAILR. If we were to construct an absorbing MDP $\tilde{\mathcal{M}}'$ described in Section 7.3.2 using an arbitrary non-partial subset $\mathcal{I}' \subseteq \mathcal{S} \times \mathcal{A}$, then the optimal policy of $\tilde{\mathcal{M}}'$ can still enter $\mathcal{I}'$ for some $c_\dagger \geq 0$, because the optimal policy of $\tilde{\mathcal{M}}'$ can use earlier rewards to make up for the penalty incurred in $\mathcal{I}'$.

To see this, consider the toy MDP $\mathcal{M}$ shown in Fig. B.1. Since there is no alternative action available at state 2, the intervention illustrated in $\tilde{\mathcal{M}}'$ is *not* partial. Suppose $c_\dagger < 1/\gamma$. Then, in $\tilde{\mathcal{M}}'$, a policy choosing to transition from 1 to 3 has a value of 0, and a policy choosing to transition from 1 to 2 has a value of $1 - \gamma c_\dagger > 0$. Therefore, the optimal policy

will transition from 1 to 2 and go into the non-partial intervention set $\mathcal{I}'$. Once applied to the original MDP $\mathcal{M}$, this policy will always go into the unsafe set.

One might think that generally it is possible to set $c_\dagger$ to be large enough to ensure the optimal policy will never go into the intervention set, which is indeed true for the counterexample above. But we remark that we need to set $c_\dagger$ to be arbitrarily large for general problems, which can cause high variance issues in return or gradient estimation (Shalev-Shwartz et al., 2016). Because of the discount factor $\gamma < 1$, the reward stemming from the absorbing state will be at most $-\gamma^T c_\dagger$, where T is the time step that the system enters $\mathcal{I}'$. For a fixed and finite $c_\dagger$, we can then extend the above MDP construction to let the agent go through a long enough chain after transitioning from 1 to 2 so that the resultant value satisfies $1 - \gamma^T c_\dagger > 0$. Like the example above, this path would be the only path with positive reward, despite intersecting the intervention set. Therefore, the optimal policy of $\tilde{\mathcal{M}}'$ will enter $\mathcal{I}'$.

B.2.2 Bias of SAILR

In Theorem 1, we give a performance guarantee of SAILR

$$V^\star(d_0) - V^{\hat{\pi}}(d_0) \leq \frac{2}{1-\gamma} P_{\mathcal{G}}(\pi^\star) + \varepsilon.$$

It shows that SAILR has a bias $P_{\mathcal{G}}(\pi^\star) \in [0, 1]$, which is the probability that the optimal policy $\pi^\star$ would be intervened by the advantage-based intervention rule. Here we discuss special cases where this bias vanishes.

The first special case is when the original problem is unconstrained (i.e., Eq. (7.2) has a trivial constraint with $\delta = 1$). In this case, we can set the threshold $\eta \geq \gamma$ in SAILR to turn off the intervention, and SAILR returns the optimal policy of the MDP $\mathcal{M}$ when the base RL algorithm can find one.

Another case is when $\pi^\star$ is a perfect safe policy, i.e., $\bar{V}^{\pi^\star}(d_0) = 0$ and we run SAILR

with the intervention rule $\mathcal{G}^\star = (\bar{Q}^\star, \bar{\pi}^\star, 0)$ (Proposition 4). Similar to the proof of Lemma 6, one can show that running $\pi^\star$ would not trigger the intervention rule $\mathcal{G}^\star$ and therefore the bias $P_{\mathcal{G}^\star}(\pi^\star)$ is zero.

However, we note that generally the bias $P_{\mathcal{G}}(\pi^\star)$ can be non-zero.

B.3 Experimental Details

B.3.1 Point Robot

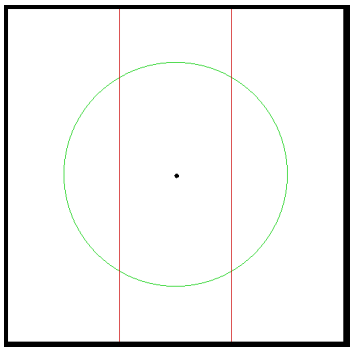


Figure B.2: The point environment. The black dot corresponds to the agent, the green circle to the desired path, and the red lines to the constraints on the horizontal position. The vertical constraints are outside of the visualized environment.

This environment (Fig. B.2) is a simplification of the point environment proposed by Achiam et al. (2017). The state is $s = (x, y, \dot{x}, \dot{y})$, where (x, y) is the x-y position and $(\dot{x}, \dot{y})$ is the corresponding velocity. The action $a = (a_x, a_y)$ is the force applied to the robot (each component has maximum magnitude $a_{\max}$). The agent has some mass m and can achieve maximum speed $v_{\max}$. The dynamics update (with time increment Δt) is:

$$(x_{t+1}, y_{t+1}) = (x_t, y_t) + (\dot{x}_t, \dot{y}_t)\Delta t + \frac{1}{2m} a_t \Delta t^2$$
$$(\dot{x}_{t+1}, \dot{y}_{t+1}) = \texttt{clip-norm}\left((\dot{x}_t, \dot{y}_t) + \frac{1}{m} a_t \Delta t, v_{\max} \right),$$

where $\texttt{clip-norm}(u, c)$ scales u so that its norm matches c if $\|u\| > c$. The reward corresponds to following a circular path of radius $R^\star$ at a high speed and the safe set to staying

within desired positional bounds $x_{\max}$ and $y_{\max}$:

$$r(s,a) = \frac{(\dot{x},\dot{y}) \cdot (-y,x)}{1 + |\|(x,y)\| - R^\star|}$$
$$\mathcal{S}_{\text{safe}} = \{s \in \mathcal{S} : |x| \leq x_{\max} \text{ and } |y| \leq y_{\max}\}$$

For our experiments, we set these parameters to $m = 1$, $v_{\max} = 2$, $a_{\max} = 1$, $\Delta t = 0.1$, $R^\star = 5$, $x_{\max} = 2.5$, and $y_{\max} = 15$.

For this environment, we also consider a *shaped* cost function $\hat{c}(s,a)$ which is a function of the *distance* of the state s to the boundary of the unsafe set, denoted by $\text{dist}(s, \mathcal{S}_{\text{unsafe}})$. Here, $\mathcal{S}_{\text{unsafe}}$ denotes the 2D unsafe region in this environment (i.e., those outside the vertical lines in Fig. B.2). Note that in the theoretical analysis $\mathcal{S}_{\text{unsafe}}$ is abstracted into $\{s_\triangleright, s_\dagger\}$.

For the point environment, the distance function is $\text{dist}(s, S_{\text{unsafe}}) = \max\{0, \min\{x_{\max} - x, x_{\max} + x, y_{\max} - y, y_{\max} + y\}\}$. For some constant $\alpha \geq 0$, the cost function is defined as a hinge function of the distance:

$$\hat{c}(s,a) = \begin{cases} \mathbf{1}\{\text{dist}(s, \mathcal{S}_{\text{unsafe}}) = 0\}, & \alpha = 0 \\ \max\{0, 1 - \frac{1}{\alpha}\text{dist}(s, \mathcal{S}_{\text{unsafe}})\}, & \text{otherwise.} \end{cases} \tag{B.9}$$

We note that $\hat{c}$ is an upper bound for c if $\alpha > 0$ and $\hat{c} = c$ if $\alpha = 0$. We shape the cost here to make it continuous, so that the effects of approximation bias is smaller than that resulting from a discontinuous cost (i.e., the original indicator function).

Intervention Rule: The backup policy μ applies a decelerating force (with component-wise magnitude up to $a_{\max}$) until the agent has zero velocity. Our experiments consider the following approaches to construct $\bar{Q}$:

- **Neural network approximation**: We construct a dataset of points mapping states and actions to state-action values $\bar{Q}^\mu$ by picking some state and action in the MDP, executing the action from that state, and then continuing the rollout with the backup

policy μ to find the empirical state-action value with respect to the shaped cost function $\hat{c}$. Our dataset consists of 10^7 points resulting from a uniform discretization of the state-action space. We apply a similar method to form a dataset for the state values $\bar{V}^{\mu}$.

We then train four networks (two to independently approximate $\bar{Q}^{\mu}$, and two for $\bar{V}^{\mu}$), where each network has three hidden layers each with 256 neurons and a ReLU activation. The predicted advantage is $\bar{A}(s,a) = \max\{\bar{Q}_1(s,a), \bar{Q}_2(s,a)\} - \min\{\bar{V}_1(s), \bar{V}_2(s)\}$, where we apply the pessimistic approach from Thananjeyan et al. (2021) to prevent overestimation bias.

- **Model-based evaluation**: Here, we have access to a model of the robot where all parameters match the real environment except possibly the mass $\hat{m}$. We refer to the modeled transition dynamics as $\hat{p}$ and the resulting trajectory distribution under μ as $\hat{\rho}^{\mu}$. The function $\bar{Q}$ is then set to be the model-based estimate of $\bar{Q}^{\mu}$ using the shaped cost function $\hat{c}$ and dynamics $\hat{p}$:

$$\bar{Q}(s,a) = \mathbb{E}_{\hat{\rho}^{\mu}(\tau|s_0=s,a_0=a)}\left[\sum_{t=0}^{\infty} \gamma^t \hat{c}(s_t,a_t)\right].$$

  For our experiments, the modeled mass $\hat{m}$ is either 1 (unbiased case) or 0.5 (biased case).

For our experiments, we set the advantage threshold $\eta = 0.08$ when using the neural network approximator and $\eta = 0$ when using the model-based rollouts.

Hyperparameters: All point experiments were run on a 32-core Threadripper machine. The given hyperparameters were found by hand-tuning until good performance was found on all algorithms.

Hyperparameter	Value
Epochs	500
Neural network architecture	2 hidden layers, 64 neurons per hidden layer, $\tanh$ act.
Batch size	4000
Discount γ	0.99
Entropy bonus	0.001
CMDP threshold δ	0.01
Penalty value $c_\dagger$	2
Lagrange multiplier step size (for constrained approaches)	0.05
Cost shaping constant α	0.5
Number of seeds	10

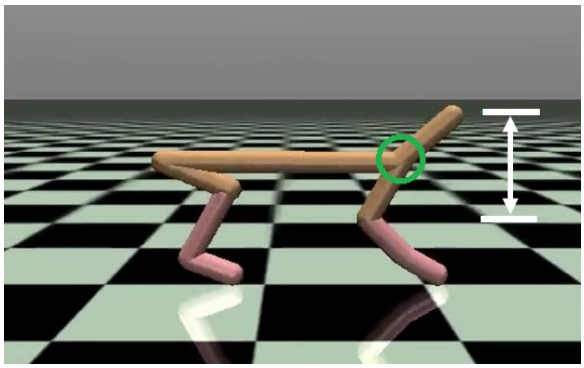


Figure B.3: The half-cheetah environment. The green circle is centered on the link of interest, and the white double-headed arrow denotes the allowed height range of the link.

B.3.2 Half-Cheetah

This environment (Fig. B.3) comes from OpenAI Gym and has reward equal to the agent's forward velocity. One of the agent's links (denoted by the green circle in Fig. B.3) is constrained to lie in a given height range, outside of which the robot is deemed to have fallen over. In other words, if h is the height of the link of interest, $h_{\min}$ is the minimum height, and $h_{\max}$ is the maximum height, the safe set is defined as $\mathcal{S}_{\text{safe}} = \{s \in \mathcal{S} : h_{\min} \leq h \leq h_{\max}\}$. For our experiments, we set $h_{\min} = 0.4$ and $h_{\max} = 1$.

Heuristic Intervention Rule: This intervention rule $\mathcal{G} = (\bar{Q}, \mu, \eta)$ relies on a dynamics model (here, unbiased) to greedily predict whether the safety constraint would be violated at the next time step. In particular, if s is the current state and $\hat{a} \sim \pi(\cdot|s)$ is the proposed action, the agent will be intervened if the height $\hat{h}'$ in the next state $\hat{s}' \sim p(\cdot|s, \hat{a})$ lies outside the range $[\hat{h}_{\min}, \hat{h}_{\max}]$, where $\hat{h}_{\min}$ and $\hat{h}_{\max}$ can be set to a smaller range than

$[h_{\min}, h_{\max}]$ to induce a more conservative intervention. Once intervened, the episode terminates. The reason for using a smaller range $[\hat{h}_{\min}, \hat{h}_{\max}]$ is an attempt to make the intervention rule possess the partial property (see the discussion in Section 7.3.1). If we were to set the range to be the ordinary range $[h_{\min}, h_{\max}]$ that defines the safe subset, the penalty $c_\dagger$ would need to be very negative, which would destabilize learning. Furthermore, there is no guarantee that the intervention set for the original range is partial since there may be no available action to keep the agent from being intervened.

MPC-Based Intervention Rule: Similarly with the model-based intervention rule for the point environment, the MPC intervention rule $\mathcal{G} = (\bar{Q}, \mu, \eta)$ uses a model of the half-cheetah. The backup policy μ is a sampling-based model predictive control (MPC) algorithm based on (Williams, Wagener, et al., 2017). The MPC algorithm has an optimization horizon of $H = 16$ time steps and minimizes the cost function corresponding to an indicator function of the link height being in the range $[0.45, 0.95]$.[1] The function $\bar{Q}$ is defined as:

$$\bar{Q}(s,a) = \mathbb{E}_{\hat{\rho}(\tau)}\left[\sum_{t=0}^{H} \gamma^t \hat{c}(\hat{s}_t, \hat{a}_t) \;\middle|\; \hat{s}_0 = s, \hat{a}_0 = a, \hat{a}_{1:H} = \mathrm{MPC}(\hat{s}_1)\right],$$

where $\hat{c}(s,a)$ is the hinge-shaped cost function (in Eq. (B.9)) corresponding to the distance function $\mathrm{dist}(s, S_{\text{unsafe}}) = \max\{0, \min\{h - h_{\min}, h_{\max} - h\}\}$.

For our experiments, we set the advantage threshold $\eta = 0.2$. We also use a modeled mass of 14 (unbiased) and 12 (biased) in our experiments.

Hyperparameters: Except for the MPC-based intervention, all half-cheetah experiments were run on a 32-core Threadripper machine. The MPC-based intervention experiments were run on 64-core Azure servers with each run taking 24 hours. The given hyperparameters were found by hand-tuning until good performance was found on all algorithms.

[1] Observe that this is slightly smaller than the $[0.4, 1]$ height range of the original safety constraint.

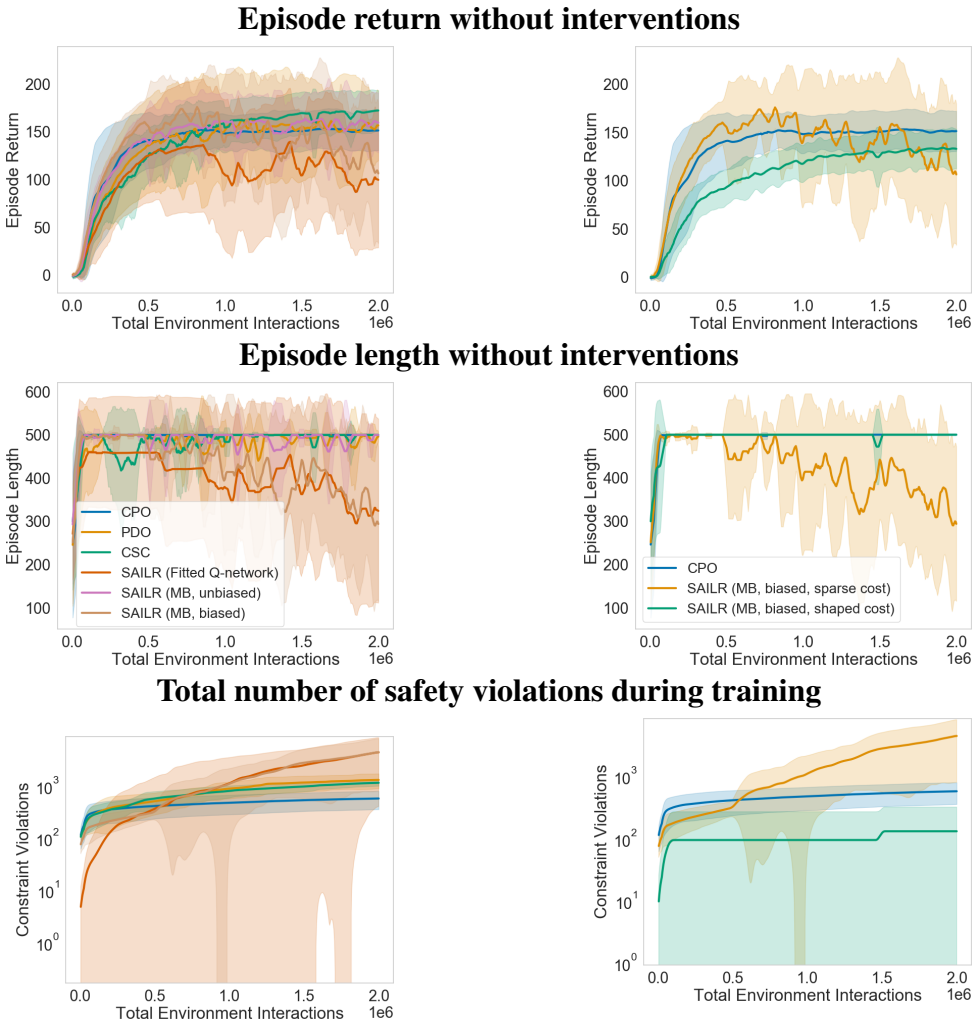
Hyperparameter	Value
Epochs	1250
Neural network architecture	2 hidden layers, 64 neurons per hidden layer, $\tanh$ act.
Batch size	4000
Discount γ	0.99
Entropy bonus	0.01
CMDP threshold δ	0.01
Penalty value $c_\dagger$	0.1
Lagrange multiplier step size (for constrained approaches)	0.05
Heuristic intervention range $[\hat{h}_{\min}, \hat{h}_{\max}]$	$[0.4, 0.9]$
Cost shaping constant α	0.05
Number of seeds	8

B.4 Ablations for Point Robot

We run the following two ablations for the point environment, with results shown in Fig. B.4:

1. We additionally run all the algorithms with the original sparse cost (Fig. B.4a). Here, the baseline algorithms as expected yield high deployment returns while violating many constraints during training. For SAILR, however, only the model-based instance with an unbiased model is able to satisfy the desiderata of high deployment returns while being safe during training. In this case, the sparse cost along with the approximation errors from the other two instances result in the slack σ being large for admissibilty, meaning the safety bounds in Theorems 1 and 2 are loose.

2. We run the model-based instance of SAILR with a biased model and either the sparse cost or the shaped cost (Fig. B.4b). Using the sparse cost with the biased model for intervention has deleterious effects in safety and performance. The model mismatch causes a compounding number of safety violations in training (bottom plot) and destabilizes the policy optimization, as observed in the deteriorated returns (top plot) and safety (middle plot), respectively. Shaping the cost function for intervention results in far fewer safety violations and stabilizes the policy optimization.



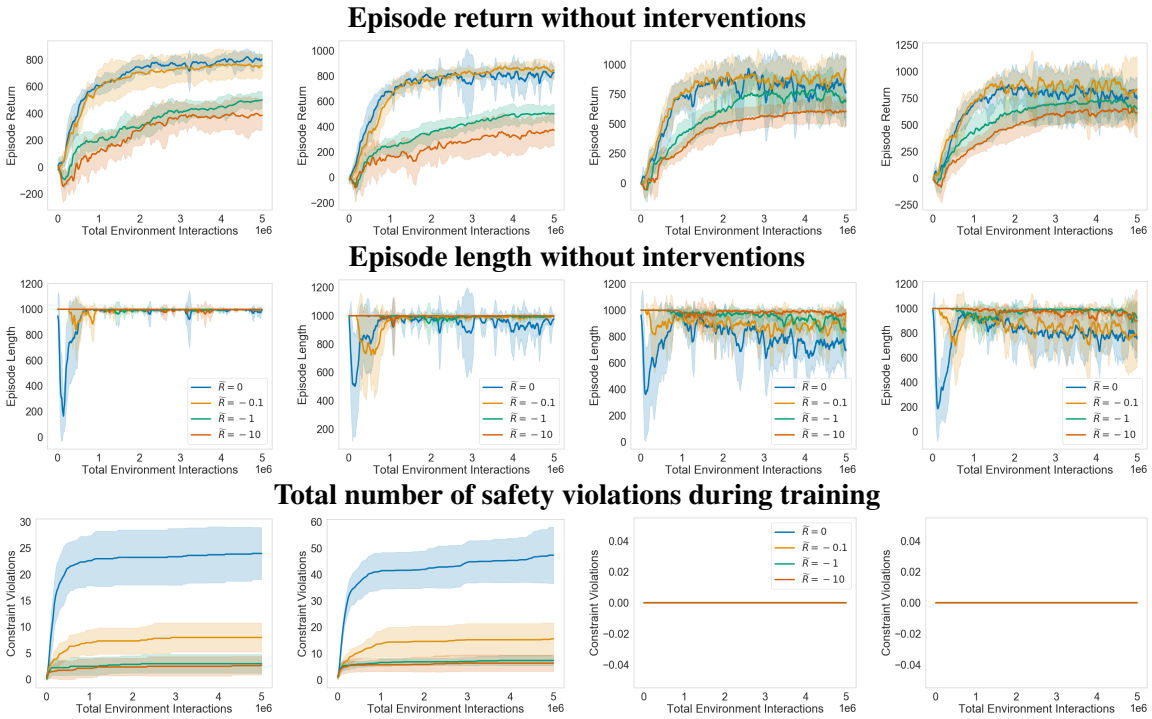
(a) Sparse cost

(b) Biased model, either a sparse cost or shaped cost

Figure B.4: Ablations for point robot experiment

B.5 Varying Intervention Penalty for Half-Cheetah

We vary the intervention penalty $c_\dagger$ for both the MPC-based intervention and heuristic intervention (Fig. B.5). Common among all results is that the deployment episode return (top row) decreases and deployment safety (middle row) increases with the magnitude of the penalty, consistent with the performance and safety bounds in Proposition 7. Furthermore, early in training, we remark that larger penalties result in the agent learning to be safe more quickly (middle row). This is likely because the large penalties prioritize the agent to not



(a) MPC with unbiased model

(b) MPC with biased model

(c) Heuristic with smaller height range

(d) Heuristic with original height range

Figure B.5: Varying intervention penalty for half-cheetah experiment. Here, $c_\dagger = -\tilde{R}$.

be intervened, which allows it to more quickly learn to be as safe as the backup policy μ.

For training-time safety with the MPC backup policy (bottom row of Figs. B.5a and B.5b), we observe that there are more violations as the penalty decreases, likely because the agent is less conservative during rollouts.

For the heuristic intervention (Figs. B.5c and B.5d), we surmise that neither heuristic is partial since we require $c_\dagger$ to be relatively large in order for the agent to learn to be safe (middle row). This is in constrast with the MPC-based intervention rule (middle row of Figs. B.5a and B.5b), where the penalty only needs to be nonzero, which indicates that the MPC-based intervention is partial.

APPENDIX C

MoCapAct: A MULTI-TASK DATASET FOR SIMULATED HUMANOID CONTROL

This appendix provides the following sections:

- Documentation of the dataset (Section C.1)
- Training details (Section C.2)
- Extra results (Section C.3)

C.1 Dataset Documentation

C.1.1 Clip Snippet Experts

We signify a clip snippet expert by the snippet it is tracking. We denote a snippet by the clip ID, its start step, and its end step. For example, `CMU_006_12-151-336` is the snippet corresponding to the clip `CMU_006_12` with start step 151 and end step 336. Taking `CMU_006_12-151-336` as an example expert, the file hierarchy for the snippet expert is:

```
CMU_006_12-151-336
├── clip_info.json ..... Contains clip ID, start step, and end step.
└── eval_rsi/model
    ├── best_model.zip ..... Contains policy parameters and hyperparameters.
    └── vecnormalize.pkl ..... Used to get normalizer for observation and reward.
```

The expert policy can be loaded using Stable-Baselines3's functionality.

C.1.2 Expert Rollouts

The expert rollouts consist of a collection of HDF5 files, one file per clip. An HDF5 file contains expert rollouts for each constituent snippet as well as miscellaneous information and statistics. To facilitate efficient loading of the observations, we concatenate all the proprioceptive observations (joint angles, joint velocities, actuator activations, etc.) from an episode into a single numerical array and provide indices for the constituent observations in the `observable_indices` group.

Taking `CMU_009_12.hdf5` (which contains three snippets) as an example, we have the following HDF5 hierarchy:

```
CMU_009_12.hdf5
├── n_rsi_rollouts ..... R, number of rollouts from random time steps in snippet.
├── n_start_rollouts ..... S, number of rollouts from start of snippet.
├── ref_steps . Indices of MoCap reference relative to current time step. Here, (1, 2, 3, 4, 5).
├── observable_indices
│   └── walker
│       ├── actuator_activation ..... (0, 1, ..., 54, 55)
│       ├── appendages_pos ..... (56, 57, ..., 69, 70)
│       ├── body_height ..... (71)
│       ├── ⋮
│       └── world_zaxis ..... (2865, 2866, 2867)
│
└── stats ..... Statistics computed over the entire dataset.
    ├── act_mean ..... Mean of the experts' sampled actions.
    ├── act_var ..... Variance of the experts' sampled actions.
    └── mean_act_mean ..... Mean of the experts' mean actions.
```

- mean_act_var ..... Variance of the experts' mean actions.
- proprio_mean ..... Mean of the proprioceptive observations.
- proprio_var ..... Variance of the proprioceptive observations.
- count ..... Number of observations in dataset.

- CMU_009_12-0-198 ..... Rollouts for the snippet CMU_009_12-0-198.
- CMU_009_12-165-363 ..... Rollouts for the snippet CMU_009_12-165-363.
- CMU_009_12-330-529 ..... Rollouts for the snippet CMU_009_12-330-529.

Each snippet group contains $R+S$ rollouts. The first S episodes correspond to episodes intialized from the start of the snippet and the last R episodes to episodes initialized at random points in the snippet. We now uncollapse the CMU_009_12-0-198 group within the HDF5 file to reveal the rollout structure:

CMU_009_12-0-198
- early_termination $(R+S)$-boolean array indicating which episodes terminated early.

- rsi_metrics ..... Metrics for episodes that initialize at random points in snippet.
  - episode_returns ..... R-array of episode returns.
  - episode_lengths ..... R-array of episode lengths.
  - norm_episode_returns ..... R-array of normalized episode rewards.
  - norm_episode_lengths ..... R-array of normalized episode lengths.

- start_metrics ..... Metrics for episodes that initialize at start in snippet.

- 0 ..... First episode, of length T.
  - observations
    - proprioceptive ..... $(T+1)$-array of proprioceptive observations.

- `walker/body_camera` $(T+1)$-array of images from body camera (not included).
- `walker/egocentric_camera...` $(T+1)$-array of images from egocentric camera (not included).

- `actions` ..... T-array of sampled actions executed in environment.
- `mean_actions` ..... T-array of corresponding mean actions.
- `rewards` ..... T-array of rewards from environment.
- `values` ..... T-array computed using the policy's value network.
- `advantages` ..... T-array computed using generalized advantage estimation.

- 1 ..... Second episode.
- ⋮
- $R+S-1$ ..... $(R+S)^{\text{th}}$ episode.

To keep the dataset size manageable, we do *not* include image observations in the dataset. We do provide code to log them when rolling out the experts for generating the dataset.

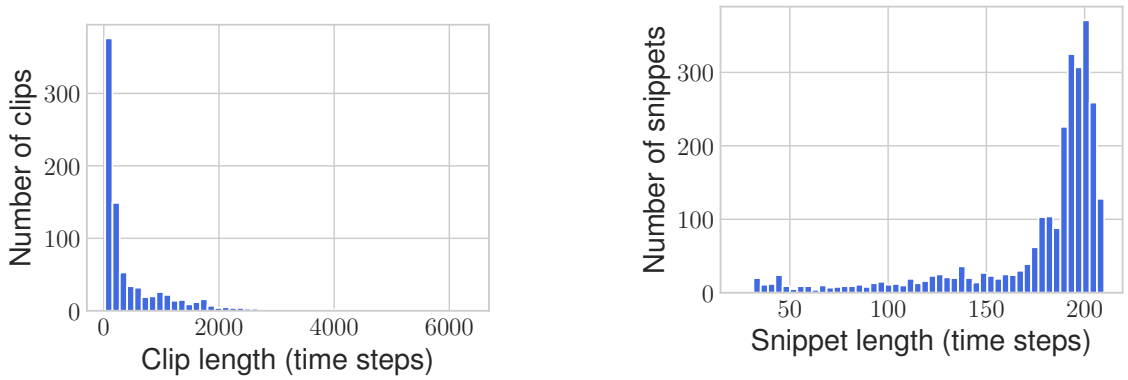
C.1.3 Hosting Plan

The link to the dataset can be found on the project website (`microsoft.github.io/MoCapAct`). We provide a "large" rollout dataset where $R = S = 100$ with size 600 GB and a "small" rollout dataset where $R = S = 10$ with size 50 GB. The dataset website also includes the policies we trained in Section 8.5, i.e., the multi-clip tracking policies, RL-trained task policies, and the GPT policy. We also provide a Python script to download individual experts and rollouts from the dataset.

C.2 Training Details

C.2.1 Clip Snippet Experts

MoCap Snippets



(a) Lengths of the MoCap clips.

(b) Lengths of the snippets generated from the clips.

Figure C.1: Lengths of clips and snippets.

The MoCap dataset has a wide spread in clip length (Fig. C.1a), with the longest clip being 6371 time steps (191 seconds). Training clip experts to track long clips is potentially slow and laborious, so we follow Merel, Hasenclever, et al. (2019) by dividing clips longer than 210 time steps (6.3 seconds) into short snippets. In particular, we divide the clip into uniformly-sized snippets with an overlap of 33 time steps (1 second) such that the longest snippet has at most 210 time steps. This yields a snippet dataset with a much tighter range of snippet lengths (Fig. C.1b). We do *not* divide the clips from the "Get Up" subset of the MoCap dataset since they contain involved motions of getting up from the ground.

Expert Training Details

We use the Stable-Baselines3 (Raffin et al., 2021) implementation of PPO (Schulman, Wolski, et al., 2017) to optimize each expert. Each expert is a neural network with three hidden layers, 1024 neurons in each hidden layer, and the $\tanh$ activation. At the start of each

Table C.1: Hyperparameters for clip snippet expert training.

Total environment steps	150 million
Environment steps per policy update	8192
PPO epochs	10
PPO minibatch size	512
PPO clipping parameter ε	0.25
GAE parameter λ	0.95
Discount factor γ	0.95
Gradient norm clipping value	1
Adam step size	1e−5 for first $50\mathrm{M}$ env. steps 6e−6 for next $50\mathrm{M}$ env. steps 3e−6 for last $50\mathrm{M}$ env. steps

episode, we randomly select a time step from the corresponding clip snippet (excluding the last 10 time steps from the snippet) and initialize the humanoid to match the clip features at the corresponding step. We evaluate the policy every 1 million environment steps using 1000 episodes under the same initialization scheme (but now excluding the last 30 time steps of the snippet) and the same action noise of 0.1 as for training rollouts. We also end the training if the average normalized episode length is at least 0.98 and the average normalized episode reward does not improve by more than 1% from the current best reward after 10 million environment steps. We normalize the observation and reward using running statistics from the environment. We give the other relevant hyperparameters in Table C.1. For details of the reward function and early termination of an episode, we refer the reader to the appendix of Hasenclever et al. (2020).

We ran the training on a mix of Azure Standard_H8 (8 CPUs), Standard_H16 (16 CPUs), Standard_NC6s_v2 (6 CPUs and 1 P100 GPU), and Standard_ND6s (6 CPUs and 1 P40 GPU) VMs.

The observables for the clip expert are: `joints_pos`, `joints_vel`, `sensors_velocimeter`, `sensors_gyro`, `end_effectors_pos`, `world_zaxis`, `actuator_activation`, `sensors_touch`, `sensors_torque`, `time_in_clip`.

C.2.2 Multi-Clip Tracking Policy

Table C.2: Hyperparameters for multi-clip tracking policy training.

Adam step size	5e–4
Minibatch sequence length T	30
Minibatch size	256
Gradient norm clipping value	1
KL divergence weight β	0.1
Autoregressive parameter α	0
Weighting temperature λ	CWR: 0.2 AWR: 8 RWR: 4

We train the multi-clip policy $\pi(a_t, z_t|s_t, s_t^{\text{ref}}, z_{t-1}) = \pi_{\text{enc}}(z_t|s_t, s_t^{\text{ref}}, z_{t-1})\pi_{\text{dec}}(a_t|s_t, z_t)$ by optimizing the following imitation objective:

$$\mathbb{E}_{\substack{(s_{1:T}, s_{1:T}^{\text{ref}}, a_{1:T}, c)\sim\mathcal{D}, \\ z_{0:T}\sim\pi_{\text{enc}}}}\left[\sum_{t=1}^{T}\left[w_c(s_t, a_t)\log\pi_{\text{dec}}(a_t|s_t, z_t) - \beta\ \text{KL}(\pi_{\text{enc}}(z_t|s_t, s_t^{\text{ref}}, z_{t-1})\,\|\,p(z_t|z_{t-1}))\right]\right],$$

where $p(z_t|z_{t-1}) = \mathcal{N}(z_t; \alpha z_{t-1}, (1-\alpha^2)I)$ for some $\alpha \in [0, 1]$. We do this (for each data point in a minibatch) by sampling $z_0 \sim \mathcal{N}(0, I)$, sampling a T-step data sequence (of humanoid states $s_{1:T}$, MoCap references $s_{1:T}^{\text{ref}}$, and expert's mean actions $\bar{a}_{1:T}$) from the dataset $\mathcal{D}$, unrolling the recurrent policy through the sampled sequence, performing backpropagation through time on the objective function, and finally updating the network using the Adam optimizer (Kingma and Ba, 2015). To speed up training, we normalize the humanoid state s_t and MoCap reference s_t^{ref} using the corresponding mean and standard deviation computed over the entire dataset. For the weighted schemes, we multiply the weight w_c by a constant that ensures the average data weight is 1 so that the KL regularization term maintains the same relative weight. For all schemes, we also sample data from shorter clips at a higher rate to ensure the rollout data from the clips is uniformly even. This gives about 1% improvement in policy evaluation compared to vanilla sampling.

We use PyTorch Lightning (Falcon, 2019) to train the multi-clip policy. The encoder and decoder are both neural networks with 1024 neurons per hidden layer and use layer norm and the ELU activation. The encoder has two hidden layers, while the decoder has

three hidden layers. We ran the training on Azure Standard_ND24s VMs, each equipped with 24 CPUs and 4 P40 GPUs. We periodically evaluate the multi-clip policy by running 1000 episodes on the set of MoCap snippets following the same reference state initialization scheme as in the rest of the paper. We found we only need to train the policy for about 50 000 steps (about 10% of an epoch) before plateauing on policy evaluation (Section C.3.2). We give the other relevant hyperparameters in Table C.2.

The observables for the policy are:

- Encoder: `joints_pos`, `joints_vel`, `sensors_velocimeter`, `sensors_gyro`, `end_effectors_pos`, `world_zaxis`, `actuator_activation`, `sensors_touch`, `sensors_torque`, `body_height`, `reference_rel_bodies_pos_local`, `reference_rel_bodies_quats`
- Decoder: `joints_pos`, `joints_vel`, `sensors_velocimeter`, `sensors_gyro`, `end_effectors_pos`, `world_zaxis`, `actuator_activation`, `sensors_touch`, `sensors_torque`

C.2.3 Transfer for Reinforcement Learning

Go-to-Target Task

This task matches that of Hasenclever et al. (2020), which we refer the reader to for details.

Velocity Control

In this task, a target speed $s^* \in [0, 4.5]$ and direction $\psi^* \in [0, 2\pi)$ are randomly sampled every 10 seconds. Defining the target velocity as $v_t^* = (s^* \cos\psi^*, s^* \sin\psi^*)$ and the humanoid's current velocity as v_t, the reward is defined as the product of a speed factor and direction factor:

$$r_t = \exp\left(-\left(\frac{\|v_t\| - \|v_t^*\|}{\eta}\right)^2\right)\left(\frac{1 + \text{score}(v_t, v_t^*)}{2}\right)^k,$$

where $\mathrm{score}(v_t, v_t^*) = v_t \cdot v_t^* / \|v_t\| \|v_t^*\|$ gives the cosine of the angle between the two velocity vectors. In our experiments, we set $\eta = 0.75$ and $k = 7$. We also experimented with the velocity error reward used by Bohez, Tunyasuvunakool, et al. (2022) but found that our reward was easier to optimize. We terminate the episode either after 2000 time steps (60 seconds) or if any body part other than the feet touches the ground.

Hyperparameters

Table C.3: Hyperparameters for RL transfer tasks.

Total environment steps	150 million
Environment steps per policy update	16 384
PPO epochs	10
PPO minibatch size	1024
PPO clipping parameter ε	0.2
KL divergence threshold for early stopping	0.3
Entropy bonus coefficient	General low-level policy: 1e−4 Locomotion low-level policy: 1e−3 No low-level policy: 1e−4
GAE parameter λ	0.95
Discount factor γ	0.99
Gradient norm clipping value	1
Adam step size	5e−5
Number of actors	32
Initial standard deviation for task policy	With low-level policy: 2.5 Without low-level policy: 0.5
Maximum per-element action magnitude for task policy	With low-level policy: 3 Without low-level policy: 1

Like the snippet experts, we train the task policies using the Stable-Baselines3 implementation of PPO. Each task policy is a neural network with three hidden layers, 1024 neurons per hidden layer, and the $\tanh$ activation. We ran the training on Azure Standard_ND6s (6 CPUs and 1 NVIDIA P40 GPU) VMs. We give other hyperparameters in Table C.3.

Table C.4: Hyperparameters for GPT training.

Adam step size	3e−6
Minibatch size	256
Gradient norm clipping value	1
Attention dropout probability	0.1
Embedding dropout probability	0.1
Residual dropout probability	0.1
Block size	32
Embedding size	768
Attention heads	8
Number of layers	8
Weight decay	0.1

C.2.4 Motion Completion with GPT

We train a variant of minGPT (Karpathy, 2020) that we adapted to accept continuous inputs and output continuous actions. This particular model has 57 million parameters and was trained with a context length of 32 time steps, corresponding to roughly one second of motion. Similar to the multi-clip policy (Section C.2.2), we sample state $s_{(t-31):t}$ and mean-action $\bar{a}_{(t-31):t}$ sequences of length 32 from the MoCapAct dataset $\mathcal{D}$. To speed up training, we normalize the humanoid state s_t using the corresponding mean and standard deviation computed over the entire dataset. We use the mean squared error loss on the sequence of predicted actions from the GPT. We trained GPT using PyTorch Lightning (Falcon, 2019) on Azure Standard_NC24s_v3, each equipped with 24 CPUs and 4 V100 GPUs, for 2 million steps, corresponding to one week of wall-clock time. We give the other relevant hyperparameters in Table C.4.

The observables for the GPT policy are: `joints_pos`, `joints_vel`, `sensors_velocimeter`, `sensors_gyro`, `end_effectors_pos`, `world_zaxis`, `actuator_activation`, `sensors_touch`, `sensors_torque`, `body_height`. Importantly, GPT is not given any reference data from the MoCap clip, so any motion generation was done only on the basis of the historical context provided.

C.3 More Results

C.3.1 Clip Snippet Experts

Training Curves

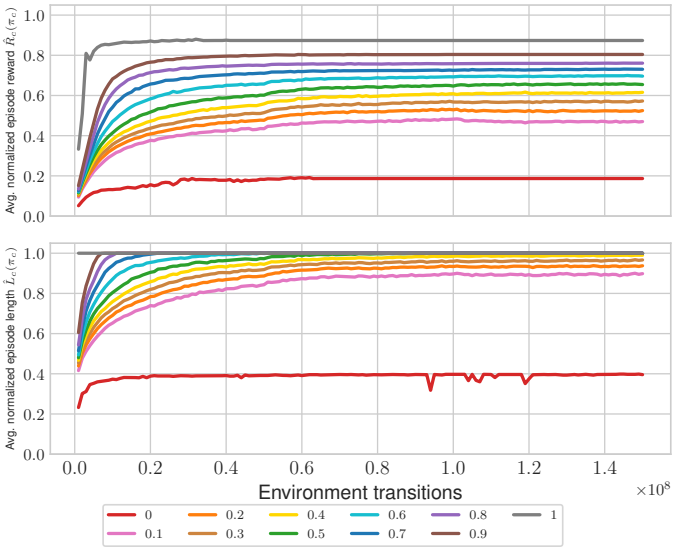


Figure C.2: Snippet expert training curves on MoCap dataset.

We give the learning curves for the experts in Fig. C.2. In particular, we plot the quantiles $0, 0.1, \ldots, 0.9, 1$ to visualize how the *distribution* of experts improves over the course of training. Overall, we see reliable improvement of the experts with convergence at about 100 million environment transitions.

Expert Performance vs. Snippet Length

Here, we study whether longer snippets are "harder" to track by the expert. Fig. C.3 shows scatter plots of the experts' normalized episode reward and length as a function of the snippet length. Overall, the snippet length does not appear to affect the experts' performance as indicated by the fitted curves being relatively flat.

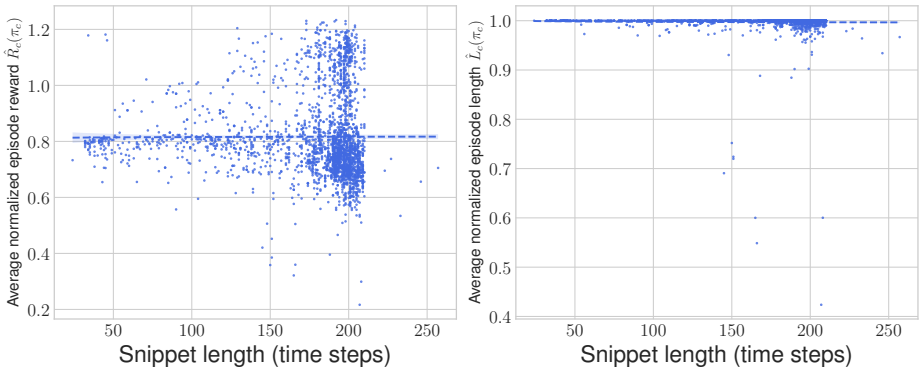
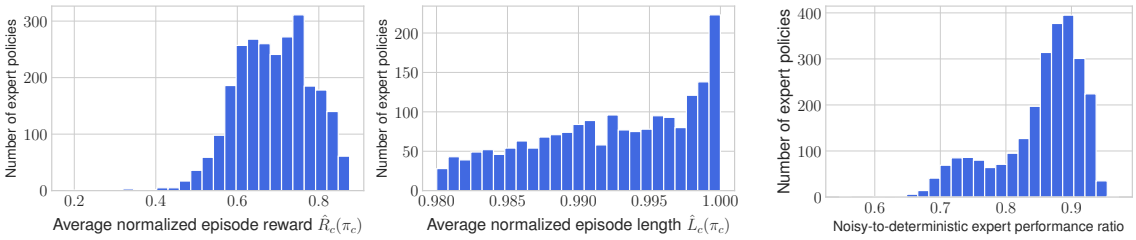


Figure C.3: Scatter plot of experts' performance versus the snippet length. Here, the Gaussian noise of the experts is disabled. The performance appears to be independent of snippet length.

Table C.5: Clip expert results on the MoCap snippets within `dm_control` using the stochastic π_c.

	Mean	Standard deviation	Median	Minimum	Maximum
Average normalized episode reward	0.689	0.092	0.690	0.179	0.876
Average normalized episode length	0.984	0.029	0.990	0.403	1.000



(a) Episode rewards and lengths of the noisy experts.

(b) Performance ratio of noisy expert to deterministic expert.

Figure C.4: Noisy expert results on the MoCap snippets within `dm_control`. The noisy experts incur a small performance drop from their deterministic counterparts.

Noisy Expert Evaluations

Because the MoCapAct dataset is formed from noisy rollouts of the experts, it is sensible to assess the performance of the experts when rolled out with noise. Table C.5 and Fig. C.4a show that the experts still have strong performance. We point out the noisy experts on average attain 85% of the performance of the deterministic experts (Fig. C.4b).

From the scatter plot of the noisy experts (Fig. C.5), we see a minor decrease in reward

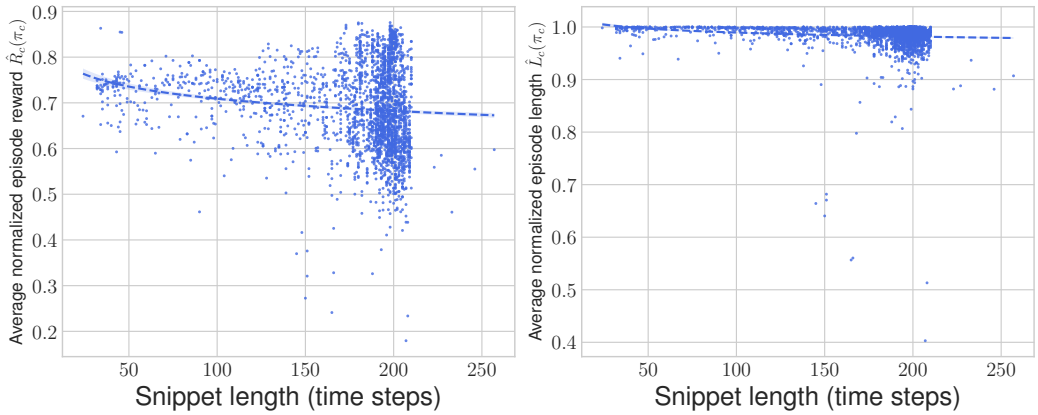


Figure C.5: Scatter plot of noisy experts' performance versus the snippet length. There is a minor decrease in performance as the snippet length increases.

and episode length as the snippet gets longer. This is probably due to longer snippets giving more time steps for the noise to destabilize the humanoid.

C.3.2 Multi-Clip Tracking Policy

Training Curves

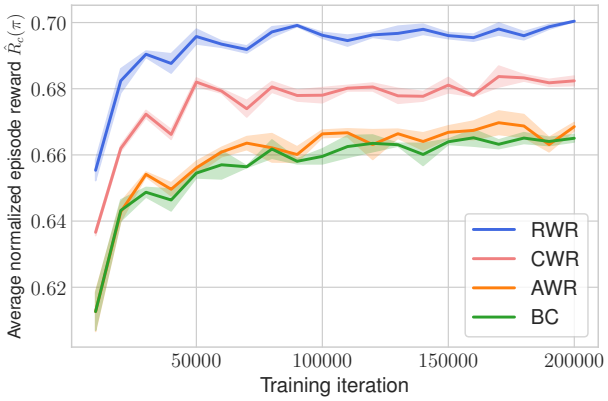


Figure C.6: Multi-clip policy training curves on MoCap snippets.

Fig. C.6 shows the reward curves for the four weighting schemes. Overall, the reward plateaus after about 50 000 iterations for each scheme, and reward-weighted regression performs markedly better than the other three schemes.

Autoregressive Parameter α

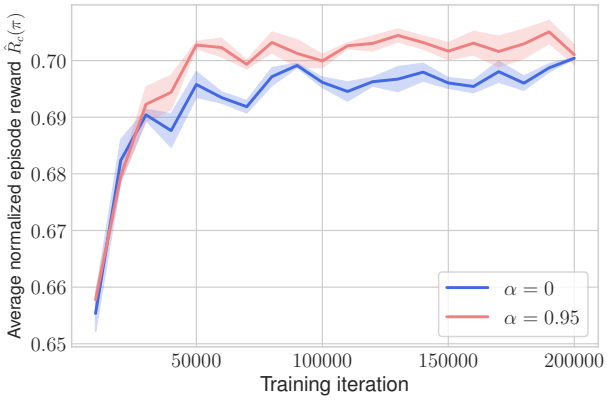


Figure C.7: Comparison of multi-clip policy's performance when varying the autoregressive parameter α for the prior distribution $p(z_t|z_{t-1})$. Here, we use the RWR-weighting scheme. Performance is broadly similar for both values of α.

Merel, Hasenclever, et al. (2019) found that using an autoregressive parameter of $\alpha =$

0.95 gave 50% improvement in policy performance over $\alpha = 0$. Interestingly, in our experiments we found that the performance gap is much smaller (Fig. C.7), with $\alpha = 0.95$ only giving 3% improvement. Accordingly, we set $\alpha = 0$ for our experiments (corresponding to a temporally independent prior of $p(z_t|z_{t-1}) = \mathcal{N}(z_t; 0, I)$) so that we could better control the size of the intentions z_t generated by our reference encoder.

Scatter Plots on Snippets and Clips

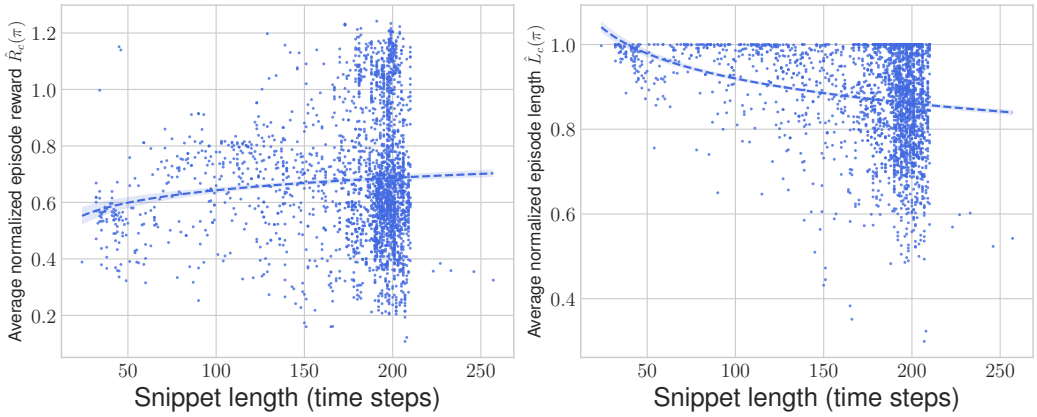


Figure C.8: Scatter plot of the multi-clip policy's performance versus the snippet length. Here, the Gaussian noise of the policy is disabled. Longer snippets tend to result in lower episode lengths.

Fig. C.8 shows the scatter plot of the multi-clip policy on all of the MoCap snippets. Compared to the noisy experts (Section C.3.1), we see a more noticeable decline in episode length on long snippets. Intuitively, this is because longer snippets allow for more opportunities for the multi-clip policy to make an episode-ending mistake. The normalized reward, on the other hand, does not give any meaningful trends.

One appealing feature of the multi-clip policy is the ability to roll out the policy on entire clips. This also allows us to discover whether the multi-clip policy has learned to "stitch" together the overlapping snippets from the dataset. Fig. C.9 shows that while there are long clips that the policy can reliably track, the overall trend is that longer clips result in lower reward and episode length. Intuitively, many clips in the MoCap dataset correspond to locomotion behaviors, which gives many opportunities for the multi-clip policy to make

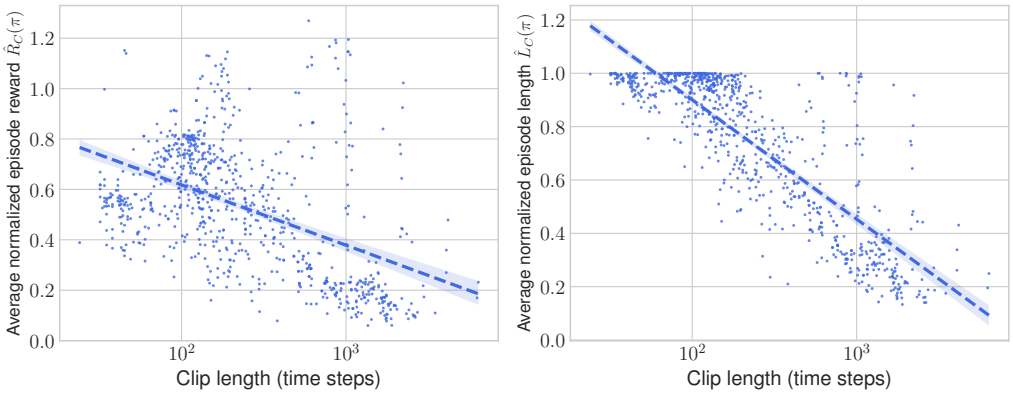


Figure C.9: Scatter plot of the multi-clip policy's performance versus the clip length. Here, the Gaussian noise of the low-level policy is disabled. Longer clips tend to result in lower episode rewards and lengths.

episode-terminating mistakes. Usually, these mistakes correspond to the humanoid legs colliding or one of the feet making bad contact with the ground, both of which cause the humanoid to fall over. The fragility on longer clips points to a shortcoming of MoCapAct: the rollouts only cover (at most) a 6-second window. Because of this, the multi-clip policy is not trained on states that would be encountered deep into a rollout (e.g., 30 seconds into a rollout), which limits the multi-clip policy's performance on many longer clips. Long clips that have high rewards and episode lengths usually have the humanoid standing for long periods of time while doing various arm motions. Here, the motions are much simpler since the humanoid merely needs to maintain balance while standing still.

REFERENCES

Abbeel, Pieter, Adam Coates, and Andrew Y. Ng (2010). "Autonomous helicopter aerobatics through apprenticeship learning". In: *The International Journal of Robotics Research* 29.13, pp. 1608–1639.

Achiam, Joshua, David Held, Aviv Tamar, and Pieter Abbeel (2017). "Constrained policy optimization". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 22–31.

Agarwal, Ananye, Ashish Kumar, Jitendra Malik, and Deepak Pathak (2023). "Legged locomotion in challenging terrains using egocentric vision". In: *Conference on Robot Learning*. PMLR, pp. 403–415.

Akimoto, Youhei, Yuichi Nagata, Isao Ono, and Shigenobu Kobayashi (2012). "Theoretical foundation for CMA-ES from information geometry perspective". In: *Algorithmica* 64.4, pp. 698–716.

Aksan, Emre, Manuel Kaufmann, Peng Cao, and Otmar Hilliges (2021). "A Spatio-Temporal Transformer for 3D Human Motion Prediction". In: *2021 International Conference on 3D Vision (3DV)*. IEEE, pp. 565–574.

Alemi, Alexander A, Ian Fischer, Joshua V Dillon, and Kevin Murphy (2017). "Deep Variational Information Bottleneck". In: *International Conference on Learning Representations*.

Alshiekh, Mohammed, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu (2018). "Safe reinforcement learning via shielding". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Altman, Eitan (1999). *Constrained Markov Decision Processes*. Vol. 7. CRC Press.

Amodei, Dario, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané (2016). "Concrete problems in AI safety". In: *arXiv preprint arXiv:1606.06565*.

Amos, Brandon, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter (2018). "Differentiable MPC for end-to-end planning and control". In: *Advances in Neural Information Processing Systems* 31.

Anderson, Greg, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri (2020). "Neurosymbolic Reinforcement Learning with Formally Verified Exploration". In: *Advances in Neural Information Processing Systems*, pp. 6172–6183.

Atkeson, Christopher G and Juan Carlos Santamaria (1997). "A comparison of direct and model-based reinforcement learning". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 4. IEEE, pp. 3557–3564.

Banerjee, Arindam, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh (2005). "Clustering with Bregman divergences". In: *Journal of Machine Learning Research* 6.Oct, pp. 1705–1749.

Beck, Amir and Marc Teboulle (2003). "Mirror descent and nonlinear projected subgradient methods for convex optimization". In: *Operations Research Letters* 31.3, pp. 167–175.

Bellman, Richard and John Casti (1971). "Differential quadrature and long-term integration". In: *Journal of Mathematical Analysis and Applications* 34.2, pp. 235–238.

Benbrahim, Hamid and Judy A Franklin (1997). "Biped dynamic walking using reinforcement learning". In: *Robotics and Autonomous Systems* 22.3-4, pp. 283–302.

Berkenkamp, Felix, Matteo Turchetta, Angela Schoellig, and Andreas Krause (2017). "Safe model-based reinforcement learning with stability guarantees". In: *Advances in Neural Information Processing Systems*, pp. 908–918.

Bertsekas, Dimitri P (2014). *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press.

— (2017). *Dynamic Programming and Optimal Control*. 4th. Vol. 1. Athena Scientific, Belmont, MA.

Bharadhwaj, Homanga, Aviral Kumar, Nicholas Rhinehart, Sergey Levine, Florian Shkurti, and Animesh Garg (2021). "Conservative Safety Critics for Exploration". In: *International Conference on Learning Representations*.

Bhardwaj, Mohak, Sanjiban Choudhury, and Byron Boots (2021). "Blending MPC & Value Function Approximation for Efficient Reinforcement Learning". In: *International Conference on Learning Representations*.

Bhardwaj, Mohak, Ankur Handa, Dieter Fox, and Byron Boots (2020). "Information theoretic model predictive Q-learning". In: *Learning for Dynamics and Control*. PMLR, pp. 840–850.

Bhardwaj, Mohak, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots (2021). "STORM: An integrated framework for fast joint-space model-predictive control for reactive manipulation". In: *Conference on Robot Learning*. PMLR, pp. 750–759.

Bishop, Christopher M (2006). *Pattern Recognition and Machine Learning*. Springer.

Bohez, Steven, Abbas Abdolmaleki, Michael Neunert, Jonas Buchli, Nicolas Heess, and Raia Hadsell (2019). "Value constrained model-free continuous control". In: *arXiv preprint arXiv:1902.04623*.

Bohez, Steven, Saran Tunyasuvunakool, Philemon Brakel, Fereshteh Sadeghi, Leonard Hasenclever, Yuval Tassa, Emilio Parisotto, Jan Humplik, Tuomas Haarnoja, Roland Hafner, Markus Wulfmeier, Michael Neunert, Ben Moran, Noah Siegel, Andrea Huber, Francesco Romano, Nathan Batchelor, Federico Casarini, Josh Merel, Raia Hadsell, and Nicolas Heess (2022). "Imitate and Repurpose: Learning Reusable Robot Movement Skills From Human and Animal Behaviors". In: *arXiv preprint arXiv:2203.17138*.

Bommasani, Rishi et al. (2021). "On the opportunities and risks of foundation models". In: *arXiv preprint arXiv:2108.07258*.

Borkar, Vivek S (2005). "An actor-critic algorithm for constrained Markov decision processes". In: *Systems & Control Letters* 54.3, pp. 207–213.

Botev, Zdravko I., Dirk P. Kroese, Reuven Y. Rubinstein, and Pierre L'Ecuyer (2013). "The cross-entropy method for optimization". In: *Handbook of Statistics*. Vol. 31. Elsevier, pp. 35–59.

Broek, Bart van den, Wim Wiegerinck, and Hilbert Kappen (2010). "Risk sensitive path integral control". In: *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*. AUAI Press, pp. 1–8.

Brohan, Anthony, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Autin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich (2022). "RT-1: Robotics transformer for real-world control at scale". In: *Robotics: Science and Systems*.

Brooks, Steve, Andrew Gelman, Galin Jones, and Xiao-Li Meng (2011). *Handbook of Markov Chain Monte Carlo*. CRC Press.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya

Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020). "Language models are few-shot learners". In: *Advances in Neural Information Processing Systems* 33, pp. 1877–1901.

Buchli, Jonas, Evangelos Theodorou, Freek Stulp, and Stefan Schaal (2011). "Variable impedance control: A reinforcement learning approach". In: *Robotics: Science and Systems*.

Camacho, Eduardo F. and Carlos Bordons Alba (2013). *Model Predictive Control*. Springer Science & Business Media.

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch (2021). "Decision Transformer: Reinforcement Learning via Sequence Modeling". In: *Advances in Neural Information Processing Systems* 34.

Cheng, Ching-An, Andrey Kolobov, and Alekh Agarwal (2020). "Policy Improvement via Imitation of Multiple Oracles". In: *Advances in Neural Information Processing Systems* 33.

Cheng, Ching-An, Andrey Kolobov, and Adith Swaminathan (2021). "Heuristic-Guided Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 34, pp. 13550–13563.

Chentanez, Nuttapong, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke (2018). "Physics-Based Motion Capture Imitation With Deep Reinforcement Learning". In: *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, pp. 1–10.

Chernoff, Herman (1952). "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations". In: *The Annals of Mathematical Statistics* 23.4, pp. 493–507.

Chow, Yinlam, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone (2017). "Risk-constrained reinforcement learning with percentile risk criteria". In: *The Journal of Machine Learning Research* 18.1, pp. 6070–6120.

Chow, Yinlam, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh (2018). "A Lyapunov-Based Approach to Safe Reinforcement Learning". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 8092–8101.

Chow, Yinlam, Ofir Nachum, Aleksandra Faust, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh (2019). "Lyapunov-based safe policy optimization for continuous control". In: *arXiv preprint arXiv:1901.10031*.

Christiano, Paul F, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei (2017). "Deep reinforcement learning from human preferences". In: *Advances in Neural Information Processing Systems* 30.

Chua, Kurtland, Roberto Calandra, Rowan McAllister, and Sergey Levine (2018). "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *Advances in Neural Information Processing Systems*, pp. 4754–4765.

Clark, Jack and Dario Amodei (2016). *Faulty Reward Functions in the Wild*. `openai.com/research/faulty-reward-functions`.

CMU (2003). *Carnegie Mellon University Graphics Lab Motion Capture Database*. `http://mocap.cs.cmu.edu`.

Collaboration, Open X-Embodiment et al. (2023). "Open X-Embodiment: Robotic learning datasets and RT-X models". In: *arXiv preprint arXiv:2310.08864*.

Craig, John J (2022). *Introduction to Robotics: Mechanics and Control*. Pearson Education, Inc.

Dalal, Gal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa (2018). "Safe exploration in continuous action spaces". In: *arXiv preprint arXiv:1801.08757*.

Ding, Dongsheng, Xiaohan Wei, Zhuoran Yang, Zhaoran Wang, and Mihailo R Jovanović (2021). "Provably efficient safe exploration via primal-dual policy optimization". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Vol. 130, pp. 3304–3312.

Drews, Paul, Grady Williams, Brian Goldfain, Evangelos A. Theodorou, and James M. Rehg (2019). "Vision-Based High-Speed Driving With a Deep Dynamic Observer". In: *IEEE Robotics and Automation Letters* 4.2, pp. 1564–1571.

Efroni, Yonathan, Shie Mannor, and Matteo Pirotta (2020). "Exploration-Exploitation in Constrained MDPs". In: *arXiv preprint arXiv:2003.02189*.

Erez, Tom, Kendall Lowrey, Yuval Tassa, Vikash Kumar, Svetoslav Kolev, and Emanuel Todorov (2013). "An integrated system for real-time model predictive control of humanoid robots". In: *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, pp. 292–299.

Escontrela, Alejandro, Xue Bin Peng, Wenhao Yu, Tingnan Zhang, Atil Iscen, Ken Goldberg, and Pieter Abbeel (2022). "Adversarial motion priors make good substitutes for complex reward functions". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 25–32.

Eysenbach, Benjamin, Shixiang Gu, Julian Ibarz, and Sergey Levine (2018). "Leave No Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning". In: *International Conference on Learning Representations*.

Facchinei, Francisco and Jong-Shi Pang (2007). *Finite-Dimensional Variational Inequalities and Complementarity Problems*. Springer Science & Business Media.

Falcon, William (2019). *PyTorch Lightning*. `github.com/PyTorchLightning/pytorch-lightning`.

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

Fisac, Jaime F, Anayo K Akametalu, Melanie N Zeilinger, Shahab Kaynama, Jeremy Gillula, and Claire J Tomlin (2018). "A general safety framework for learning-based control in uncertain robotic systems". In: *IEEE Transactions on Automatic Control* 64.7, pp. 2737–2752.

Freeman, C. Daniel, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem (2021). "Brax - A differentiable physics engine for large scale rigid body simulation". In: *Neural Information Processing Systems Datasets and Benchmarks Track*.

Fu, Justin, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine (2020). "D4RL: Datasets for Deep Data-Driven Reinforcement Learning". In: *arXiv preprint arXiv:2004.07219*.

Fuchioka, Yuni, Zhaoming Xie, and Michiel Van de Panne (2023). "OPT-Mimic: Imitation of optimized trajectories for dynamic quadruped behaviors". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5092–5098.

Fukushima, Kunihiko (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4, pp. 193–202.

García, Javier and Fernando Fernández (2015). "A comprehensive survey on safe reinforcement learning". In: *Journal of Machine Learning Research* 16.1, pp. 1437–1480.

Geibel, Peter and Fritz Wysotzki (2005). "Risk-sensitive reinforcement learning applied to control under constraints". In: *Journal of Artificial Intelligence Research* 24, pp. 81–108.

Glynn, Peter W. (1990). "Likelihood ratio gradient estimation for stochastic systems". In: *Communications of the ACM* 33.10, pp. 75–84.

Goldfain, Brian, Paul Drews, Changxi You, Matthew Barulic, Orlin Velev, Panagiotis Tsiotras, and James M. Rehg (2019). "AutoRally: An Open Platform for Aggressive Autonomous Driving". In: *IEEE Control Systems Magazine* 39.1, pp. 26–55.

Gómez, Vicenç, Sep Thijssen, Andrew Symington, Stephen Hailes, and Hilbert Kappen (2016). "Real-time stochastic optimal control for multi-agent quadrotor systems". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 26, pp. 468–476.

Goschin, Sergiu, Ari Weinstein, and Michael Littman (2013). "The Cross-Entropy Method Optimizes for Quantiles". In: *International Conference on Machine Learning*. PMLR, pp. 1193–1201.

Greydanus, Samuel, Misko Dzamba, and Jason Yosinski (2019). "Hamiltonian neural networks". In: *Advances in Neural Information Processing Systems* 32.

Hall, Eric and Rebecca Willett (2013). "Dynamical models and tracking regret in online convex programming". In: *International Conference on Machine Learning*, pp. 579–587.

Hans, Alexander, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft (2008). "Safe exploration for reinforcement learning". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pp. 143–148.

Hansen, Nikolaus, Sibylle D. Müller, and Petros Koumoutsakos (2003). "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)". In: *Evolutionary Computation* 11.1, pp. 1–18.

Harvey, Félix G, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal (2020). "Robust Motion In-Betweening". In: *ACM Transactions on Graphics (TOG)* 39.4, pp. 60–1.

Hasenclever, Leonard, Fabio Pardo, Raia Hadsell, Nicolas Heess, and Josh Merel (2020). "CoMic: Complementary Task Learning & Mimicry for Reusable Skills". In: *International Conference on Machine Learning*. PMLR, pp. 4105–4115.

Hazan, Elad (2016). "Introduction to online convex optimization". In: *Foundations and Trends® in Optimization* 2.3-4, pp. 157–325.

Heess, Nicolas, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David

Silver (2017). “Emergence of Locomotion Behaviours in Rich Environments”. In: *arXiv preprint arXiv:1707.02286*.

Helvik, Bjarne E. and Otto Wittner (2002). “Using the Cross-Entropy Method to Guide/Govern Mobile Agent’s Path Finding in Networks”. In: *International Workshop on Mobile Agents for Telecommunication Applications* 2164.

Henighan, Tom, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M Ziegler, John Schulman, Dario Amodei, and Sam McCandlish (2020). “Scaling laws for autoregressive generative modeling”. In: *arXiv preprint arXiv:2010.14701*.

Howell, Taylor, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa (2022). “Predictive sampling: Real-time behaviour synthesis with MuJoCo”. In: *arXiv preprint arXiv:2212.00541*.

Ionescu, Catalin, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu (2013). “Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.7, pp. 1325–1339.

Janner, Michael, Qiyang Li, and Sergey Levine (2021). “Offline Reinforcement Learning as One Big Sequence Modeling Problem”. In: *Advances in Neural Information Processing Systems* 34.

Kakade, Sham and John Langford (2002). “Approximately optimal approximate reinforcement learning”. In: *International Conference on Machine Learning*.

Kania, Kacper, Marek Kowalski, and Tomasz Trzciński (2021). “TrajeVAE: Controllable Human Motion Generation from Trajectories”. In: *arXiv preprint arXiv:2104.00351*.

Kaplan, Jared, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei (2020). “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361*.

Karpathy, Andrej (2020). *minGPT*. `github.com/karpathy/minGPT`.

Kingma, Diederik P and Jimmy Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations*.

Kirillov, Alexander, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girschick (2023). “Segment anything”. In: *arXiv preprint arXiv:2304.02643*.

Kivinen, Jyrki and Manfred K Warmuth (1997). "Exponentiated gradient versus gradient descent for linear predictors". In: *Information and Computation* 132.1, pp. 1–63.

Kober, Jens, J Andrew Bagnell, and Jan Peters (2013). "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11, pp. 1238–1274.

Kober, Jens and Jan Peters (2008). "Policy search for motor primitives in robotics". In: *Advances in Neural Information Processing Systems* 21.

Kobilarov, Marin (2012). "Cross-entropy randomized motion planning". In: *Robotics: Science and Systems*. Vol. 7, pp. 153–160.

Kormushev, Petar, Sylvain Calinon, and Darwin G Caldwell (2010). "Robot motor skill coordination with EM-based reinforcement learning". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 3232–3237.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems* 25.

Kumar, Ashish, Zipeng Fu, Deepak Pathak, and Jitendra Malik (2021). "RMA: Rapid motor adaptation for legged robots". In: *Robotics: Science and Systems*.

Laskey, Michael, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg (2017). "DART: Noise Injection for Robust Imitation Learning". In: *Conference on Robot Learning*. PMLR, pp. 143–156.

Le, Hoang, Cameron Voloshin, and Yisong Yue (2019). "Batch policy learning under constraints". In: *International Conference on Machine Learning*. PMLR, pp. 3703–3712.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lee, Jason D, Ioannis Panageas, Georgios Piliouras, Max Simchowitz, Michael I Jordan, and Benjamin Recht (2019). "First-Order Methods Almost Always Avoid Strict Saddle Points". In: *Mathematical Programming* 176.1, pp. 311–337.

Lee, Joonho, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter (2020). "Learning quadrupedal locomotion over challenging terrain". In: *Science Robotics* 5.47.

Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel (2016). "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1, pp. 1334–1373.

Levine, Sergey, Aviral Kumar, George Tucker, and Justin Fu (2020). "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: *arXiv preprint arXiv:2005.01643*.

Levine, Sergey, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen (2018). "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International Journal of Robotics Research* 37.4-5, pp. 421–436.

Li, Chenhao, Marin Vlastelica, Sebastian Blaes, Jonas Frey, Felix Grimminger, and Georg Martius (2022). "Learning agile skills via adversarial imitation of rough partial demonstrations". In: *Conference on Robot Learning*. PMLR, pp. 342–352.

Li, Shuo and Osbert Bastani (2020). "Robust model predictive shielding for safe reinforcement learning with stochastic dynamics". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 7166–7172.

Lin, Tianyi, Chi Jin, and Michael Jordan (2020). "On gradient descent ascent for nonconvex-concave minimax problems". In: *International Conference on Machine Learning*. PMLR, pp. 6083–6093.

Liu, Siqi, Guy Lever, Zhe Wang, Josh Merel, S. M. Ali Eslami, Daniel Hennes, Wojciech M. Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, Noah Y. Siegel, Leonard Hasenclever, Luke Marris, Saran Tunyasuvunakool, H. Francis Song, Markus Wulfmeier, Paul Muller, Tuomas Haarnoja, Brendan D. Tracey, Karl Tuyls, Thore Graepel, and Nicolas Heess (2022). "From Motor Control to Team Play in Simulated Humanoid Football". In: *Science Robotics* 7.69.

Ljung, Lennart (1987). *System Identification: Theory for the User*. Prentice Hall.

Lowe, David G (2004). "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision* 60, pp. 91–110.

Lowrey, Kendall, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch (2019). "Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control". In: *International Conference on Learning Representations*.

Ma, Yecheng Jason, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar (2023). "Eureka: Human-level reward design via coding large language models". In: *arXiv preprint arXiv:2310.12931*.

Makoviychuk, Viktor, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State (2021). "Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning". In: *Neural Information Processing Systems Datasets and Benchmarks Track*.

Mannor, Shie, Reuven Y. Rubinstein, and Yohai Gat (2003). "The cross entropy method for fast policy search". In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 512–519.

Mao, Wei, Miaomiao Liu, Mathieu Salzmann, and Hongdong Li (2019). "Learning Trajectory Dependencies for Human Motion Prediction". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9489–9497.

Margolis, Gabriel B, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal (2022). "Rapid locomotion via reinforcement learning". In: *Robotics: Science and Systems*.

Mayne, David Q. (2014). "Model predictive control: Recent developments and future promise". In: *Automatica* 50.12, pp. 2967–2986.

Menache, Ishai, Shie Mannor, and Nahum Shimkin (2005). "Basis Function Adaptation in Temporal Difference Reinforcement Learning". In: *Annals of Operations Research* 134 (1), pp. 215–238.

Merel, Josh, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne (2019). "Hierarchical Visuomotor Control of Humanoids". In: *International Conference on Learning Representations*.

Merel, Josh, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess (2019). "Neural Probabilistic Motor Primitives for Humanoid Control". In: *International Conference on Learning Representations*.

Merel, Josh, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess (2017). "Learning Human Behaviors from Motion Capture by Adversarial Imitation". In: *arXiv preprint arXiv:1707.02201*.

Merel, Josh, Saran Tunyasuvunakool, Arun Ahuja, Yuval Tassa, Leonard Hasenclever, Vu Pham, Tom Erez, Greg Wayne, and Nicolas Heess (2020). "Catch & Carry: Reusable Neural Controllers for Vision-Guided Whole-Body Tasks". In: *ACM Transactions on Graphics (TOG)* 39.4, pp. 39–1.

Michael, Nathan, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar (2010). "The GRASP multiple micro-UAV testbed". In: *IEEE Robotics & Automation Magazine* 17.3, pp. 56–65.

Mitrovic, Djordje, Stefan Klanke, and Sethu Vijayakumar (2010). "Adaptive optimal feedback control with learned internal dynamics models". In: *From Motor Learning to Interaction Learning in Robots*, pp. 65–84.

Miyashita, Megumi, Shiro Yano, and Toshiyuki Kondo (2018). "Mirror descent search and its acceleration". In: *Robotics and Autonomous Systems* 106, pp. 107–116.

Morimoto, Jun and Christopher Atkeson (2002). "Minimax differential dynamic programming: An application to robust biped walking". In: *Advances in Neural Information Processing Systems* 15.

Mourot, Lucas, Ludovic Hoyet, François Le Clerc, François Schnitzler, and Pierre Hellier (2022). "A Survey on Deep Learning for Skeleton-Based Human Animation". In: *Computer Graphics Forum*. Vol. 41. 1. Wiley Online Library, pp. 122–157.

Murphy, Kevin P (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

Nielsen, Frank and Vincent Garcia (2009). "Statistical exponential families: A digest with flash cards". In: *arXiv preprint arXiv:0911.4863*.

Okada, Masashi and Tadahiro Taniguchi (2018). "Acceleration of Gradient-based Path Integral Method for Efficient Optimal and Inverse Optimal Control". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3013–3020.

OpenAI (2023). "GPT-4 technical report". In: *arXiv preprint arXiv:2303.08774*.

Ouyang, Long, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe (2022). "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems* 35, pp. 27730–27744.

Pan, Yunpeng, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A Theodorou, and Byron Boots (2020). "Imitation learning for agile autonomous driving". In: *The International Journal of Robotics Research* 39.2-3, pp. 286–302.

Peng, Xue Bin, Pieter Abbeel, Sergey Levine, and Michiel van de Panne (2018). "DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills". In: *ACM Transactions on Graphics (TOG)* 37.4, pp. 1–14.

Peng, Xue Bin, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine (2019). "MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies". In: *Advances in Neural Information Processing Systems* 32.

Peng, Xue Bin, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine (2020). "Learning Agile Robotic Locomotion Skills by Imitating Animals". In: *Robotics: Science and Systems*.

Peng, Xue Bin, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler (2022). "ASE: Large-Scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters". In: *ACM Transactions On Graphics (TOG)* 41.4, pp. 1–17.

Peng, Xue Bin, Aviral Kumar, Grace Zhang, and Sergey Levine (2019). "Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning". In: *arXiv preprint arXiv:1910.00177*.

Perkins, Theodore J and Andrew G Barto (2002). "Lyapunov design for safe reinforcement learning". In: *Journal of Machine Learning Research* 3.Dec, pp. 803–832.

Peters, Jan and Stefan Schaal (2006a). "Policy gradient methods for robotics". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 2219–2225.

— (2006b). "Reinforcement learning for parameterized motor primitives". In: *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. IEEE, pp. 73–80.

— (2007). "Reinforcement Learning by Reward-Weighted Regression for Operational Space Control". In: *Proceedings of the 24th International Conference on Machine Learning*, pp. 745–750.

— (2008a). "Learning to control in operational space". In: *The International Journal of Robotics Research* 27.2, pp. 197–212.

— (2008b). "Reinforcement learning of motor skills with policy gradients". In: *Neural Networks* 21.4, pp. 682–697.

Pinto, Lerrel and Abhinav Gupta (2016). "Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3406–3413.

Polo, Francisco Javier Garcia and Fernando Fernandez Rebollo (2011). "Safe reinforcement learning in high-risk tasks through policy improvement". In: *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, pp. 76–83.

Puterman, Martin L (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Qin, S Joe and Thomas A Badgwell (2003). "A survey of industrial model predictive control technology". In: *Control Engineering Practice* 11.7, pp. 733–764.

Qiu, Shuang, Xiaohan Wei, Zhuoran Yang, Jieping Ye, and Zhaoran Wang (2020). "Upper Confidence Primal-Dual Reinforcement Learning for CMDP with Adversarial Loss". In: *Advances in Neural Information Processing Systems* 33.

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever (2018). "Improving language understanding by generative pre-training". In: *OpenAI Blog*.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). "Language models are unsupervised multitask learners". In: *OpenAI Blog*.

Raffin, Antonin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann (2021). "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research*.

Rattray, Magnus, David Saad, and Shun-ichi Amari (1998). "Natural gradient descent for on-line learning". In: *Physical Review Letters* 81.24, p. 5461.

Reed, Scott, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-maron, Mai Giménez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas (2022). "A generalist agent". In: *Transactions on Machine Learning Research*.

Ross, Stéphane, Geoffrey Gordon, and Drew Bagnell (2011). "A reduction of imitation learning and structured prediction to no-regret online learning". In: *International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, pp. 627–635.

Roy, Nicholas, Ingmar Posner, Tim Barfoot, Philippe Beaudoin, Yoshua Bengio, Jeannette Bohg, Oliver Brock, Isabelle Depatie, Dieter Fox, Dan Koditschek, Tomás Lozano-Pérez, Vikash Mansinghka, Christopher Pal, Blake Richards, Dorsa Sadigh, Stefan Schaal, Gaurav Sukhatme, Denis Thérien, Marc Toussaint, and Michiel van de Panne (2021). "From machine learning to robotics: Challenges and opportunities for embodied intelligence". In: *arXiv preprint arXiv:2110.15245*.

Rudin, Nikita, David Hoeller, Philipp Reist, and Marco Hutter (2022). "Learning to walk in minutes using massively parallel deep reinforcement learning". In: *Conference on Robot Learning*. PMLR, pp. 91–100.

Schaal, Stefan (1996). "Learning from demonstration". In: *Advances in Neural Information Processing Systems* 9.

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel (2016). "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *International Conference on Learning Representations*.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Shalev-Shwartz, Shai, Shaked Shammah, and Amnon Shashua (2016). "Safe, multi-agent reinforcement learning for autonomous driving". In: *arXiv preprint arXiv:1610.03295*.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489.

Smith, Laura, J Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine (2022). "Legged robots that keep on learning: Fine-tuning locomotion policies in the real world". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1593–1599.

Srinivas, Aravind, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn (2018). "Universal planning networks: Learning generalizable representations for visuomotor control". In: *International Conference on Machine Learning*. PMLR, pp. 4732–4741.

Srinivasan, Krishnan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn (2020). "Learning to be Safe: Deep RL with a Safety Critic". In: *arXiv preprint arXiv:2010.14603*.

Stachowicz, Kyle, Dhruv Shah, Arjun Bhorkar, Ilya Kostrikov, and Sergey Levine (2023). "FastRLAP: A system for learning High-speed driving via deep RL and autonomous practicing". In: *Conference on Robot Learning*.

Stulp, Freek, Evangelos A Theodorou, and Stefan Schaal (2012). "Reinforcement learning with sequences of motion primitives for robust manipulation". In: *IEEE Transactions on Robotics* 28.6, pp. 1360–1370.

Sünderhauf, Niko, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, and Peter Corke (2018). "The limits and potentials of deep learning for robotics". In: *The International Journal of Robotics Research* 37.4-5, pp. 405–420.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement Learning: An Introduction*. MIT Press.

Szita, István and András Lörincz (2006). "Learning Tetris using the noisy cross-entropy method". In: *Neural Computation* 18.12, pp. 2936–2941.

Tamar, Aviv, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel (2016). "Value iteration networks". In: *Advances in Neural Information Processing Systems* 29.

Tassa, Yuval, Nicolas Mansard, and Emo Todorov (2014). "Control-limited differential dynamic programming". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1168–1175.

Tessler, Chen, Daniel J Mankowitz, and Shie Mannor (2018). "Reward Constrained Policy Optimization". In: *International Conference on Learning Representations*.

Tevet, Guy, Brian Gordon, Amir Hertz, Amit H Bermano, and Daniel Cohen-Or (2022). "MotionCLIP: Exposing Human Motion Generation to CLIP Space". In: *European Conference on Computer Vision*. Springer.

Thananjeyan, Brijen, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg (2021). "Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones". In: *IEEE Robotics and Automation Letters* 6.3, pp. 4915–4922.

Theodorou, Evangelos, Jonas Buchli, and Stefan Schaal (2010). "A generalized path integral control approach to reinforcement learning". In: *The Journal of Machine Learning Research* 11, pp. 3137–3181.

Theodorou, Evangelos A (2015). "Nonlinear stochastic control and information theoretic dualities: Connections, interdependencies, and thermodynamic interpretations". In: *Entropy* 17.5, pp. 3352–3375.

Theodorou, Evangelos A and Emanuel Todorov (2012). "Relative entropy and free energy dualities: Connections to path integral and KL control". In: *2012 IEEE 51st Conference on Decision and Control (CDC)*. IEEE, pp. 1466–1473.

Tieleman, Tijmen and Geoffrey Hinton (2012). "Lecture 6.5 — RMSprop: Divide the gradient by a running average of its recent magnitude". COURSERA: Neural Networks for Machine Learning.

Todorov, Emanuel (2006). "Linearly-solvable Markov decision problems". In: *Advances in Neural Information Processing Systems* 19.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "MuJoCo: A Physics Engine for Model-Based Control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5026–5033.

Tunyasuvunakool, Saran, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa (2020). "`dm_control`: Software and Tasks for Continuous Control". In: *Software Impacts* 6, p. 100022.

Turchetta, Matteo, Andrey Kolobov, Shital Shah, Andreas Krause, and Alekh Agarwal (2020). "Safe Reinforcement Learning via Curriculum Induction". In: *Advances in Neural Information Processing Systems* 33.

Urmson, Chris, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William Whittaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson (2008). "Autonomous driving in urban environments: Boss and the Urban Challenge". In: *Journal of Field Robotics* 25.8, pp. 425–466.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems*, pp. 5998–6008.

Venkatraman, Arun, Martial Hebert, and J Andrew Bagnell (2015). "Improving multi-step prediction of learned time series models". In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Wabersich, Kim Peter and Melanie N Zeilinger (2021). "A predictive safety filter for learning-based control of constrained nonlinear dynamical systems". In: *Automatica* 129, p. 109597.

Wainwright, Martin J and Michael I Jordan (2008). "Graphical models, exponential families, and variational inference". In: *Foundations and Trends® in Machine Learning* 1.1–2, pp. 1–305.

Wang, Borui, Ehsan Adeli, Hsu-kuang Chiu, De-An Huang, and Juan Carlos Niebles (2019). "Imitation Learning for Human Pose Prediction". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7124–7133.

Wang, Yikai, Zheyuan Jiang, and Jianyu Chen (2023). "Learning robust, agile, natural legged locomotion skills in the wild". In: *arXiv preprint arXiv:2304.10888*.

Wang, Ziyu, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess (2017). "Robust Imitation of Diverse Behaviors". In: *Advances in Neural Information Processing Systems* 30.

Wierstra, Daan, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber (2014). "Natural evolution strategies". In: *The Journal of Machine Learning Research* 15.1, pp. 949–980.

Williams, Grady, Andrew Aldrich, and Evangelos A Theodorou (2017). "Model predictive path integral control: From theory to parallel computation". In: *Journal of Guidance, Control, and Dynamics* 40.2, pp. 344–357.

Williams, Grady, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou (2016). "Aggressive driving with model predictive path integral control". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1433–1440.

— (2018). "Information-Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving". In: *IEEE Transactions on Robotics* 34.6, pp. 1603–1622.

Williams, Grady, Brian Goldfain, Paul Drews, Kamil Saigol, James M. Rehg, and Evangelos A. Theodorou (2018). "Robust sampling based model predictive control with sparse objective information". In: *Robotics: Science and Systems*.

Williams, Grady, Nolan Wagener, Brian Goldfain, Paul Drews, James M. Rehg, Byron Boots, and Evangelos A. Theodorou (2017). "Information theoretic MPC for model-based reinforcement learning". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1714–1721.

Yang, Yuxiang, Ken Caluwaerts, Atil Iscen, Tingnan Zhang, Jie Tan, and Vikas Sindhwani (2020). "Data efficient reinforcement learning for legged robots". In: *Conference on Robot Learning*. PMLR, pp. 1–10.

Yuan, Ye and Kris Kitani (2020). "Residual Force Control for Agile Human Behavior Imitation and Extended Motion Synthesis". In: *Advances in Neural Information Processing Systems* 33, pp. 21763–21774.

Zhang, Lijun, Shiyin Lu, and Zhi-Hua Zhou (2018). "Adaptive Online Learning in Dynamic Environments". In: *Advances in Neural Information Processing Systems*, pp. 1323–1333.

Zhang, Wei, Han Wang, Carsten Hartmann, Marcus Weber, and Christof Schütte (2014). "Applications of the cross-entropy method to importance sampling and optimal control of diffusions". In: *SIAM Journal on Scientific Computing* 36.6, A2654–A2672.

Zhuang, Ziwen, Zipeng Fu, Jianren Wang, Christopher G Atkeson, Sören Schwertfeger, Chelsea Finn, and Hang Zhao (2023). "Robot Parkour Learning". In: *Conference on Robot Learning*.

Zitkovich, Brianna, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R Sanketi, Grecia Salazar, Michael S Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J Joshi, Alex Irpan, brian ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han (2023). "RT-2: Vision-language-action models transfer web knowledge to robotic control". In: *Conference on Robot Learning*.