Exploiting Structure in Learning:
A Path Toward Building Safe and Adaptive Robots

Anqi Li

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2023

Reading Committee:

Byron Boots, Chair

Dieter Fox

Siddhartha S. Srinivasa

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Exploiting Structure in Learning:
A Path Toward Building Safe and Adaptive Robots

Anqi Li

Chair of the Supervisory Committee:
Byron Boots
Computer Science & Engineering

As robots venture into real-world applications, there is an increasing need for them to effectively learn from experience and adapt to unseen situations. This thesis addresses a few critical challenges to practical robot learning, including safety and sample efficiency. The main perspective of this thesis is to leverage various types of structure, ranging from explicit domain knowledge about robotics problem, such as system dynamics, task decomposition, etc., to hidden structure in robotics data. Such structure can provide valuable insight into solving robotics tasks, but cannot be directly used by most off-the-shelf learning-based solutions. The first part of this thesis focuses on encoding known domain knowledge into structured policy classes for learning, giving standard off-the-shelf learning algorithms the ability to admit formal safety guarantees, learn efficiently with small datasets, and generalize well to new conditions. The second part of this thesis considers the structure in robotics data. This thesis considers two types of structure in data: *1)* explicit structure given by what information each subset of data provides and *2)* implicit structure induced by common data collection processes. By reasoning about the explicit structure in data, this thesis introduces a general learning paradigm and an associated learning algorithm which have theoretical guarantees and work well empirically. Finally, this thesis shows that implicit structure from data collection processes can be sometimes unintentionally leveraged by learning algorithms to achieve seemingly surprising robustness and safety properties.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Byron Boots, for being unbelievably supportive throughout my Ph.D. journey, allowing me to bounce around on different research and non-research topics. Byron has been such a great listener and mentor, who is able to put up with even my most naïve opinions and always gives me his most genuine advise.

Next, I am grateful that Magnus Egerstedt took me as his student and co-advised me during my time at Georgia Institute of Technology, from whom I received a lot of precious advice that I treasure until today. I would like to thank my thesis committee members, Dieter Fox, Siddhartha Srinivasa and Karen Leung for their insightful comments, sharp questions, and valuable feedback on my research. I am fortunate to have met a few awesome mentors outside of the University of Washington and the Georgia Institute of Technology. I spent two wonderful summers at NVIDIA with Nathan Ratliff and Karl Van Wyk, and a great summer at FAIR working with Franziska Meier. I also want to thank Katia Sycara who kindly advised me during my Master's study at Carnegie Mellon University and guided me through my first few steps as a researcher.

My Ph.D. journey wouldn't be the same without my amazing peers at UW and GT. I am here to thank all members and friends of the UW (and earlier GT) Robot Learning Lab: Jing, Mustafa, Amirreza, Ching-An, Asif, Xinyan, Sasha, Siddarth, Nolan, Jake, Boling, Nathan H, Siyuan, Guanya, Alex, Schmittle, Mohak, Sandesh, Adam, Yuxiang, Carolina, Rosario, Tyler, Kevin, Sanghun, Mateo and Rohan, and GT Grits Lab: Paul, Ian, Siddharth, Gennaro, María, Yousef, Chris, Jeremy, Pietro and Sean.

Lastly, I am indebted to my family who genuinely supported my throughout my Ph.D. study. I would like to thank Ching-An for being a great partner both in work and in life. Finally, I owe special thanks to our cats, Alice, Ramesses and Argo, for significant contributions to my mental health and work-life balance during the last few years.

DEDICATION

To my friends and family

Chapter 1

INTRODUCTION

As robots venture into real-world applications, there is an increasing need for them to effectively learn from experience and adapt to unseen situations. However, real-world robot learning remains a challenge as state-of-the-art machine learning algorithms still struggle to solve simple robotics problems. This is because robotics problems impose the following special requirements to a machine learning algorithm:

- *Safety:* The violation of safety, either during learning or upon deployment, can present a significant danger to humans, the environment, and the robot itself; and

- *Efficiency:* Collecting samples for robotics problems demands significant time and human effort.

These requirements are especially challenging to generic machine learning algorithms which make little or no assumptions on the underlying task or data collection mechanism. To address the challenges of safety and efficiency, this thesis proposes to leverage various types structure in robot learning, ranging from explicit domain knowledge about robotics problem, such as system dynamics, task decomposition, etc., to (perhaps hidden) structure in robotics data. Such structure is often readily available in robotics tasks. It can provide valuable insight into solving robotics tasks, but cannot be directly used by most off-the-shelf machine learning algorithms. This leads to the following thesis statement:

By exploiting structure, we can build robots that can learn both safely and efficiently.

This thesis is consist of two complementary parts:

1) Designing robot policy classes that can exploit structure in *robotics problems*; and

2) Developing and studying machine learning algorithms that can leverage structure in *robotics data*.

The first part of this thesis focuses on encoding known domain knowledge into structured policy classes for learning. We propose a novel policy class, called RMP^2 policies, which can leverage known domain knowledge such as system dynamics, robot kinematics, task decomposition, etc. By incorporating control theory into design, RMP^2 policies provide generalizable inductive biases and non-statistical guarantees in learning, such as safety and stability. This further improves learning efficiency as it inherently encourages exploration within high-reward regions. We show that learning with RMP^2 policies is sample efficient and has strong generalization capability. In imitation learning scenarios (Chapter 5), RMP^2 policies can achieve high performance given less than ten demonstrations and show strong robustness against disturbances. In reinforcement learning scenarios (Chapter 6), RMP^2 policies succeed in tasks that have proved challenging for general function approximator policies, result in significantly fewer safety violations, and are less sensitive to reward designs.

The second part of this thesis considers structure in robotics data. This thesis considers two types of structure in data: explicit structure given by what information each subset of data provides (Chapter 7); and implicit structure induced by common data collection processes (Chapter 8). When robotics data is collected from heterogeneous sources, some subset of robotics data may contain either dynamics or reward information (but not both). Such data can not be directly usable by existing learning paradigm due to missing information. By explicit reasoning about this structure of data composition, we introduce a general learning paradigm and an associated learning algorithm which have theoretical guarantees and work well empirically. Finally, in many cases, structure is not given to the learning algorithm explicitly, but rather presented *implicitly* in data due to biases in data collection processes. For example, it can be hard to enumerate all the safety constraints for a robot in a complex environment, while it is very common during data collection to stop the robot when it is about to take dangerous actions. We show that a few offline reinforcement learning algorithms can leverage such hidden structure in data, which explains the surprising robustness property to wrong reward and safety property we observe in these algorithms.

1.1 Organization

The remainder of the thesis is organized as follows.

Chapter 2: Preliminaries We discuss notations and conventions being used in the rest of the thesis, and introduce basic concepts in control, with the focus on Lyapunov stability, and reinforcement learning.

Part I: Leveraging Structure in Robotics Problem We focus on encoding known structure in robotics problems, such as system dynamics, task decomposition, success and failure conditions, etc., into a structured policy class called RMP^2 policies for learning.

Chapter 3: Riemannian Motion Policies We introduce the Riemannian motion policy (RMP) framework and an algorithm for motion generation called RMP^2. We discuss the control-theoretic properties of RMP^2.

Chapter 4: Parameterizing RMP^2 Policies We convert RMP from a control framework to a structured policy class for learning. We discuss how to parameterize RMP^2 policies in an expressive way while leveraging known problem structure.

Chapter 5: Learning RMP^2 Policies from Human Demonstrations We provide a few case studies on efficiently learning RMP^2 policies from expert demonstrations given by humans.

Chapter 6: Reinforcement Learning with RMP^2 Policies We show that RMP^2 policies can improve safety and efficiency of online reinforcement learning.

Part II: Leveraging Structure in Robotics Data We consider the scenario of learning robot policy from offline data. We discuss how to leverage explicit and implicit structure in offline robotics data.

Chapter 7: Leveraging Explicit Structure in Robotics Data We propose offline policy learning from observation (PLfO), a learning formulation for when subsets of offline data contain either dynamics or reward information, and MAHALO, an learning algorithm for solving offline PLfO problems.

Chapter 8: Exploiting Implicit Structure in Robotics Data We present empirical results of surprising robustness and safety properties of offline reinforcement algorithms, and discuss how these properties can be partially attributed to implicit structure in data induced by its collection process.

Chapter 2

PRELIMINARIES

In order to enable safe and efficient robot learning, we need to draw complimentary tools from different fields, including control theory and reinforcement learning. The main goal for this chapter is to get readers familiar with the minimum set of concepts in these fields that are necessary to follow the remaining of the thesis. Therefore, the contents in this chapter are by no means thorough, and a lot of technicalities are intentionally omitted. Interested readers will be referred to more comprehensive materials, e.g., textbooks, for focused and technical discussions on these matters.

The reason that there is a chapter dedicated to these preliminaries (rather than making them scattered around the thesis) is to help readers make the connections across different fields, and understand why and how these tools can be used in conjunction with one another.

Careful readers may notice that, across different sections of this chapter (and later in the thesis), that different notations can be used to refer to the same (or similar) concepts. For example, the *state* is represented by either $\mathbf{x}$ or s, and the *control input* (i.e., action) is represented by either $\mathbf{u}$ or a, depending on whether it is in a control theory or reinforcement learning context. Such inconsistencies in notation is kept deliberately to minimize the gap between this thesis and its surrounding literature. With the understanding of the potential confusions, Section 2.4 compares notations commonly used in control theory and reinforcement learning literature and discuss the subtleties of these notations.

2.1 Notations

In this thesis, typefaces will be used to help denote different concepts, e.g., scalars, vectors, matrices, sets, operators, etc., as is listed in Table 2.1. For example, boldface lowercase letters like $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are used to represent vectors, and boldface uppercase letters like $\mathbf{A}, \mathbf{M}, \mathbf{J}$ are refer to matrices. Normal-weight letters can be used to represent different concepts,

depending on the context, or when referring to an abstract concept. Later on, we will see that we use s to denote the state of an Markov decision process (MDP), where s is an abstract concept which could be a scalar, a vector, a matrix, or something else.

Font Typeface	Typical Meaning(s)
f, g, x, y, R	Normal-weight letters can be used for scalars, vectors, maps, abstract concepts, etc., depending on the context
α, β, γ	Lowercase Greek letters are often used for scalars, with exceptions of π (policy), ϕ, ψ (map), μ, ν (measure)
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	Bold lowercase letters are used for vectors
$\mathbf{A}, \mathbf{M}, \mathbf{J}$	Bold uppercase letters are used for Matrices
$\mathbb{P}, \mathbb{B}$	Blackboard uppercase letters are often used for operators, with the exception of $\mathbb{R}$ (real numbers), $\mathbb{E}$ (expectation), $\mathbb{I}$ (identity matrix)
$\mathcal{C}, \mathcal{D}$	Calligraphic font is used for sets, with the exception of $\mathcal{N}$ (Gaussian distribution)
`FK`, `chol`	Teletype font is for algorithms or computational procedure

Table 2.1: Table of common font typefaces and their typical meanings

For a vector $\mathbf{x}$, we use x_i to denote its i-th component. For a matrix $\mathbf{A}$, we use $\mathbf{a}_i$ to denote the its i-th column vector and A_{ij} to represent its component at row i and column j. The transpose of a vector or a matrix is denoted as $(\cdot)^\top$. For example, $\mathbf{x}^\top$ and $\mathbf{A}^\top$ are the transposes of vector $\mathbf{x}^\top$ and matrix $\mathbf{A}^\top$, respectively. We use $(\cdot)^\dagger$ to denote the Moore–Penrose inverse of a matrix. A square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is said to be positive definite, denoted $\mathbf{A} \succ 0$ if $\mathbf{x}^\top \mathbf{A} \mathbf{x} > 0$ for all $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, and is positive semi-definite, denoted $\mathbf{A} \succeq 0$ if $\mathbf{x}^\top \mathbf{A} \mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$. We use $\mathbb{R}_+^{n \times n}$ to denote the set of all positive definite $n \times n$ matrices.

In Table 2.2, we list some functions that we will be used in this thesis. In particular, for a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$, we denote its differential as $\frac{\partial f}{\partial \mathbf{x}}$, and its gradient as $\nabla_{\mathbf{x}} f$. We follow the convention that differential to a real-valued function is a *row* vector, and the gradient is a *column* vector, i.e., $\frac{\partial f}{\partial \mathbf{x}} = \nabla_{\mathbf{x}} f^\top$.

Function	Meaning
sup	Supremum
inf	Infimum
$\text{sign}(\cdot)$	Sign function
$\|\cdot\|_p$	ℓ_p norm
$\|\cdot\|$	ℓ_2 norm
$\|\cdot\|_{\mathbf{A}}$, where $\mathbf{A} \succ 0$	Weighted ℓ_2 norm, $\|\mathbf{x}\|_{\mathbf{A}} = \mathbf{x}^\top \mathbf{A}\mathbf{x}$
∇	Gradient
∂	Differential and Partial derivative
$\circ$	Composition of maps, $f \circ g : x \mapsto f(g(x))$
$\odot$	Pointwise multiplication
$\text{supp}(\cdot)$	Support, for $f : \mathcal{X} \to \mathbb{R}$, $\text{supp}(f) \coloneqq \{x \in \mathcal{X} : f(x) \neq 0\}$
$\mathbb{E}$	Expectation
$\mathbb{1}$	Indicator function.
	Given a predicate P, $\mathbb{1}(P) = 1$ if P is True, and 0 otherwise.
$\|\mathcal{A}\|$	Set cardinality
$\setminus$	Set difference (unless otherwise specified),
	Given sets $\mathcal{A}, \mathcal{B}$, $\mathcal{A} \setminus \mathcal{B} \coloneqq \{x \in \mathcal{A} : x \notin \mathcal{B}\}$
$\Delta(\cdot)$	Probability simplex

Table 2.2: Table of functions

2.2 Control Theory

This section aims to give readers an brief introduction to control theory, with the focus on (Lyapunov) stability analysis. We will first define dynamical systems and different notions of stability. Next, we will introduce control theoretic tools for ensuring stability.

Later on in Part I of this thesis, we will use stability to establish desirable long-term properties of the robotics system including safety guarantees, e.g., the robot will never collide with obstacles or violating joint limits, and performance guarantees, e.g., the robot can always reach the goal. Interested readers are recommend to check out nonlinear control textbooks [Slotine et al., 1991, Khalil and Grizzle, 2002, Sastry, 2013] for more thorough discussions on this subject.

2.2.1 Dynamical Systems

As its name suggested, a dynamical system is a system whose state evolves over *time*. In Part I of this thesis, we will be constantly working with dynamical system modeled by a set of differential equations,

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}), \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the *state* variables, $\mathbf{u} \in \mathbb{R}^m$ is the *control input*, and $f : (\mathbb{R}^n \times \mathbb{R}^m) \to \mathbb{R}^n$ is a locally Lipschitz continuous[1] map describing the *dynamics* of the system. We use $x(t)$ to denote the *solution* to the differential equation (2.1), with the initial value defined at $x(0)$. It describes the state trajectory of system (2.1) if initialized at $x = x(0)$.

Below we provide a few simple examples of dynamical systems. These dynamical systems can be commonly encountered in robotics. We will study one of them more extensively later in Chapter 3.

Example 2.2.1 (Single Integrator System)**.** A *single integrator system* is described by the differential equation below,

$$\dot{x} = u,$$

[1]The Lipschitz continuity assumption is used to ensure the existence and uniqueness of solutions to the differential equations. The readers can refer to Khalil and Grizzle [2002] for details.

where $x \in \mathbb{R}, u \in \mathbb{R}$. The name of the system follows from the fact that the value of x at any time t is given by *integrating* the control input over time, i.e., $x(t) = x_0 + \int_0^t u(\tau)\, d\tau$ where x_0 is the value of x at $t = 0$. This single integrator system is commonly used to model robotics systems where we can directly control the velocity of the robot. Single integrator systems can be generalized to cases where $x, u \in \mathbb{R}^d$. For example, we can use $d = 2$ to model a 2-d robot which we can directly control the velocity along the x- and y-directions.

Example 2.2.2 (Double Integrator System)**.** A *double integrator system* is described as,

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ u \end{bmatrix}, \tag{2.2}$$

where $\mathbf{x} \in \mathbb{R}^2, u \in \mathbb{R}$. The double integrator system can be used to model an acceleration-driven system, where the control input u is the acceleration, x_1 and x_2 are the position and velocity, respectively. Similar to single integrator systems, double integrator systems can be generalized to $\mathbf{x} \in \mathbb{R}^{2\times d}, \mathbf{u} \in \mathbb{R}^d$, In Chapter 3, we will use double integrator system to model an acceleration-controlled d-degree-of-freedom (DOF) robot manipulator.

Remark 2.2.1 (Continuous-time and Discrete-time Dynamical Systems). The system (2.1) is often referred to as a *continuous-time* dynamical system. This is because the state and control input are defined over a continuous variable t (provided that a solution to differential equation exists). There is another class of dynamical system called *discrete-time* dynamical system, where the state and control input are only defined at discrete time steps, e.g., $\{0, 1, 2, \ldots\}$. A discrete-time dynamical system can be described in the following form,

$$\mathbf{x}_{k+1} = g(\mathbf{x}_k, \mathbf{u}_k), \tag{2.3}$$

where $\mathbf{x}_k \in \mathbb{R}^n$ and $\mathbf{u}_k \in \mathbb{R}^m$ the value of state $\mathbf{x}$ and control input $\mathbf{u}$ at time step k, respectively. A discrete-time system (2.3) can be viewed as a deterministic special case of the Markovian transition dynamics that we will discuss in Section 2.3. In practice, even when the system dynamics (2.1) is continuous-time, e.g., when it is given by physics, the control input is commonly synthesized at *discrete* time steps, which gives a discrete-time system,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \int_{k\cdot\delta t}^{(k+1)\cdot\delta t} f(\mathbf{x}(\tau), \mathbf{u}_k)\, d\tau, \tag{2.4}$$

where $\mathbf{x}(\cdot)$ is the solution to (2.1), δt is the sample time, and x_k corresponds to $\mathbf{x}(k \cdot \delta t)$. It is worth noting that the system above (2.4) can be *approximated* via the Euler method,

$$\mathbf{x}_{k+1} \approx \mathbf{x}_k + f(\mathbf{x}_k, \mathbf{u}_k) \cdot \delta t. \tag{2.5}$$

There is a rich literature on control of discrete-time systems Franklin et al. [1998], Ogata [1995]. But for the rest of this section, we will be focusing on continuous-time systems in the form of Eq. (2.1). This means that, for the sake of simplicity, we assume that we can synthesize our control input as a continuous-time signal.

2.2.2 Stability

Central to control theory is the notion of *stability*. Intuitively, stability means that something would not "blow up" over time. But there are a lot of flavors of stability that has been studied in control theory, for example, whether the state of the system can stay close to a particular point if it starts near the point, or whether the state of the system will converge to a particular point (or set) no matter where it starts, or the norm of the state stays bounded given bounded disturbance[2], etc. Though no matter which flavor of stability we consider, we are always reasoning about the *long-term* behavior of the system, either whether certain property holds throughout state trajectories, or the behavior of the system as time goes to infinity.

Although sometimes it is possible to establish whether a "system" is stable[3], for general nonlinear systems in the form of Eq. (2.1), stability is not a "global" property of a system, e.g., a system may be converging to a point if initialized at its neighborhood while blowing up to infinity if initialized somewhere else. It makes more sense of analyze the stability of certain points or sets (e.g., periodic orbits). One common example is the stability of *equilibrium points*. An equilibrium point is a point where the system stays at that point if initialized at it. Formally, it is defined as the following.

Definition 2.2.1 (Equilibrium Point)**.** A point $\mathbf{x}_e \in \mathbb{R}^n$ is an *equilibrium point* of a system (2.1) under control input $\mathbf{u}$ if $f(\mathbf{x}_e, \mathbf{u}) = 0$.

---

[2]This corresponds to the notion of input-to-state stability (ISS) that will not be covered in this thesis.

[3]For example, stability is a global property for linear systems in the form of $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$.

We can then define the stability of an equilibrium point $\mathbf{x}_e$ in the sense of Lyapunov. Below we define three levels of stability, each stronger than the previous one.

Definition 2.2.2 ((Lyapunov) Stability)**.** An equilibrium point $\mathbf{x}_e$ of system (2.1) under control input $\mathbf{u}$ is

- *stable* if, for each $\epsilon > 0$, there is $\delta > 0$ such that $\|\mathbf{x}(0) - \mathbf{x}_e\| < \delta \Rightarrow \|\mathbf{x}(t) - \mathbf{x}_e\| < \epsilon, \forall t > 0$.

- *asymptotically stable* if it is stable, and there exists $\delta_0 > 0$ such that $\|\mathbf{x}(0) - \mathbf{x}_e\| < \delta_0 \Rightarrow \lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}_e$.

- *globally asymptotically stable* if it is stable, and for all $\mathbf{x}(0) \in \mathbb{R}^n$, $\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}_e$.

Let $\mathcal{B}_r(\mathbf{x}_e)$ with some $r > 0$ to denote the open ball centered at $\mathbf{x}_e$ with radius r, i.e., $\mathcal{B}_r(\mathbf{x}_e) \coloneqq \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{x}_e\| < r\}$. The notion of (Lyapunov) *stability* means that, for any $\epsilon > 0$, we can always find another ball of (a smaller) radius $\mathcal{B}_\delta(\mathbf{x}_e)$, such that as long as the system starts within the ball, $\mathbf{x}(0) \in \mathcal{B}_\delta(\mathbf{x}_e)$, the state stay within the ball $\mathcal{B}_\epsilon(\mathbf{x}_e)$ thereafter. *Asymptotic stability* further requires that there exists $\delta_0 > 0$ such that, if the system is initiated within $\mathcal{B}_{\delta_0}(\mathbf{x}_e)$, the state will converge to $\mathbf{x}_e$ as $t \to \infty$. Finally, *globally asymptotic stability* requires that $\mathbf{x}(t) \to \mathbf{x}_e$ for any initial state $\mathbf{x}(0)$.

If we want the system to eventually[4] converge to a certain state, e.g., reaching the goal, then we usually need *asymptotic stability* on top of *stability*. For example, a trivial system $\dot{\mathbf{x}} \equiv 0$ is stable, as for any ϵ, we can choose any $\delta \leq \epsilon$ such that $x(t) \equiv x(0) \in \mathcal{B}_\delta(\mathbf{x}_e) \subseteq \mathcal{B}_\epsilon(\mathbf{x}_e)$. However, the state does not ever move "towards" $\mathbf{x}_e$. It is important to note that stability and asymptotic stability are *local* properties, since it only concerns what happen if the system starts in a neighborhood around the equilibrium point $\mathbf{x}_e$, which could be arbitrarily small. This is why *global* asymptotic asymptotic stability can be useful in some cases, as it ensures global convergence. However, it is often harder to establish global

[4]In some sense, asymptotic stability is also a weak notion, as it does care about the *rate* of convergence. There are stronger notions including exponential stability [Khalil and Grizzle, 2002] and finite-time stability [Bhat and Bernstein, 2000].

asymptotic stability. For example, if a system has multiple equilibrium points, then any of the equilibrium point cannot be globally asymptotically stable.

2.2.3 Lyapunov Functions

Lyapunov function is a useful tool to establish the stability (and other stronger notions of stability) of an equilibrium point. Lyapunov functions are powerful, as they provide an easy-to-validate necessary condition to stability, which is a *long-term* behavior, based on *instantaneous* quantities.

Below we state two Theorems, which are quite similar, that use Lyapunov functions (with slightly different properties) to establish stability, asymptotic stability and globally asymptotic stability.

Theorem 2.2.1 (Lyapunov Stability Theorem [Khalil and Grizzle, 2002])**.** *Let* $\mathbf{x}_e \in \mathbb{R}^n$ *be an equilibrium point of the system* (2.1) *under control input* $\mathbf{u}$*. Let* $V : \mathbb{R}^n \to \mathbb{R}$ *be a continuously differentiable function such that*

- $V(\mathbf{x}_e) = 0$ *and* $V(\mathbf{x}) > 0$ *for all* $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{x}_e\}$*,*
- $\dot{V}(\mathbf{x}) \leq 0$ *for all* $\mathbf{x} \in \mathbb{R}^n$*.*

Then $\mathbf{x} = \mathbf{x}_e$ *is stable. Further, if*

$$\dot{V}(\mathbf{x}) < 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{x}_e\},$$

then $\mathbf{x} = \mathbf{x}_e$ *is asymptotic stable.*

Theorem 2.2.2 (Barbashin-Krasovskii Theorem [Khalil and Grizzle, 2002])**.** *Let* $\mathbf{x}_e \in \mathbb{R}^n$ *be an equilibrium point of the system* (2.1) *under control input* $\mathbf{u}$*. Let* $V : \mathbb{R}^n \to \mathbb{R}$ *be a continuously differentiable function such that*

- $V(\mathbf{x}_e) = 0$ *and* $V(\mathbf{x}) > 0$ *for all* $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{x}_e\}$*,*
- $\dot{V}(\mathbf{x}) < 0$ *for all* $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{x}_e\}$*,*

- $\|x\| \to \infty \Rightarrow V(\mathbf{x}) \to \infty$.[5]

Then $\mathbf{x} = \mathbf{x}_e$ *is globally asymptotically stable.*

Another commonly used condition for establishing asymptotic stability is LeSalle's invariance principle.

Theorem 2.2.3 (LeSalle's Invariance Principle [Khalil and Grizzle, 2002])**.** *Let* $\Omega \subset \mathbb{R}^n$ *be a compact set that is forward invariant with respect to* (2.1) *under control input* $\mathbf{u}$*. Let* $V : \mathbb{R}^n \to \mathbb{R}$ *be a continuously differentiable function such that* $\dot{V}(\mathbf{x}) \leq 0$ *in* Ω*. Let* $\mathcal{E}$ *be the set of all points in* Ω *such that* $\dot{V}(\mathbf{x}) = 0$*. Let* $\mathcal{C}$ *be the largest invariant set in* $\mathcal{E}$*. Then every solution starting in* Ω *approaches* $\mathcal{C}$ *as* $t \to \infty$*.*

LeSalle's invariance principle provides another way to establish asymptotic stability when it is hard to construct a Lyapunov function V such that $V(\mathbf{x}) < 0$ for all $\mathbf{x} \neq \mathbf{x}_e$. For example, consider the double integrator system (2.2) under control input $u = -x_1 - x_2$. With the Lyapunov function candidate $V(\mathbf{x}) = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2$, we have

$$\dot{V}(\mathbf{x}) = x_1\dot{x}_1 + x_2\dot{x}_2 = x_1x_2 + x_2(-x_1 - x_2) = -x_2^2 \leq 0.$$

With Lyapunov stability theorem (Theorem 2.2.1), we can conclude that the equilibrium point $\mathbf{x} = \mathbf{0}$ is a stable equilibrium point, but we can't show that it is asymptotic stable (it is in fact asymptotically stable). However, with LeSalle's invariance principle (Theorem 2.2.3), since $\{\mathbf{0}\}$ is the only[6] invariant set in $\{\mathbf{x} : x_2 = 0\}$, we can establish that $\mathbf{x} = \mathbf{0}$ is indeed asymptotic stable.

2.3 Markov Decision Processes and Reinforcement Learning

In dynamical system (2.1), the state evolves in a deterministic manner. In other words, the system is initialized to a (deterministic) initial state and it always produces the same trajectory if initialized at the same point. However, there is a different school of thoughts that consider the world as a fundamental stochastic system. This means that instead of

[5]This condition is often referred to as V being *radially unbounded*.

[6]Otherwise, we would have $\dot{x}_2 \neq 0$, and we can't stay in $\{\mathbf{x} : x_2 = 0\}$.

having a deterministic initial state, we have a *distribution* over all possible initial states. The dynamics is also stochastic, which means that we don't always land at the same future states even given at the same the state taking the same action, for example, when there are noises in the dynamics. Therefore, it is natural to model the control problem as a *stochastic process*, where the system state is a *random variable*.

Markov decision process (MDP) is then created a model for control, or sequential decision making, problems. It has an explicit objective, which is to maximize the expected (discounted) cumulative reward, which gives a notion of optimality. Reinforcement learning (RL) studies the problem of finding the optimal policy of an unknown MDP[7]. We refer the readers to Sutton and Barto [2018] for a detailed introduction to MDP and RL. We will provide references to deep online and offline RL algorithms later in this section.

2.3.1 Markov Decision Processes

A *Markov decision process* (MDP) is used to model a *sequential decision making* problem. In a sequential decision making problem, there is an *agent* which interacts with an *environment*. At each step t, the agent observes *state* s_t from the environment, chooses an action a_t, and receives the reward r_t associated the action[8].

An MDP considers a special case of sequential decision making problem where the state satisfies the *Markov* property. This means that the *future* states[9] are conditionally independent of *past* states and actions given the current state.

An MDP is denoted as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, P, \gamma, d_0)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action space, and R is the reward function. Without loss of generality, we assume $R : \mathcal{S} \times \mathcal{A} \to [0, 1]$. The transition probability is given by $P : (\mathcal{S} \times \mathcal{A}) \to \Delta(\mathcal{S})$, where $P(s'|s, a)$ is the probability of next state being s' given current state s and action a. $\gamma \in [0, 1)$ is the discount factor, which determines the present value of future rewards[10]. The initial

[7] A big part of current field of reinforcement learning actually originated from optimal control, which doesn't necessarily study MDPs. But now, it is standard to define RL in the context of sovling MDPs.

[8] One may notice that discrete time dynamical system (2.3) also fits into this process, except that rewards are not explicitly modeled there.

[9] And current and future rewards as well.

[10] It may be helpful thinking about inflation.

state distribution is given by $d_0 \in \Delta(\mathcal{S})$.

A *policy* is a map from a state to a distribution over actions, denoted $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. Given an MDP, some policies are "better", some are "worse". This "quality" of policies is given by the expected discounted cumulative reward of running the policy, starting from an initial state sampled from d_0, and following the transition probability P of the MDP $\mathcal{M}$. This gives us the *objective function* associated with the MDP $\mathcal{M}$,

$$\begin{aligned} J(\pi) &\coloneqq \mathbb{E}_{s_0 \sim d_0, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t)} \left[\sum_{t=0}^{\infty} \gamma^t \, R(s_t, a_t) \right] \\ &\overset{\text{denoted}}{=} \mathbb{E}_{\pi, P} \left[\sum_{t=0}^{\infty} \gamma^t \, R(s_t, a_t) \big| s_0 \sim d_0 \right]. \end{aligned} \tag{2.6}$$

In the second line of (2.6), we slightly abuse notations to shorten the expression of the expectation. We will use a similar notation later on when defining other quantities. The objective function (2.6) then defines an *optimal* policy, i.e., the policy which maximizes the objective $\pi^\star \coloneqq \arg\max_\pi J(\pi)$.

There are a few quantities that often becomes handy when doing analysis on an MDP. Given a policy π, its *state value function* and *state-action value function* are given by

$$\begin{aligned} V^\pi(s) &\coloneqq \mathbb{E}_{\pi, P} \left[\sum_{t=0}^{\infty} \gamma^t \, R(s_t, a_t) \big| s_0 = s \right], \\ Q^\pi(s, a) &\coloneqq \mathbb{E}_{\pi, P} \left[\sum_{t=0}^{\infty} \gamma^t \, R(s_t, a_t) \big| s_0 = s, a_0 = a \right]. \end{aligned} \tag{2.7}$$

One can see that $V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}[Q^\pi(s, a)]$ and $Q^\pi(s, a) = R(s, a) + \gamma \, \mathbb{E}_{s' \sim P(\cdot|s,a)}[V(s')]$. Finally. we have $J(\pi) = \mathbb{E}_{s \sim d_0}[V^\pi(s)]$.

The *optimal state and state-action value functions*, denoted $V^\star(s)$ and $Q^\star(s)$, are given by, $V^\star(s) \coloneqq \max_\pi V^\pi(s)$ and $Q^\star(s, a) \coloneqq \max_\pi Q^\pi(s, a)$. The optimal state and state-action value functions satisfy the following *backup* rule (assuming that $\mathcal{A}$ is compact), known as the *Bellman optimality equation*,

$$V^\star(s) = \max_{a \in \mathcal{A}} \; Q(s, a) = \max_{a \in \mathcal{A}} \; \left\{ R(s, a) + \gamma \, \mathbb{E}_{s' \sim P(\cdot|s,a)}[V^\star(s')] \right\}. \tag{2.8}$$

Another important quantity is the *average state-action visitation* of a policy π, $d^\pi(s, a)$, also known as its *occupancy measure*. It represents how likely the policy will visit a state-

action pair (s, a), where future visitation are discounted given the discount factor γ,

$$d^\pi(s,a) \coloneqq (1-\gamma)\,\mathbb{E}_{\pi,P}\left[\gamma^t\,\mathbb{1}(s_t = s, a_t = a)\middle| s_0 \sim d_0\right]. \tag{2.9}$$

The means that $J(\pi) = \frac{1}{1-\gamma}\mathbb{E}_{(s,a)\sim d^\pi(s,a)}[R(s,a)]$.

Notations For a function $f : (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$ and policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, we denote $f(s,\pi) \coloneqq \mathbb{E}_{a\sim\pi(\cdot|s)}[f(s,a)]$. Similarly, for a function $f : \mathcal{S} \to \mathbb{R}$ and distribution $d \in \Delta(\mathcal{S})$, we denote $f(d) \coloneqq \mathbb{E}_{s\sim d}[f(s)]$. This allows us to succinctly write, e.g., $V^\pi(s) = Q^\pi(s,\pi)$ and $J(\pi) = V^\pi(d_0)$.

For any function $f : (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$ and policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, we define the *Bellman operator* $\mathbb{B}^\pi$ and *transition operator* $\mathbb{P}^\pi$ as

$$\begin{aligned}(\mathbb{B}^\pi f)(s,a) &\coloneqq R(s,a) + \gamma\,\mathbb{E}_{s'\sim P(\cdot|s,a)}[f(s',\pi)],\\ (\mathbb{P}^\pi f)(s,a) &\coloneqq \gamma\,\mathbb{E}_{s'\sim P(\cdot|s,a)}[f(s',\pi)].\end{aligned} \tag{2.10}$$

In particular, Q^π is the only fixed point of operator $\mathbb{B}^\pi$, i.e., Q^π is the only function such that $Q^\pi = \mathbb{B}^\pi\,Q^\pi$.

To *solve* an MDP $\mathcal{M}$, we need to find its optimal policy $\pi^\star$. Given a known MDP, if the state and action spaces are finite, algorithms like value iteration, policy iteration, modified policy iteration, etc. [Sutton and Barto, 2018] can be used to find the optimal policy. However, when given an *unknown* MDP, we need to resort to *reinforcement learning* to find the optimal policy.

2.3.2 Online Reinforcement Learning

Online reinforcement learning (RL), or sometimes just being referred to as reinforcement learning (RL), seeks to find the optimal policy $\pi^\star$ through *interacting* with the unknown MDP. For online reinforcement learning, new experiences are collected through interacting with the environment, i.e., the unknown MDP as learning proceeds, in other words, as the policy and/or models are updated.

Numerous online RL algorithms have been proposed in the literature, for continuous [Lillicrap et al., 2016, Haarnoja et al., 2018, Schulman et al., 2017] and (finite) discrete action

spaces [Mnih et al., 2015, Hessel et al., 2018]. Based on whether the algorithm explicitly constructs an MDP, i.e., a model, online RL algorithms can be considered *model-based* Sutton [1990], Williams et al. [2017], Janner et al. [2019], Bhardwaj et al. [2020] or *model-free*, e.g., Mnih et al. [2015], Lillicrap et al. [2016], Schulman et al. [2017], Haarnoja et al. [2018]. Within the online RL algorithm, a policy can be constructed either *explicitly*[Lillicrap et al., 2016, Haarnoja et al., 2018, Schulman et al., 2017] or *implicitly* by a state-action value function [Mnih et al., 2015, Hessel et al., 2018] or a model [Williams et al., 2017, Bhardwaj et al., 2020]. For algorithms that explicitly model the policy, policy updates can be done either through optimizing a value function [Lillicrap et al., 2016, Haarnoja et al., 2018], policy gradient [Williams, 1992, Kakade, 2001, Schulman et al., 2017] or zeroth order optimization [Rubinstein, 1997, Hansen and Ostermeier, 2001, Mania et al., 2018].

An important concept in online RL is *on-policy* or *off-policy* learning. For *on-policy* learning, policy and/or model updates can only be done with experience of the current policy and model. This means that new experiences need to be collected after every single updates. Typical examples of on-policy RL algorithms include REINFORCE [Williams, 1992], natural policy gradient [Kakade, 2001], augmented random search (ARS) [Mania et al., 2018], and proximal policy optimization (PPO) [Schulman et al., 2017]. On-policy RL algorithms are generally considered to behave more stably, but has poor sample efficiency, i.e., a lot of experiences to be collected to learn a reasonable policy. In Chapter 5, we will explore learning structured robot policies using PPO [Schulman et al., 2017], a popular on-policy reinforcement learning algorithm for continuous control.

By contrast, *off-policy* algorithms can leverage experiences collected by other policies (notably policies from the previous iterations) to improve the policy and/or model. Therefore, off-policy RL algorithms are believed to be more sample efficient. Popular off-policy RL algorithms include soft actor-crtic (SAC) [Haarnoja et al., 2018], deep Q network (DQN) [Mnih et al., 2015], and most model-based RL algorithms. As we will discuss in Section 2.3.3, there are important connections between off-policy RL algorithms and *offline* RL algorithms. In fact, a lot of offline RL algorithms, e.g., [Fujimoto and Gu, 2021, Kumar et al., 2020a, Cheng et al., 2022] are adapted from off-policy RL algorithms.

2.3.3 Offline Reinforcement Learning

Online RL requires exploratory interactions with the environment to improve the policy and/or the model. These exploratory interactions often can not be afforded in risk-sensitive applications, such as robotics [Ibarz et al., 2021] and healthcare [Gottesman et al., 2018]. In these domains, it is more practical to consider an offline setting [Levine et al., 2020], where data is collected by policies satisfying certain criteria.

In contrast to online RL, *offline RL* studies the problem solving an unknown MDP $\mathcal{M}$ given *offline* data. An offline dataset is a collection of transition tuples,

$$\mathcal{D} \coloneqq \{(s, a, r, s') | (s, a) \sim \mu(\cdot, \cdot), r = R(s, a), s' \sim P(\cdot|s, a)\}.$$

where μ denotes the state-action data distribution induced by the data collection process. Recall that R is the reward function and P captures the MDP's transition dynamics. It is typically assumed that the offline dataset $\mathcal{D}$ is *compatible* with the unknown task MDP, i.e., the reward function R and transition dynamics P are consistent with the unknown MDP[11].

Technically, any *offline policy* online RL algorithms, e.g., fitted-Q iterations (FQI) [Riedmiller, 2005], deep-Q network (DQN) [Mnih et al., 2015], deep deterministic policy gradient (DDPG) [Lillicrap et al., 2016], soft actor-critic (SAC) [Haarnoja et al., 2018], and most model-based RL algorithms, can be directly used for offline RL. The practical challenge is that, when the data distribution μ does not have full coverage over the state-action space, these algorithms may over-estimate the value at state-action pairs not covered by data, and produces policies that are arbitrarily bad. Although over-estimation is already problematic for online RL, it becomes a more serious issue for offline RL, as such over-estimate can not be corrected due to the lack of online interactions.

Modern offline RL algorithms adopt pessimism to address the issue of policy learning when μ does not have the full state-action space as its support. The basic idea is to optimize for a performance lower bound that penalizes actions leading to out-of-support future states.

[11] In Chapter 8, we will look into when the data *reward*does not match the true reward for the unknown MDP. The robust offline RL literature has studied when the transition dynamics and/or reward in data mismatches with the unknown MDP.

It can be implemented in a model-free [Cheng et al., 2022, Xie et al., 2021, Kostrikov et al., 2021, Kumar et al., 2020a, Jin et al., 2021, Liu et al., 2020, Rashidinejad et al., 2021, Wu et al., 2019, Fujimoto and Gu, 2021] or a model-based [Uehara and Sun, 2022, Xie et al., 2022, Yu et al., 2020b, Kidambi et al., 2020, Rigter et al., 2022] manner. Such a penalty can take the form of behavior regularization [Kostrikov et al., 2021, Fujimoto and Gu, 2021, Wu et al., 2019], negative bonuses to discourage visiting less frequent state-action pairs [Jin et al., 2021, Rashidinejad et al., 2021, Yu et al., 2020b], pruning less frequent actions [Liu et al., 2020, Kidambi et al., 2020], adversarial training [Cheng et al., 2022, Xie et al., 2021, Uehara and Sun, 2022, Xie et al., 2022, Rigter et al., 2022] or value penalties in modified dynamic programming [Kumar et al., 2020a, Yu et al., 2021].

In Chapter 7, we will investigate when robotics data has a certain structure which does not comply with the assumptions of offline RL. We will draw the connection between offline RL and offline imitation learning from observation (ILfO), another offline learning paradigm, and extend offline RL to scenarios where reward and action are missing from some transitions in the offline data. In Chapter 8, we will discuss implicit structure in robotics data, which is naturally induced by its collection process. We will investigate the behavior of offline RL under inaccurate or incorrect data reward when data has such structure.

2.4 Comparison between Control and RL Terminologies

Readers may notice that in Section 2.2 and Section 2.3, similar concepts are defined slightly differently and denoted with different symbols. These terminologies are summarized in Table 2.3. Hopefully this table can help readers connect and distinguish these similar concepts.

There are exceptions to the comments in Table 2.3. For example, in stochastic control theory, the state and dynamics (sometimes the control input as well) are also stochastic. Moreover, a deterministic process like a discrete-time system Eq. (2.3) can be modeled as an MDP, where s is a deterministic variable, and the transition P is also deterministic. This is in fact why in Part I, we are able to use control theoretical tools to analyze the systems therein, while also able to train RL policies for these systems (since they are also MDPs).

Control	MDP & RL	Comments
state variable $\mathbf{x}$	state s	$\mathbf{x}$ is commonly treated as deterministic, while s is often considered as a random variable.
control input $\mathbf{u}$	action a	Similar to state.
controller $\mathbf{u}$	policy π	In control theory, it is common to abuse notation and use $\mathbf{u}$ for controller along with the control input. $\mathbf{u}$ is often considered a map to deterministic values, while π often maps to a distribution over actions.
dynamics $f(\mathbf{x}, \mathbf{u})$	transition $P(\cdot\|s, a)$	$f(\mathbf{x}, \mathbf{u})$ is by default a deterministic value, while $P(s'\|s, a)$ is a probability distribution.

Table 2.3: Comparisons between control theory and RL terminologies

Part I

LEVERAGING STRUCTURE IN ROBOTICS PROBLEM

Chapter 3

RIEMANNIAN MOTION POLICIES

In this chapter, we will go over the basics of controlling a robot manipulator, and introduce Riemannian motion policies (RMPs), a framework for multi-objective robot control. Readers are recommended to check out textbooks, e.g., [Choset et al., 2005, Siciliano et al., 2010, Lynch and Park, 2017], for more thorough introduction of control of robot manipulators. More detailed descriptions of RMPs can be found in [Ratliff et al., 2018, Cheng et al., 2018, 2020, Li et al., 2019a, 2021]. Section 3.2 of this chapter is adapted from [Li et al., 2019a, 2021].

3.1 Operational Space Control

In Part I of this thesis, we focus on *acceleration-based* control of robotics systems. We have seen a simplified 1-d system in (2.2) from Chapter 2. Typically this type of control problems arises when one wishes to reactively generate smooth reference trajectories for a low-level tracking controller, or wants to control a robot that is fully actuated and feedback linearized, e.g., by an inverse dynamics model[1] [Khalil and Grizzle, 2002, Siciliano et al., 2010]. In this section, we will discuss a framework for doing acceleration-based control for robots called *operational space control* (OSC).

Although OSC is initialized proposed by Khatib [1987] for torque-based control of robot manipulators, i.e., the control input is the torque applied to, e.g., each joint of a robot manipulator. The term has been later extended to acceleration-based and velocity based systems [Nakanishi et al., 2008]. In this thesis, we will focus on acceleration-based control.

[1]The term "inverse dynamics" could refer to different concepts in different fields. In the control literature, The dynamics is often as opposed to kinematics. The forward dynamics is defined to be the map from generalized coordinate and velocity, and applied forces or torques to generalized acceleration, while inverse dynamics is defined as the map from generalized coordinate, velocity, and acceleration to applied forces and torques. However, in an MDP context, the inverse dynamics often refers to the map from state and next state to action (which may or may not exist). Here we refer to the former.

The extension to torque-based controller is straightforward if given a inverse dynamics model. It is also possible to extend Part I of this thesis to velocity-based systems.

3.1.1 Configuration Space and Task Space

Before we discuss operational space control, we need to first introduce a few important concepts for robot control. We would like to note that some definitions are adapted from Choset et al. [2005].

First, a *configuration* of a robot is a complete specification of the position of every point of the system. A representation of a configuration of a robot is often denoted as $\mathbf{q}$. For example, for the planner three link robot shown in Fig. 3.1a, the joint angles of the three joints $\mathbf{q} = [q_1 \quad q_2 \quad q_3]^\top$ is a sufficient representation of the configuration of the three-link robot. The *configuration space*, denoted $\mathcal{C}$ of a robot is the space of all possible configurations of a robot. Suppose each joint angles of the three link robot is in the range of (known as *joint limits*[2]), e.g., $(-\pi, \pi)$, the configuration space of the robot can be represented by the 3-d space $(-\pi, \pi)^3$.

In this thesis, we focus on robots whose configuration space $\mathcal{C}$ is a d-dimensional smooth manifold that admits a global[3] coordinate. We call the coordinates *generalized coordinates* of the robot, denoted $\mathbf{q}$. The *generalized velocities* $\dot{\mathbf{q}}$ are the time derivatives of generalized coordinates, and *generalized accelerations* $\ddot{\mathbf{q}}$ are the time derivatives of the generalized velocities. Since we are concerned with acceleration-based control, we assume that the *control input* to the robot is the generalized acceleration, i.e., $\mathbf{u} = \ddot{\mathbf{q}}$. The *state* of the system is the concatenation of generalized position and velocity $(\mathbf{q}, \dot{\mathbf{q}})$. We also slightly abuse MDP notations (to continuous-time systems) and denote the acceleration-based *policy* as π, i.e., $\mathbf{u} = \ddot{\mathbf{q}} = \pi(\mathbf{q}, \dot{\mathbf{q}})$.

In these motion control problems, the desired behavior[4] of a task is often not directly described in the generalized coordinates $\mathbf{q}$, but in terms of another set of task coordinates

---

[2]The joint limits make things simpler since otherwise, the configuration space would be $\mathcal{S}^3_1$, which does not admit a global coordinate.

[3]The later derivation also applies to local coordinates under proper change of coordinates.

[4]For example, as given by the reward function.

$\mathbf{x} \in \mathbb{R}^m$ that are related to the generalized coordinates through a nonlinear map $\psi : \mathbb{R}^d \to \mathbb{R}^m$, i.e.,

$$\mathbf{x} = \psi(\mathbf{q}). \tag{3.1}$$

We call this map ψ the *task map* and refer to the image manifold of $\mathcal{C}$ under ψ as the *task space*, which is denoted as $\mathcal{T}$. In robotics, the task map ψ is often given by the *forward kinematics* of the robot. For example, in controlling of the three link robot in Fig. 3.1a, we may want the tip of the (third link of the) robot to reach the green target. This task coordinates $\mathbf{x} \in \mathbb{R}^2$ are given by the position of the tip.

3.1.2 Operational Space Control

Operational space control (OSC) [Khatib, 1987, Nakanishi et al., 2008] aims to find a controller (or equivalently, a policy) on the configuration space $\mathcal{C}$ such that the robot shows desirable behavior in the task space $\mathcal{T}$. Here, we assume that the dimension of the configuration space is larger than the dimension of the task space, i.e., $d \geq m$.

For example, we may want to the tip of the three link robot in Fig. 3.1a to reach the (fixed) green target or follow a certain trajectory. If we were able to directly control the acceleration of the tip $\ddot{\mathbf{x}}$, then, we can apply the following controller (on the *task space*),

$$\ddot{\mathbf{x}} = -\mathbf{K}_p(\mathbf{x} - \mathbf{x}^d) - \mathbf{K}_d(\dot{\mathbf{x}} - \dot{\mathbf{x}}^d) + \ddot{\mathbf{x}}^d, \tag{3.2}$$

where $\mathbf{x}^d, \dot{\mathbf{x}}^d$, and $\ddot{\mathbf{x}}^d$ are the position, velocity, and acceleration of the target on the task space, and $\mathbf{K}_p$ and $\mathbf{K}_d$ are diagonal matrices with positive diagonal entries. However, this does not directly give us the control input on the *configuration space*, i.e., what joint accelerations should be applied.

To find the appropriate configuration space control input, we need to first understand how generalized position, velocity, and acceleration translate into position, velocity, and acceleration on the task space. The map from generalized position and task space position is given directly by (3.1). The maps for velocity and acceleration are given by taking the

time derivatives of (3.1),

$$\dot{\mathbf{x}} = \mathbf{J}_\phi(\mathbf{q})\,\dot{\mathbf{q}}, \tag{3.3}$$

$$\ddot{\mathbf{x}} = \mathbf{J}_\phi(\mathbf{q})\,\ddot{\mathbf{q}} + \dot{\mathbf{J}}_\phi(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}, \tag{3.4}$$

where $\mathbf{J}_\phi$ is the Jacobian of the task map $\mathbf{J}_\phi(\mathbf{q}) = {}^{\partial\phi}/_{\partial\mathbf{q}}$, $\dot{\mathbf{J}}_\phi(\mathbf{q},\dot{\mathbf{q}})$ is the time derivative of the Jacobian $\mathbf{J}_\phi(\mathbf{q})$.

It can be noted that, when $\mathbf{q},\dot{\mathbf{q}}$, and $\ddot{\mathbf{x}}$ are given, (3.4) defines a set of linear equations of $\ddot{\mathbf{q}}$. The desired configuration space control input is then given by the solution of the least-squares problem,

$$\ddot{\mathbf{q}} = \pi(\mathbf{q},\dot{\mathbf{q}}) = \mathbf{J}_\phi(\mathbf{q})^\dagger\big(\ddot{\mathbf{x}} - \dot{\mathbf{J}}_\phi(\mathbf{q})\,\dot{\mathbf{q}}\big). \tag{3.5}$$

Since $d \geq m$, we know that under the controller (3.5), the robot follows the controller (3.2) on the task space. In later parts of this chapter, we will generalize the above observation to cases when we have multiple task spaces of interest (which we will call each of them a subtask space).

We now summarize how the configuration space control input $\ddot{\mathbf{q}}$ is obtained given state (in the configuration space), i.e., generalized position and velocity, $(\mathbf{q},\dot{\mathbf{q}})$.

1. Compute the task space position and velocity $(\mathbf{x},\dot{\mathbf{x}})$, given by (3.1) and (3.3).

2. Compute the task space acceleration $\ddot{\mathbf{x}}$ given the task space controller, e.g., (3.2).

3. Obtain the control input on the configuration space $\ddot{\mathbf{q}}$ through (3.5).

Note that this involves a "forward" pass (step 1) from the configuration space to the task space, follwed by a "backward" pass (step 3) from the task space back to the configuration space. The Riemannian motion policy (RMP) framework that we will soon introduce also follows a similar structure for computing the control input.

Remark If we actually run the controller (3.5) on our robot, we often notice that the robot will have a lot of redundant motion. For example, the end effector of the robot would reach the fixed target and stop there, but the joints may keep rotating while keeping the

end effector position fixed. This is because (3.4) is an *under-defined* linear system, due to the fact that $d \leq m$. Therefore, there are infinite number of configuration space controller that would lead to the same task space behavior. What the robot is doing in the *null space* of the Jacobian $\mathbf{J}_\phi(\mathbf{q})$ is not concerned by (3.4). In practice [Nakanishi et al., 2008], it is common to add a damping effect to the null space of the Jacobian $\mathbf{J}_\phi(\mathbf{q})$,

$$\ddot{\mathbf{q}} = \pi(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{J}_\phi(\mathbf{q})^\dagger \left(\ddot{\mathbf{x}} - \dot{\mathbf{J}}_\phi(\mathbf{q})\, \dot{\mathbf{q}}\right) + \lambda \left(\mathbb{I} - \mathbf{J}_\phi(\mathbf{q})^\dagger \mathbf{J}_\phi(\mathbf{q})\right) \dot{\mathbf{q}}, \quad \text{with } \lambda \geq 0. \tag{3.6}$$

This is quite specific to when $d \geq m$. In later parts of this chapter, we do not need to worry about this as our linear system is more likely to be *over-defined* rather than under-defined.

3.2 Riemannian Motion Policies

OSC may be sufficient when there is only a single task that we want the robot to achieve, e.g., reaching the goal. However, in practice, this is rarely the case. For example, for the three link robot shown in Fig. 3.1a, the robot should also avoid collisions with the three pillars. This means that we cannot simply just consider the tip of the robot, there are a lot of other points[5] on the robot that also important when controlling the robot, and we call those points *control points*.

As shown in the above example, robotics problem are often multi-objective in nature, requiring the robot to satisfy various performance criteria. We call each of the smaller tasks, e.g., reaching a goal, avoiding collision between a point on the robot with an obstacle, etc., a *subtask*. Mathematically, this implies that the task space $\mathcal{T}$ is a composition of many subtask spaces, such as $\mathcal{T} = \prod_{k=1}^{K} \mathcal{T}_k$ in a K-objective control problem on subtask manifolds $\{\mathcal{T}_k\}_{k=1}^{K}$. These subtask coordinates are often not independent but intertwined together as the image of the common configuration space $\mathcal{C}$ under the task map ψ (in the three link robot example, moving the first joint affects the configurations all the robot links). Therefore, generally, controller (policies) for each subtask cannot be trivially combined together (e.g., through directly averaging) to generate a good policy for the robotics problem of interest.

Riemannian motion policies (RMPs) are introduced as a framework to solve multi-task

[5] Although technically we need to worry about an infinite number of points along the robot, in practice, only consider a finite number of them is sufficient.

robotics problems. In Section 3.2.1, we will introduce what an RMP is, and how the RMP framework differs from OSC. In Section 3.2.2, we will introduce RMP^2, an algorithm for synthesizing the control input in the configuration space. Finally, in Section 3.2.3, we will discuss the stability properties of RMP^2.

In this section, we focus on the optimization interpretation of the RMP framework [Li et al., 2021], which is slightly different from the originally proposed message passing framework from [Ratliff et al., 2018, Cheng et al., 2018, 2020]. This is because the optimization interpretation is easier to understand (without the need of customized data structure) and can translate to an algorithm, i.e., RMP^2, that is easier to implement and more memory efficient. Interested reader should see [Li et al., 2021] for a dedicated discussion.

3.2.1 Riemannian Motion Policies

The RMP framework considers a K-objective problem where the task space $\mathcal{T}$ is a composition of K *subtask spaces*, i.e., $\mathcal{T} = \prod_{k=1}^{K} \mathcal{T}_k$. Examples of subtasks include, reaching a goal, avoiding collision between a point on the robot with an obstacle, respecting joint limits, etc.

For the k-th subtask, let ψ_k be the *subtask map* from the configuration space coordinate, i.e., generalized coordinate, $\mathbf{q}$ to the subtask coordinate $\mathbf{x} \in \mathbb{R}^m$, where m is the dimension[6] of k-th subtask space $\mathcal{T}_k$. Similar to OSC, the subtask coordinate, velocity, and acceleration are given by,

$$\begin{aligned} \mathbf{x}_k &= \psi_k(\mathbf{q}) \\ \dot{\mathbf{x}}_k &= \mathbf{J}_k(\mathbf{q})\,\dot{\mathbf{q}} \\ \ddot{\mathbf{x}}_k &= \mathbf{J}_k(\mathbf{q})\,\ddot{\mathbf{q}} + \dot{\mathbf{J}}_k(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}}, \end{aligned} \tag{3.7}$$

where $\mathbf{J}_k(\mathbf{q})$ is the Jacobian matrix of the subtask map for the k-th subtask space, ψ_k, and $\dot{\mathbf{J}}_k(\mathbf{q}, \dot{\mathbf{q}})$ is the time-derivative of $\mathbf{J}_k(\mathbf{q})$. We define $(\mathbf{x}, \dot{\mathbf{x}}_k)$ as the *state* of subtask k. We assume that all subtask maps, Jacobians and their derivatives are locally Lipschitz continuous[7].

[6] Different subtask spaces may have different dimensions.

[7] This is for the stability analysis in Section 3.2.3.

The fundamental difference between a multi-objective problem and a single-task one (that OSC solves) is that different subtasks may conflict with each other. This means that there may not exist a configuration space control input $\ddot{\mathbf{q}}$ that perfectly satisfies all subtask requirements. For example, the goal reaching subtask may want the robot to move forward to reach a goal, while the collision avoidance subtask may instead want the robot to back up since it is getting close to an obstacle.

To resolve conflicts between subtasks, the RMP framework assigns each subtask with a state-dependent importance weight $\mathbf{M}_k : (\mathbf{x}_k, \dot{\mathbf{x}}_k) \mapsto \mathbf{M}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k) \in \mathbb{R}_+^{m \times m}$, $k \in \{1, \ldots, K\}$. The higher the importance weight is, the more important the subtask is at a given state $(\mathbf{x}_k, \dot{\mathbf{x}}_k)$. The main benefit of having a state-dependent importance weight is that it allows different relative ranking among tasks at different states. For example, the collision importance subtask can have large importance when the robot is close to an obstacle and moving towards it, while it can have a low importance weight if the robot is far from the obstacle or is moving away from it. The fact that $\mathbf{M}_k$ is a matrix allows importance to differ along different directions on $\mathcal{T}_k$. Similar to OSC, e.g. (3.2), each subtask has an acceleration-based policy (controller) $\mathbf{a}_k : (\mathbf{x}_k, \dot{\mathbf{x}}_k) \mapsto \mathbf{a}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k) \in \mathbb{R}^m$ which specifies the desired acceleration for a subtask, e.g., moving towards the goal, moving away from a obstacle, etc. A *Riemannian*[8] *motion policy* (RMP) for the k-th subtask is defined as the tuple $(\mathbf{M}_k, \mathbf{a}_k)$.

Given the subtask RMPs $\{(\mathbf{M}_k, \mathbf{a}_k)\}_{k=1}^K$, the configuration space control input is then given by the solution of the following optimization problem,

$$\ddot{\mathbf{q}} = \pi(\mathbf{q}, \dot{\mathbf{q}}) = \min_{\mathbf{a} \in \mathbb{R}^d} \quad \sum_{k=1}^{K} \frac{1}{2} \left\| \mathbf{J}_k(\mathbf{q})\, \mathbf{a} + \dot{\mathbf{J}}_k(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}} - \mathbf{a}_k \right\|_{\mathbf{M}_k}^2 . \tag{3.8}$$

Note that the Jacobians and velocities here are treated as constants in (3.8) as they are only dependent on the state (not the accelerations). Therefore, the optimization problem (3.8) is a linear least-squares problem over $\mathbf{a}$. To simplify the notations, we omit the argument to Jacobians, and define $\mathbf{c}_k \coloneqq \dot{\mathbf{J}}_k(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}$ ($\mathbf{c}$ stands for curvature). This allows us to write,

$$\ddot{\mathbf{q}} = \pi(\mathbf{q}, \dot{\mathbf{q}}) = \min_{\mathbf{a} \in \mathbb{R}^d} \quad \sum_{k=1}^{K} \frac{1}{2} \left\| \mathbf{J}_k\, \mathbf{a} + \mathbf{c}_k - \mathbf{a}_k \right\|_{\mathbf{M}_k}^2 . \tag{3.9}$$

[8]The term "Riemannian" comes from the connection between the importance weight matrix $\mathbf{M}_k$ and a Riemannian metric $\mathbf{G}_k$ on the subtask space $\mathcal{T}_k$. See Section 3.2.3 for details.

In fact, the optimization problem (3.9) has a closed-form solution:

$$\pi(\mathbf{q}, \dot{\mathbf{q}}) = \underbrace{\left(\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \mathbf{J}_k\right)^\dagger}_{\mathbf{M}^\dagger} \underbrace{\left(\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k (\mathbf{a}_k - \mathbf{c}_k)\right)}_{\mathbf{f}}, \tag{3.10}$$

where the term $\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k(\mathbf{a}_k - \mathbf{c}_k)$ is denoted $\mathbf{f}$ as it acts like a virtual force[9] $\mathbf{f} = \mathbf{M}\,\mathbf{a}$.

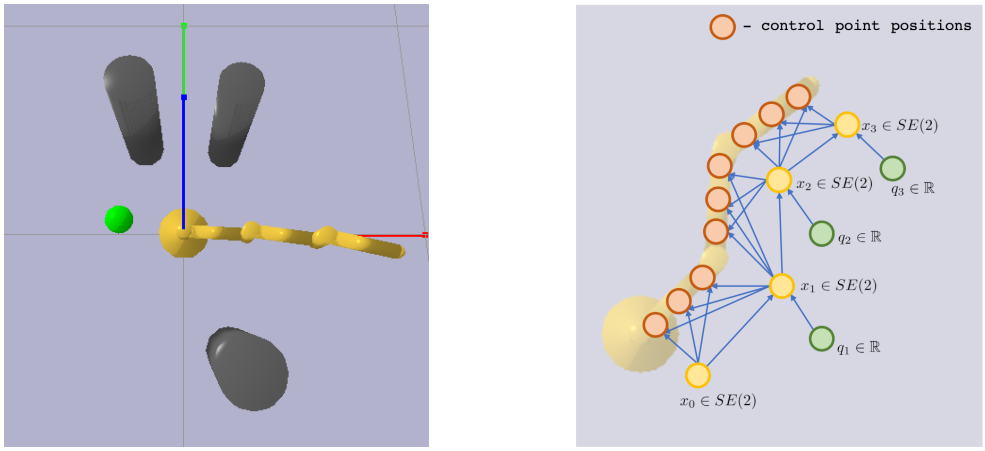
Remarks We would like to note that Cheng et al. [2018, 2020] instead define the configuration space policy $\pi(\mathbf{q}, \dot{\mathbf{q}})$ as the result of a message passing algorithm called RMPflow. But it can be shown that these two definitions are equivalent. Although the optimization problem formulation (3.9) is straightforward, we will show in Section 3.2.3 that the solution to (3.9) has desirable stability properties if all RMPs satisfy certain conditions. This is quite remarkable, as (state-dependent) blending of stable systems does not always result in a stable system [Li et al., 2019a].

3.2.2 RMP^2 Algorithm

In this section, we introduce an algorithm called RMP^2 [Li et al., 2021] that efficiently solve the optimization (3.9). RMP^2 makes use of basic operators in automatic differentiation libraries, i.e., building computational graph and back-propagating on the computational graph, to compute the solution to (3.9) given subtask maps. This makes RMP^2 simpler to implement (without needing to define customized data structure and implement message passing on the data structure) than the originally proposed message passing algorithm RMPflow [Cheng et al., 2018, 2020].

We first discuss why an efficient algorithm for solving Eq. (3.9) is necessary, despite that one could naïvely compute all necessary quantities in the closed-form solution (3.10). In practice, the number of subtasks are large. For example, to control a 7 degree-of-freedom Franka Emika Panda robot, Li et al. [2021] use around 50 control points. This means that around 150 RMPs are needed (just for collision avoidance) in an environment with 3

[9] In fact, Cheng et al. [2018] define the *natural* form of an RMP as $[\mathbf{M}, \mathbf{f}]$, where $\mathbf{f} = \mathbf{M}\,\mathbf{a}$. The RMP definition we have in the main text, i.e., $(\mathbf{M}, \mathbf{a})$ is called the *canonical* form of an RMP. We will use the natural form in Section 3.2.6.



(a) The three link robot

(b) The computational graph used by RMP^2

Figure 3.1: (a) A planar three-link robot. The robot is tasked with reaching the green target with its tip, while avoiding collision with three obstacle pillars. (b) The computation graph built by automatic differentiation libraries when computing control point positions. RMP^2 directly operates on this graph. q_i is the joint angle of the i-th joint, x_i is the pose (position and orientation) of the i-th link, and x_0 is the base pose.

obstacles. Although it is possible to compute the subtask maps $\{\psi_k\}_{k=1}^K$, as well as their Jacobians $\{\mathbf{J}_k\}_{k=1}^K$ and curvature terms $\{\mathbf{c}_k\}_{k=1}^K$ individually. This is going to be inefficient because there are a lot of *redundant* computation if these quantities are computed separately.

To better see the redundancy, let us consider the three link robot with 9 control points as shown in Fig. 3.1b. We assign three control point per link. The control points are given by linearly interpolating between both ends of the link, which can be computed by forward kinematics of the robot. Assume that we have computed the pose of both end of the last link, denoted[10] x_2 and x_3, we can get the pose of all three control points almost for free. However, if each subtask maps are treated separately, the (mostly) same computation would need to be done for three (which is the number of control points for that link) times.

[10]Here we use normal weight as x_i is in the manifold $SE(2)$ (so it is not exactly a vector in, e.g., $\mathbb{R}^3$). But this is contradict to our convention that x_i is used to denote the i-th component of a vector $\mathbf{x}$.

Algorithm 1 Jacobian-vector product [Christianson, 1992] `jvp(y,x,v)`

Input: $\mathbf{x}$, $\mathbf{v}$

Return: $(\partial_{\mathbf{x}}\mathbf{y})\,\mathbf{v}$

$\mathbf{y} \leftarrow$ `graph(x)`

$\mathbf{w} \leftarrow \mathbf{1}_m$ *// dummy variable for reverse accumulation*

$\mathbf{g} \leftarrow$ `gradient(`$\mathbf{w}^\top$`y,x)` *// numerically equals to the sum of partial derivatives*

compute Jacobian-vector product

$$(\partial_{\mathbf{x}}\mathbf{y})\,\mathbf{v} \leftarrow \texttt{gradient(}\mathbf{g}^\top\texttt{y,w)}$$

Redundancy also exists when computing the forward kinematics of the robot. For example, computing x_1 and x_2 is necessary when computing x_3. This means that computation can also be reused between computing the poses of control points on, e.g., the last link and the first link. This redundancy in subtask map evaluation also transfers to redundancy in Jacobian and curvature term evaluation by the chain rule of calculus.

To avoid redundant computation, we should keep track of the (intermediate) values already computed, and use these pre-computed values later on for other quantities that depend on these values. Fortunately, in automatic differentiation libraries, a computation graph is automatically built as functions are specified, and derivatives can be computed through message passing on the computation graph, i.e., back-propagation. RMP^2 leverages this feature of automatic differentiation libraries so that computation can re-use efficiently without the need of additional data structure and message passing routines.

Let $\mathbf{x}$ and $\mathbf{y}$ be the input and output of the computation graph, i.e., $\mathbf{y} =$ `graph(x)`. We use y (instead of $\mathbf{y}$) if the output of the graph is a scalar. RMP^2 uses the following common functionalities provided by automatic differentiation libraries:

- `gradient(`y`,x)`: the gradient operator. It computes the gradient of a scalar graph output y with respect to the graph input vector $\mathbf{x} \in \mathbb{R}^n$ through back-propagation.

- `jacobian(y,x)`: the Jacobian operator. It computes the Jacobian matrix $\partial_{\mathbf{x}}\mathbf{y} \in \mathbb{R}^{m\times n}$, where $\mathbf{y} \in \mathbb{R}^m, \mathbf{x} \in \mathbb{R}^n$, through back-propagation; `jacobian` is equivalent to

Algorithm 2 RMP^2

1: **Input:** root state $(\mathbf{q}, \dot{\mathbf{q}})$, `task_map`, `rmp_eval`

2: **Return:** motion policy $\pi(\mathbf{q}, \dot{\mathbf{q}})$

// forward pass

3: $\{\mathbf{x}_k\}_{k=1}^K \leftarrow$ `task_map(q)` *// subtask positions*

4: $\{\dot{\mathbf{x}}_k\}_{k=1}^K \leftarrow$ `jvp(`$\{\mathbf{x}_k\}_{k=1}^K$`,q,`$\dot{\mathbf{q}}$`)` *// subtask velocities*

5: $\{\mathbf{c}_k\}_{k=1}^K \leftarrow$ `jvp(`$\{\dot{\mathbf{x}}_k\}_{k=1}^K$`,q,`$\dot{\mathbf{q}}$`)` *// curvature terms*

// evaluate subtask RMPs

6: $\{(\mathbf{M}_k, \mathbf{a}_k)\}_{k=1}^K \leftarrow$ `rmp_eval(`$\{(\mathbf{x}_k, \dot{\mathbf{x}}_k)\}_{k=1}^K$`)`

// backward pass

7: $\mathbf{q}' =$ `copy(q)`, $\mathbf{q}'' =$ `copy(q)` *// copies of* $\mathbf{q}$

8: $\{\mathbf{x}'_k\}_{k=1}^K \leftarrow$ `task_map(q')`, $\{\mathbf{x}''_k\}_{k=1}^K \leftarrow$ `task_map(q'')` *// mirrored images*

9: $\mathbf{M} \leftarrow$ `jacobian(gradient(`$\sum_{k=1}^K (\mathbf{x}'_k)^\top \mathbf{M}_k \mathbf{x}''_k$`,q),q')`, *// configuration space importance*

10: $\mathbf{f} \leftarrow$ `gradient(`$\sum_{k=1}^K (\mathbf{x}'_k)^\top \mathbf{M}_k (\mathbf{a}_k - \mathbf{c}_k)$`,q)` *// configuration space virtual force*

// configuration space policy

11: $\pi(\mathbf{q}, \dot{\mathbf{q}}) \leftarrow \mathbf{M}^\dagger \mathbf{f}$

multiple calls of `gradient`.

- `jvp(y,x,v)`: the Jacobian-vector product. It computes $(\partial_{\mathbf{x}} \mathbf{y})\, \mathbf{v} \in \mathbb{R}^m$ given the graph input vector $\mathbf{x} \in \mathbb{R}^n$, the output vector $\mathbf{y} \in \mathbb{R}^m$, and an addition vector $\mathbf{v} \in \mathbb{R}^n$. The Jacobian-vector product can be efficiently realized by `gradient` (see Algorithm 1) using a technique, called reverse accumulation [Christianson, 1992] (also known as double backward). The algorithm computes Jacobian-vector product through two backward passes on the computational graph. An auxiliary all-ones vector $\mathbf{w}$ is created and the gradient with respect to $\mathbf{w}$ is tracked (line 4).

By using the `gradient`, `jacobian`, `jvp` operators, RMP^2 in Algorithm 2 efficiently computes quantities needed for evaluating (3.10). One may notice that RMP^2 follows a similiar structure as OSC, as it also consists of a forward pass, mapping configuration state

to subtask space states, a evaluation step which computes subtask policies, and a backward pass which computes the configuration space policy given the subtask policies.

In the forward pass of RMP^2 (Algorithm 2, line 3–5), it evaluates subtask maps and compute their velocities and curvature terms using Jacobian-vector products. Specifically, for the k-th subtask map $\psi_k : \mathbf{q} \mapsto \mathbf{x}_k$ (which we may think as the map from the joint space of a robot manipulator to the workspace), the velocity $\dot{\mathbf{x}}_k = \mathbf{J}_k\dot{\mathbf{q}}$ and curvature term $\mathbf{c}_k = \dot{\mathbf{J}}_k\dot{\mathbf{q}}$ on the subtask space can be computed through Jacobian-vector products:

$$\dot{\mathbf{x}}_k = \texttt{jvp}(\mathbf{x}_k, \mathbf{q}, \dot{\mathbf{q}}), \quad \text{and} \quad \mathbf{c}_k = \texttt{jvp}(\dot{\mathbf{x}}_k, \mathbf{q}, \dot{\mathbf{q}}). \tag{3.11}$$

Using $\{(\mathbf{x}_k, \dot{\mathbf{x}}_k)\}_{k=1}^K$, RMP^2 then evaluates the values of the subtask RMPs (line 6), which implicitly define the objective of the least-squares problem in (3.9). Next, in the backward pass (line 7–11), RMP^2 computes the configuration space virtual force $\mathbf{f}$ and importance weight matrix $\mathbf{M}$ in (3.10) using reverse accumulation (i.e., the technique used in `jvp` in Algorithm 1). This is accomplished by creating auxiliary variables $\mathbf{q}'$ and $\mathbf{q}''$, which have the same numerical value as $\mathbf{q}$ (but are different nodes in the computational graph of automatic differentiation), and their mirrored subtask images (line 7 and 8). The configuration space virtual force and importance weight matrix are then given by:

$$\begin{aligned} \mathbf{f} &= \texttt{gradient}\Big(\sum_{k=1}^{K} (\mathbf{x}_k')^\top \mathbf{M}_k(\mathbf{a}_k - \mathbf{c}_k), \mathbf{q}'\Big), \\ \mathbf{M} &= \texttt{jacobian}\Big(\texttt{gradient}\big(\sum_{k=1}^{K} (\mathbf{x}_k')^\top \mathbf{M}_k \mathbf{x}_k'', \mathbf{q}'\big), \mathbf{q}''\Big), \end{aligned} \tag{3.12}$$

where $\mathbf{x}_k' = \psi_k(\mathbf{q}')$, $\mathbf{x}_k'' = \psi_k(\mathbf{q}'')$, and $\mathbf{q} = \mathbf{q}' = \mathbf{q}''$.

Remark One may notice that RMP^2 (Algorithm 2) never explicitly creates the Jacobians of the subtask maps $\{\mathbf{J}_k\}_{k=1}^K$ and its time derivatives $\{\dot{\mathbf{J}}_k\}_{k=1}^K$. This is because computing Jacobians are not efficient, as it requires many backward passes on the computation graph. This gives the algorithm a worse time (and sometimes space) complexity. The readers can see [Li et al., 2021], Section D for the complexity analysis for such an algorithm.

Complexity Analysis of RMP^2

We analyze the time and space complexities of RMP^2. We show that RMP^2 has a time complexity of $O(Nbd^3)$ and a memory complexity of $O(Nd + Ld^2)$, where N is the total number of nodes of the computational graph, L is the number of leaf nodes in the computational graph, b is the maximum branching factor, and d is the maximum dimension of nodes. In comparison, the original RMPflow message passing algorithm [Cheng et al., 2018, 2020] has a time complexity of $O(Nbd^3)$ and a *worse* space complexity of $O(Nd^2 + Ld^2)$, see [Li et al., 2021] for a complexity analysis for RMPflow.

Specifically, consider a directed-acyclic-graph-structured task map with N nodes, where each node has dimension in $O(d)$ and has at most b parents. We suppose that $L \leq N$ nodes are leaf nodes, and that the automatic differentiation library is based on reverse-mode[11] automatic differentiation. We first analyze the complexity of task map evaluation and Jacobian-vector-product subroutine (Algorithm 1) based on reverse accumulation in preparation for the complexity analysis for RMP^2.

Task map evaluation: For each node in the graph, the input and output dimensions are bounded by $O(bd)$ and $O(d)$, respectively. Hence, evaluating each node has a time complexity in $O(bd^2)$. Because each node is evaluated exactly once in in computing the full task map, the total time complexity of task map evaluation is $O(Nbd^2)$. If the gradient oracle `gradient` will be called (as in RMP^2), the value of each node needs to be stored in preparation for the gradient computation. Overall this would require a space complexity in $O(Nd)$ to store the values in the entire graph.

Jacobian-vector product with L output nodes: Suppose that the output of the graph in Algorithm 1 is a collection of L nodes in the graph. During reverse accumulation, the task map is first computed, which, based on the previous analysis, has time and space complexity of $O(Nbd^2)$ and $O(Nd)$, respectively. The dummy variable $\mathbf{w}$ is of size $O(Ld)$ and computing the inner product $\mathbf{w}^\top \mathbf{y}$ requires $O(Ld)$ computation (i.e. it creates a new node

[11] Popular deep learning libraries, e.g., TensorFlow [Abadi et al., 2015] and PyTorch [Paszke et al., 2017], are usually based on reverse-mode automatic differentiation.

of dimension 1 with $2L$ parent nodes of dimension in $O(d)$). By the reverse-mode automatic differentiation assumption, the first backward pass on the graph (line 5) has time complexity of $O(Nbd^2 + Ld) = O(Nbd^2)$ and space complexity of $O(Nd + Ld) = O(Nd)$ [Griewank and Walther, 2008]. The final backward pass (line 6) is on a graph of size $O(N)$, as the first backward pass creates additional $O(N)$ nodes. With a similar analysis, the second backward pass have time complexity of $O(Nbd^2 + Ld) = O(Nbd^2)$ and space complexity of $O(Nd+Ld) = O(Nd)$. Therefore, the time and space complexity of Algorithm 1 is $O(Nbd^2)$ and $O(Nd)$.

Then, we can analyze the time and space complexity of RMP^2 shown in Algorithm 2.

Forward pass: The complexity of line 3–5 in Algorithm 2 follows the analyses above. Here the computation graph is always of size $O(N)$ (the original task map is in $O(N)$ and each call of gradient oracle `gradient` in the Jacobian-vector-product subroutine `jvp` creates additional $O(N)$ nodes in the computation graph). By previous analysis, the time and space complexity of the forward pass are $O(Nbd^2)$ and $O(Nd)$.

RMP evaluation: Assume, for each leaf node, $O(d^3)$ computation is needed for evaluating the importance weight matrix and $O(d^2)$ for acceleration in an RMP (i.e., $O(d)$ for filling in each component of the importance weight and acceleration). The RMP evaluation step then has time complexity of $O(Ld^3)$ and space complexity of $O(Ld^2 + Ld) = O(Ld^2)$.

Backward pass: By previous analyses, task map evaluation requires $O(Nbd^2)$ computation and $O(Nd)$ space. The vector-matrix-vector product for computing auxiliary variables r and s has time complexity of $O(Ld^2)$. To compute the importance weight matrix at the root, $\mathbf{M}_\mathtt{r}$, the first backward pass, $\mathtt{gradient(}\sum_{k=1}^{K}(\mathbf{x}'_k)^\top \mathbf{M}_k \mathbf{x}''_k\mathtt{,q')}$, needs $O(Nbd^2)$ time and $O(Nd)$ space as it operates on a graph of size $O(N)$ where the number of parents of each node is in $O(b)$[12]. The `jacobian` operator in line 9 is done by $O(d)$ sequential calls of the gradient oracle `gradient`. Hence, it has a time complexity of $O(Nbd^3)$ and a space complexity of $O(Nd)$. (If we are not taking further derivatives, the values of the new graphs

[12]Except the final node aggregating L outputs. However, it does not change the complexity as it adds a complexity in $O(Ld^2) < O(Nbd^2)$.

created in calling the `jacobian` operator do not need to be stored.) With a similar analysis, computing $\mathbf{f}$ requires $O(Nbd^3)$ computation and $O(Nd)$ space.

Matrix Inversion: Matrix inversion has time complexity of $O(d^3)$ and space complexity of $O(d^2)$.

In summary, the time complexity of RMP^2 is $O(Nbd^2 + Ld^3 + Nbd^3 + d^3) = O(Nbd^3)$ and the space complexity is $O(Nd + Ld^2)$.

3.2.3 Stability Properties of RMP^2

As we discussed in Section 2.2, stability is an important property for a dynamical system. It can be used to establish desirable long-term properties of the robotics system including safety guarantees, e.g., the robot will never collide with obstacles or violating joint limits, and performance guarantees, e.g., the robot can always reach the goal. In this section, we will establish the stability properties of RMP^2. This section is adapted from [Li et al., 2019a]. It should be noted that Li et al. [2019a] analyze the message passing algorithm RMPflow [Cheng et al., 2018, 2020], which makes the stability analysis unnecessarily convoluted. Instead, we directly analyze the stability of the closed-form solution (3.10). Our stability results are strong than that of Cheng et al. [2018, 2020], which only provide guarantees for a special type of systems called *geometric dynamical systems* (GDSs).

Although it is tempting to think that if each subtask controller (policy) is stable, then the system under configuration space controller (policy) is stable, this is in general not true. This is because the importance matrices $\{\mathbf{M}_k\}_{k=1}^K$ are state-dependent. In some cases, the stability of the overall system has been shown through a common Lyapunov function among all subtasks [Vu and Liberzon, 2005, Narendra and Balakrishnan, 1994], but the existence of a common Lyapunov function is not guaranteed. Finding a common Lyapunov function can be particularly challenging for robotics applications because the tasks can potentially conflict, e.g. the robot need to move through a cluttered environment to go to the goal.

In this section, we provide a necessary condition for the system to be stable under the configuration space controller (3.10). The necessary condition has two parts. First, each of the subtask importance weight matrix $\mathbf{M}_k$ should be induced by a Riemannian metric

$\mathbf{G}_k : (\mathbf{x}_k, \dot{\mathbf{x}}_k) \mapsto \mathbf{G}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k) \in \mathbb{R}_+^{m \times m}$ defined on the tangent bundle[13] of $\mathcal{T}_k$. Second, each subtask RMP $(\mathbf{M}_k, \mathbf{a}_k)$ should satisfy a control Lyapunov function (CLF) condition.

Condition on Importance Weights

We now formally define the first condition, i.e., each importance weight should be induced by a Riemannian metric. Let $\mathbf{G} : (\mathbf{x}, \dot{\mathbf{x}}) \mapsto \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) \in \mathbb{R}_+^{m \times m}$ be a Riemannian metric defined on the space of $(\mathbf{x}, \dot{\mathbf{x}})$. We define *curvature terms*

$$\begin{aligned} \boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) &\coloneqq \frac{1}{2} \sum_{i=1}^{m} \dot{x}_i \, \partial_{\dot{\mathbf{x}}} \, \mathbf{g}_i(\mathbf{x}, \dot{\mathbf{x}}), \\ \boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) &\coloneqq \overset{\mathbf{x}}{\dot{\mathbf{G}}}(\mathbf{x}, \dot{\mathbf{x}}) \, \dot{\mathbf{x}} - \frac{1}{2} \nabla_{\mathbf{x}} \left(\dot{\mathbf{x}}^\top \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) \, \dot{\mathbf{x}} \right), \end{aligned} \tag{3.13}$$

where $\overset{\mathbf{x}}{\dot{\mathbf{G}}}(\mathbf{x}, \dot{\mathbf{x}}) \coloneqq [\partial_{\mathbf{x}} \, \mathbf{g}_i(\mathbf{x}, \dot{\mathbf{x}}) \, \dot{\mathbf{x}}]_{i=1}^{m}$, $\mathbf{g}_i(\mathbf{x}, \dot{\mathbf{x}})$ is the ith column of $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$, and $\dot{x}_i$ is the ith component of $\dot{\mathbf{x}}$. The importance weight matrix (also known as the *inertial*) $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})$ induced by $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ is defined as,

$$\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}}) \coloneqq \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) + \boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}). \tag{3.14}$$

Note that $\boldsymbol{\Xi}_{\mathbf{G}} \equiv \mathbf{0}_{m \times m}$ if $\mathbf{G}$ is not dependent on velocity, i.e., $\mathbf{G} : \mathbf{x} \mapsto \mathbf{G}(\mathbf{x})$. In this case, we have $\mathbf{M} \equiv \mathbf{G}$, i.e., the importance weight is exactly the same as the Riemannian metric. The term "Riemanniaon" in Riemannian motion policy (RMP) is given by this connection between the importance weight and Riemannian metrics.

We now summarize the first part of the necessary condition as below:

Condition 3.2.1. For each subtask k, there exists a Riemannian metric $\mathbf{G}_k$ on the tangent bundle of $\mathcal{T}_k$ such that $\mathbf{M}_k$ satisfies (3.14) for all $(\mathbf{x}_k, \dot{\mathbf{x}}_k)$.

3.2.4 Control Lyapunov Function Condition on RMPs

Previously, we define a Riemannian metric $\mathbf{G}_k$ for each subtask k. Now, we additionally consider a radially unbounded[14], continuously differentiable and lower-bounded potential

[13] The space of all possible $(\mathbf{x}_k, \dot{\mathbf{x}}_k)$.

[14] See Section 2.2 for definition.

function $\Phi_k : \mathbb{R}^m \to \mathbb{R}$. Then, the *energy* for the k-th subtask is given by the sum of *kinetic* energy and *potential* energy, i.e.,

$$V_k(\mathbf{x}_k, \dot{\mathbf{x}}_k) = \frac{1}{2}\, \dot{\mathbf{x}}_k^\top\, \mathbf{G}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k)\, \dot{\mathbf{x}}_k + \Phi_k(\mathbf{x}_k), \tag{3.15}$$

which is the Lyapunov function candidate that we will be working with. The control Lyapunov function condition is given by:

Condition 3.2.2. Under the acceleration policy $\mathbf{a}_k$, the acceleration-based system on the subtask space $\mathcal{T}_k$ should satisfy,

$$\dot{V}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k) \leq -\alpha_k\left(\|\dot{\mathbf{x}}_k\|\right), \tag{3.16}$$

for some Lyapunov function candidate V_k defined in (3.15), where α_k is a locally Lipschitz continuous class $\mathcal{K}$ functions [Khalil and Grizzle, 2002], i.e., $\alpha_k : [0, \infty) \to [0, \infty)$ is strictly increasing and $\alpha_k(0) = 0$.

Remark Given Condition 3.2.2, we can show, by LaSalle's invariance principle (Theorem 2.2.3) that the system under acceleration policy $\mathbf{a}_k$ converges to the largest invariant set in $\{(\mathbf{x}_k, \dot{\mathbf{x}}_k) : \mathbf{a}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k) = 0\}$.

Example 3.2.1 (Geometric Dynamical Systems)**.** Cheng et al. [2018, 2020] define a type of dynamical systems called *Geometric dynamical systems* (GDSs). Given a Riemannian metric $\mathbf{G}$ and potential function Φ, a GDS is in the form of,

$$\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})\, \ddot{\mathbf{x}} + \boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) = -\nabla_{\mathbf{x}} \Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}}, \tag{3.17}$$

where $\boldsymbol{\xi}$ and $\mathbf{M}$ are defined in (3.13) and (3.14), respectively, $\mathbf{B} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+^{m \times m}$ is the *damping matrix*. When $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{G}(\mathbf{x})$, a GDS reduces to the widely studied equation of motion [Bullo and Lewis, 2004],

$$\mathbf{G}(\mathbf{x})\, \ddot{\mathbf{x}} + \mathbf{C}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} = -\nabla_{\mathbf{x}} \Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}}, \tag{3.18}$$

where the term $\mathbf{C}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}}$ corresponds to the centrifugal and Coriolis force. We will now show that the RMP $(\mathbf{M}, \mathbf{a})$, where $\mathbf{a} \coloneqq -\mathbf{M}^{-1}(-\nabla_{\mathbf{x}} \Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} - \boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})))$ satisfies Condition 3.2.2.

Lemma 3.2.1. *Given Lyapunov function candidate*

$$V(\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2}\, \dot{\mathbf{x}}^\top\, \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} + \Phi(\mathbf{x}),$$

a geometric dynamical system in the form of (3.17) *satisfies*

$$\dot{V}(\mathbf{x}, \dot{\mathbf{x}}) = -\dot{\mathbf{x}}^\top\, \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} \leq -\lambda_{\min} \|\dot{\mathbf{x}}\|^2,$$

where $\lambda_{\min}$ *is the smallest eigenvalue of* $\mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})$*. Moreover, by LaSalle's invariance principle (Theorem 2.2.3), the system converges to the invariant set* $\{(\mathbf{x}, \dot{\mathbf{x}}) : \nabla_{\mathbf{x}} \Phi(\mathbf{x}) = \mathbf{0}, \dot{\mathbf{x}} = \mathbf{0}\}$.

Proof. By taking the time derivative of the Lyapunov function candidate, following similar derivation from [Li et al., 2019a], we have,

$$\begin{aligned} \dot{V}(\mathbf{x}, \dot{\mathbf{x}}) &= \frac{1}{2}\, \dot{\mathbf{x}}^\top \left(\frac{d}{dt} \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) \right) \dot{\mathbf{x}} + \dot{\mathbf{x}}^\top\, \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})\, \ddot{\mathbf{x}} + \dot{\mathbf{x}}^\top \nabla_{\mathbf{x}} \Phi(\mathbf{x}) \\ &= \frac{1}{2}\, \dot{\mathbf{x}}^\top\, \overset{\mathbf{x}}{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} + \dot{\mathbf{x}}^\top\, \mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})\, \ddot{\mathbf{x}} + \dot{\mathbf{x}}^\top \nabla_{\mathbf{x}} \Phi(\mathbf{x}), \end{aligned} \tag{3.19}$$

where $\overset{\mathbf{x}}{\mathbf{G}}$ is defined below (3.13). Since the controller is given by GDS (3.17), we have,

$$\dot{\mathbf{x}}^\top\, \mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})\, \ddot{\mathbf{x}} = \dot{\mathbf{x}}^\top \big(-\nabla_{\mathbf{x}} \Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} - \boldsymbol{\xi}(\mathbf{x}, \dot{\mathbf{x}}) \big). \tag{3.20}$$

Plugging this back into (3.19),

$$\begin{aligned} \dot{V}(\mathbf{x}, \dot{\mathbf{x}}) &= \frac{1}{2}\, \dot{\mathbf{x}}^\top\, \overset{\mathbf{x}}{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} - \dot{\mathbf{x}}^\top \boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) - \dot{\mathbf{x}}^\top\, \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} \\ &= -\frac{1}{2}\, \dot{\mathbf{x}}^\top\, \overset{\mathbf{x}}{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} + \frac{1}{2}\, \dot{\mathbf{x}}^\top \nabla_{\mathbf{x}} \left(\dot{\mathbf{x}}^\top \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} \right) - \dot{\mathbf{x}}^\top\, \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}}, \end{aligned}$$

where the last equation is by definition of $\boldsymbol{\xi}_{\mathbf{G}}$ from (3.13). Note that

$$\begin{aligned} \dot{\mathbf{x}}^\top\, \overset{\mathbf{x}}{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} &= \sum_{i=1}^{m} \sum_{j=1}^{m} \dot{x}_i \dot{x}_j \sum_{k=1}^{m} \dot{x}_k \frac{\partial g_{ij}(\mathbf{x}, \dot{\mathbf{x}})}{\partial x_k} \\ &= \sum_{i=1}^{m} \dot{x}_i \sum_{j=1}^{m} \sum_{k=1}^{m} \dot{x}_j \dot{x}_k \frac{\partial g_{jk}(\mathbf{x}, \dot{\mathbf{x}})}{\partial x_i} = \dot{\mathbf{x}}^\top \nabla_{\mathbf{x}} \left(\dot{\mathbf{x}}^\top \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} \right). \end{aligned}$$

Hence, we have,

$$\dot{V}(\mathbf{x}, \dot{\mathbf{x}}) = 0 - \dot{\mathbf{x}}^\top\, \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}} = -\dot{\mathbf{x}}^\top\, \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\, \dot{\mathbf{x}}.$$

The rest of the proof follows directly from LaSalle's invariance principle[15] (Theorem 2.2.3).

□

[15]Since $\mathbf{G}$ is positive definite and then Φ is radially unbounded with respect to $\mathbf{x}$, V is radially unbounded with respect to $(\mathbf{x}, \dot{\mathbf{x}})$. Given any $(\mathbf{x}_0, \dot{\mathbf{x}}_0)$, we can choose Ω to be the closure of $\{(\mathbf{x}, \dot{\mathbf{x}}) : V(\mathbf{x}, \dot{\mathbf{x}}) \leq V(\mathbf{x}_0, \dot{\mathbf{x}}_0)\}$, which a compact set.

3.2.5 Stability Properties of RMP^2

Now, under Condition 3.2.1 and Condition 3.2.2, we discuss the stability properties of RMP^2, which is given by the following proposition. This proposation shows how control Lyapunov function (CLF) constraints are propagated from the subtask spaces to the configuration space through RMP^2.

Proposition 3.2.1. *Assume that Condition 3.2.1 and Condition 3.2.2 hold. Assume that all subtask maps, Jacobians and their derivatives are locally Lipschitz continuous. Consider the Lyapunov function candidate*

$$V(\mathbf{q}, \dot{\mathbf{q}}) \coloneqq \frac{1}{2} \dot{\mathbf{q}}^\top \mathbf{G}(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} + \Phi(\mathbf{q}), \tag{3.21}$$

with the configuration space (pseudo[16]) Riemannian metric $\mathbf{G}$ *and potential function* Φ

$$\begin{aligned} \mathbf{G}(\mathbf{q}, \dot{\mathbf{q}}) &\coloneqq \sum_{k=1}^{K} \mathbf{J}_k(\mathbf{q})^\top \mathbf{G}_k(\psi_k(\mathbf{q}), \mathbf{J}_k(\mathbf{q}) \dot{\mathbf{q}}) \mathbf{J}_k(\mathbf{q}) = \sum_{k=1}^{K} \mathbf{J}_k(\mathbf{q})^\top \mathbf{G}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k) \mathbf{J}_k(\mathbf{q}), \\ \Phi(\mathbf{q}) &\coloneqq \sum_{k=1}^{K} \Phi_k(\psi_k(\mathbf{q})) = \sum_{k=1}^{K} \Phi_k(\mathbf{x}_k). \end{aligned} \tag{3.22}$$

Then, under the configuration space control policy produced by RMP^2 defined in (3.10)*,*

$$\dot{V}(\mathbf{q}, \dot{\mathbf{q}}) \leq -\sum_{k=1}^{K} \alpha_k \left(\| \mathbf{J}_k(\mathbf{q}) \dot{\mathbf{q}} \| \right). \tag{3.23}$$

Proof. For notational convenience, we suppress the arguments of functions. First, note that,

$$V = \frac{1}{2} \sum_{k=1}^{K} \dot{\mathbf{q}}^\top \mathbf{J}_k^\top \mathbf{G}_k \mathbf{J}_k \dot{\mathbf{q}} + \sum_{k=1}^{K} \Phi_k = \frac{1}{2} \sum_{k=1}^{K} \dot{\mathbf{x}}^\top \mathbf{G}_k \dot{\mathbf{x}} + \sum_{k=1}^{K} \Phi_k = \sum_{k=1}^{K} V_k. \tag{3.24}$$

Hence, we have $\dot{V} = \sum_{k=1}^{K} \dot{V}_k$. Following the same step as (3.19),

$$\dot{V} = \sum_{k=1}^{K} \dot{\mathbf{x}}_k^\top \mathbf{M}_k \ddot{\mathbf{x}}_k + \frac{1}{2} \dot{\mathbf{x}}_k^\top \overset{\mathbf{x}_k}{\mathbf{G}}_k \dot{\mathbf{x}}_k + \dot{\mathbf{x}}_k^\top \nabla_{\mathbf{x}_k} \Phi_k. \tag{3.25}$$

[16] Technically, it is possible that $\mathbf{G}(\mathbf{q}, \dot{\mathbf{q}})$ is positive semi-definite for some $(\mathbf{q}, \dot{\mathbf{q}})$.

Note that $\dot{\mathbf{x}}_k = \mathbf{J}_k \dot{\mathbf{q}}$ and $\ddot{\mathbf{x}}_k = \mathbf{J}_k \ddot{\mathbf{q}} + \mathbf{c}_k$. Hence, the first term can be rewritten as

$$\begin{aligned}\sum_{k=1}^{K} \mathbf{x}_k^\top \mathbf{M}_k \ddot{\mathbf{x}}_k &= \dot{\mathbf{q}}^\top \left(\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \left(\mathbf{J}_k \ddot{\mathbf{q}} + \mathbf{c}_k \right) \right) \\ &= \dot{\mathbf{q}}^\top \left(\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \mathbf{J}_k \right) \ddot{\mathbf{q}} + \dot{\mathbf{q}}^\top \left(\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \mathbf{c}_k \right) \\ &= \dot{\mathbf{q}}^\top \mathbf{M} \ddot{\mathbf{q}} + \dot{\mathbf{q}}^\top \left(\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \mathbf{c}_k \right).\end{aligned}$$

Under configuration space policy given by RMP^2, i.e., (3.10),

$$\mathbf{M} \ddot{\mathbf{q}} = \mathbf{f} = \sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k (\mathbf{a}_k - \mathbf{c}_k).$$

Therefore,

$$\begin{aligned}\sum_{k=1}^{K} \mathbf{x}_k^\top \mathbf{M}_k \ddot{\mathbf{x}}_k &= \dot{\mathbf{q}}^\top \mathbf{f} + \dot{\mathbf{q}}^\top \left(\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \mathbf{c}_k \right) \\ &= \dot{\mathbf{q}}^\top \left(\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k (\mathbf{a}_k - \mathbf{c}_k) \right) + \dot{\mathbf{q}}^\top \left(\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \mathbf{c}_k \right) \\ &= \dot{\mathbf{q}}^\top \left(\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \mathbf{a}_k \right) \\ &= \sum_{k=1}^{K} \dot{\mathbf{x}}_k^\top \mathbf{M}_k \mathbf{a}_k.\end{aligned}$$

The time-derivative of V can then be simplified as

$$\dot{V} = \sum_{k=1}^{K} \left(\dot{\mathbf{x}}_k^\top \mathbf{M}_k \mathbf{a}_k + \frac{1}{2} \dot{\mathbf{x}}_k^\top \overset{\mathbf{x}_k}{\mathbf{G}}_k \dot{\mathbf{x}}_k + \dot{\mathbf{x}}_k^\top \nabla_{\mathbf{x}_k} \Phi_k \right) \leq - \sum_{k=1}^{K} \alpha_k(\|\dot{\mathbf{x}}_k\|), \tag{3.26}$$

where the last inequality follows from Condition 3.2.2. □

With this insight, we state the stability theorem of RMP^2 by applying LaSalle's invariance principle. We assume that the matrix $\mathbf{M}$ is non-singular for simplicity; a sufficient condition for $\mathbf{M}$ being non-singular is provided in Cheng et al. [2018].

Theorem 3.2.1. *Assume that Condition 3.2.1 and Condition 3.2.2 hold. Assume that all subtask maps, Jacobians and their derivatives are locally Lipschitz continuous. Suppose that*

$\mathbf{M}$ *is non-singular, and the map* $\mathbf{q} \mapsto (\mathbf{x}_k)_{k=1}^K$ *is injective. Then the control policy generated by RMP*^{2} (3.10) *renders the system converging to the invariant set*

$$\mathcal{C}_\infty \coloneqq \left\{ (\mathbf{q}, \dot{\mathbf{q}}) : \sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \, \mathbf{a}_k = \mathbf{0}, \ \dot{\mathbf{q}} = \mathbf{0} \right\}. \tag{3.27}$$

Further, if for all subtask RMPs $\{(\mathbf{M}_k, \mathbf{a}_k)\}_{k=1}^K$, $\mathbf{M}_k \, \mathbf{a}_k = -\nabla_{\mathbf{x}_k} \Phi_k(\mathbf{x}_k)$ *when* $\dot{\mathbf{x}}_k = 0$, *the system converges to the invariant set*

$$\mathcal{C}_\infty^\Phi \coloneqq \{(\mathbf{q}, \dot{\mathbf{q}}) : \nabla_{\mathbf{q}} \Phi(\mathbf{q}) = 0, \ \dot{\mathbf{q}} = 0\}. \tag{3.28}$$

Proof. By assumption, V is radially unbounded, continuously differentiable and lower bounded. Hence, the system converges to the largest invariant set in $\{(\mathbf{q}, \dot{\mathbf{q}}) : \dot{V}(\mathbf{q}, \dot{\mathbf{q}}) = 0\}$ by LaSalle's invariance principle (Theorem 2.2.3). By (3.23) in Proposition 3.2.1, $\dot{V} = 0$ if and only if $\mathbf{J}_k \, \dot{\mathbf{q}} = \mathbf{0}$ for all $k = 1, \ldots, K$. Since the map $\mathbf{q} \mapsto (\mathbf{x}_k)_{k=1}^K$ is injective, we have $\dot{\mathbf{q}} = \mathbf{0}$. Hence, the system converges to a invariant set in $\{(\mathbf{q}, \dot{\mathbf{q}}) : \dot{\mathbf{q}} = \mathbf{0}\}$. Any invariant set must have $\ddot{\mathbf{q}} = \mathbf{0}$, which implies that $\mathbf{f} = \mathbf{0}$, hence,

$$\mathbf{0} = \mathbf{f} = \sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \, (\mathbf{a}_k - \mathbf{c}_k) = \sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \, \mathbf{a}_k$$

where the last equality follows from $\dot{\mathbf{q}} = 0$. Thus, the system converges to the forward invariant set $\mathcal{C}_\infty$ defined in (3.27).

Now, assume that $\mathbf{M}_k \mathbf{a}_k = -\nabla_{\mathbf{x}_k} \Phi_k(\mathbf{x}_k)$ when $\dot{\mathbf{x}}_k = 0$ (which is implied by $\dot{\mathbf{q}} = 0$). Notice that by the definition of Φ in (3.22), $\Phi(\mathbf{q}) = \sum_{k=1}^K \Phi_k(\mathbf{x}_k)$. By the chain rule,

$$\nabla_{\mathbf{q}} \Phi(\mathbf{q}) = \sum_{k=1}^{K} \mathbf{J}_k^\top \nabla_{\mathbf{x}_k} \Phi_k(\mathbf{x}_k) = -\sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \, \mathbf{a}_k.$$

Hence $\sum_{k=1}^K \mathbf{J}_k^\top \mathbf{M}_k \, \mathbf{a}_k = 0$ is equivalent to $\nabla_{\mathbf{q}} \Phi(\mathbf{q}) = 0$. This system converges to $\mathcal{C}_\infty^\Phi$ defined in (3.28). □

Theorem 3.2.1 implies that subtask RMPs satisfying CLF constraints (3.16) can be stably combined by RMP^2. Based on this result, we propose a computational framework in the following section to design subtask RMPs with stability guarantees under RMP^2.

Remark Theorem 3.2.1 is more general than what is established by Cheng et al. [2018, 2020], i.e., RMP^2 can stably combine GDSs, as Theorem 3.2.1 implies the latter. Assume that all subtask RMPs are given by GDSs in the form of (3.17). By definition of a GDS, Condition 3.2.1 holds. Moreover, by Lemma 3.2.1, Condition 3.2.2 also holds. Therefore, by Theorem 3.2.1, the system converges to the invariant set $\mathcal{C}_\infty^\Phi$, as, by definition (3.13), $\boldsymbol{\xi}_{\mathbf{G}_k} = \mathbf{0}$ if $\dot{\mathbf{x}}_k = \mathbf{0}$.

3.2.6 A Computational Framework for RMP^2 with Stability Guarantees

Based on the stability results (Theorem 3.2.1), we introduce a computational framework for generating RMPs (not necessarily derived from GDSs) which can be stably combined by RMP^2, i.e., RMPs that satisfy both Condition 3.2.1 and Condition 3.2.2.

Assume that a nominal acceleration policy (which is not derived by a GDS) $\pi_k(\mathbf{x}_k, \dot{\mathbf{x}}_k)$ is provided for each subtask k. These policies can be given by, e.g., heuristic, motion planner, or demonstrations (as we will see in Chapter 5). It should be noted that $\pi_k(\mathbf{x}_k, \dot{\mathbf{x}}_k)$ does not necessarily satisfy the CLF constraint (3.29), in other words, Condition 3.2.2 does not hold. This means that directly combining subtask RMPs $\{(\mathbf{M}_k, \pi_k)\}_{k=1}^K$ using RMP^2 may not yield a stable behavior. Although it is possible to directly design a GDS in the form of (3.17), it is not desirable to do so for two main reasons:

1. It does not make use of the nominal policies, which contain useful information for solving the task.

2. For certain behaviors, designing GDSs could be difficult or impossible[17], as we will see in the example shown in Fig. 3.2.

Our main idea is to leverage Proposition 3.2.1, which says that RMP^2 is capable of preserving CLF constraints in certain form. Recall from Condition 3.2.2 that the constraint on the time-derivative of subtask Lyapunov function is $\dot{V}_k \leq -\alpha_k(\|\dot{\mathbf{x}}_k\|)$. Combined with

[17]Cheng et al. [2018, 2020] discuss the relationship between the trajectories of a GDS and the geodesics on the corresponding Riemannian manifold.

the choice of subtask Lyapunov function candidate in (3.15), this yields a CLF constraint[18]

$$\dot{\mathbf{x}}_k^\top \mathbf{M}_k \mathbf{a}_k \leq -\dot{\mathbf{x}}_k^\top (\nabla_{\mathbf{x}_k} \Phi_k + \boldsymbol{\xi}_{\mathbf{G}_k}) - \alpha_k(\|\dot{\mathbf{z}}_k\|), \tag{3.29}$$

where $\boldsymbol{\xi}_{\mathbf{G}_k}$ is defined in (3.13). Proposition 3.2.1 shows that, when the subtask RMPs satisfy (3.29), RMP^2 will yield a stable policy in the sense that it renders the system converges to a certain invariant set. This provides a constructive principle to design (and as we shall see later in Chapter 5, parameterize) RMPs.

To make the notation more succinct, we introduce the *natural* form of an RMP. Given an RMP (in its *canonical* form) $(\mathbf{M}, \mathbf{a})$, its *natural* form is defined as $[\mathbf{M}, \mathbf{f}]$, where $\mathbf{f} = \mathbf{M}\,\mathbf{a}$. Note that we use brackets $[\cdot, \cdot]$ for natural-form RMPs and parentheses $(\cdot, \cdot)$ for canonical RMPs. When $\mathbf{M}$ is non-singular, the canonical-form RMP $(\mathbf{M}, \mathbf{a})$ is equivalent to the natural-form RMP $[\mathbf{M}, \mathbf{M}\,\mathbf{a}]$.

Given subtask RMPs (in natural forms) $\{[\mathbf{M}_k, \mathbf{f}_k]\}_{k=1}^K$, we can rewrite the CLF constraints in (3.29) as,

$$\dot{\mathbf{x}}_k^\top \mathbf{f}_k \leq -\dot{\mathbf{x}}_k^\top (\nabla_{\mathbf{x}_k} \Phi_k + \boldsymbol{\xi}_{\mathbf{G}_k}) - \alpha_k(\|\dot{\mathbf{z}}_k\|), \tag{3.30}$$

where is a linear constraint on $\mathbf{f}_k$. It should be noted that the close-form solution (3.10) can also be given by natural-form subtask RMPs,

$$\pi(\mathbf{q}, \dot{\mathbf{q}}) = \underbrace{\left(\sum_{k=1}^K \mathbf{J}_k^\top \mathbf{M}_k \mathbf{J}_k\right)^\dagger}_{\mathbf{M}^\dagger} \underbrace{\left(\sum_{k=1}^K \mathbf{J}_k^\top \left(\mathbf{f}_k - \mathbf{M}_k \mathbf{c}_k\right)\right)}_{\mathbf{f}}. \tag{3.31}$$

Given subtask nominal acceleration policies $\{\pi_k\}_{k=1}^K$, we design subtask RMP as a *minimally invasive* controller that modifies the nominal controller as little as possible while satisfying the CLF constraint (3.30). This yields a quadratic objective with linear constraints on $\mathbf{f}_k$, which is can be solved via quadratic programming (QP).

$$\begin{aligned} \mathbf{f}_k^\star = \arg\min_{\mathbf{f}_k} \quad & \|\mathbf{f}_k - \mathbf{M}_k\,\pi_k\|_{\mathbf{P}_k}^2 \\ \text{s.t.} \quad & \dot{\mathbf{x}}_k^\top \mathbf{f}_k \leq -\dot{\mathbf{x}}_k^\top (\nabla_{\mathbf{x}_k} \Phi_k + \boldsymbol{\xi}_{\mathbf{G}_k}) - \alpha_k(\|\dot{\mathbf{x}}_k\|), \end{aligned} \tag{3.32}$$

[18]This a linear constraint with respect to $\mathbf{f}_k \coloneqq \mathbf{M}_k \mathbf{a}_k$. When $\dot{\mathbf{x}}_k = 0$, the constraint (3.29) holds trivially because both sides equal 0.

where $\mathbf{P}_k \succ 0$ and $\mathbf{M}_k$ is given by $\mathbf{G}_k$ through (3.14). Possible choices of $\mathbf{P}_k$ include the identity matrix $\mathbb{I}_{m\times m}$ and the inverse of the importance weight matrix $\mathbf{M}_k^{-1}$. In particular, $\mathbf{P}_k = \mathbf{M}_k^{-1}$ yields an objective function equivalent to $\|\mathbf{a}_k - \pi_k\|^2_{\mathbf{M}_k}$, where $\mathbf{a}_k$ is the acceleration policy in the canonical form of the subtask RMP.

We combine this minimally invasive controller design with RMP^2 as a new computational framework for policy generation, called *RMP^2–CLF*. RMP^2–CLF follows the same procedure as the original RMP^2 [Li et al., 2021] as is discussed in Section 3.2. The difference is that the evaluation step solves the optimization problem (3.32). Note that (3.32) is a QP problem with a single linear constraint, so it can be solved analytically by projecting $\mathbf{M}_k\pi_k$ onto the half-plane given by the constraint.

The form of (3.32) together with Theorem 3.2.1 and the results of Morris et al. [2013] yields the following theorem:

Theorem 3.2.2. *Under the assumptions in Theorem 3.2.1, if $\{\pi_k\}_{k=1}^K$, $\{\mathbf{M}_k\}_{k=1}^K$ are locally Lipschitz continuous, then the policy generated by RMP^2–CLF is locally Lipschitz continuous and renders the system converging forwardly to* (3.27).

Proof. By Theorem 3.2.1, the system converges to (3.27). By Morris et al. [2013], for all $k \in \{1, \dots, K\}$, $\mathbf{f}_k$ is locally Lipschitz continuous. Since under the Lipschitz continuity assumption in Theorem 3.2.1, the closed-form solution (3.31) is Lipschitz continuous with respect to $(\mathbf{q}, \dot{\mathbf{q}})$; the statement follows. □

Note that RMP^2 can be interpreted as a soft version of the QP–CLF formulation [Morris et al., 2013] that enforces the decay-rates of *all* Lyapunov function candidates. Meanwhile, compared with the QP–CLF framework with slack variables Ames et al. [2014] that requires the users to design the objective function trade off between control specifications, RMP^2 provides a structured way to implicitly generate such an objective function so that the system is stable.

It should be noted that the system can also be stabilized by directly enforcing a single constraint on the time derivative of the combined Lyapunov function candidate (3.23), rather than enforcing the CLF constraint for every subtask (3.16). However, this can be less desirable: although the stability can be guaranteed for the resulting controller, the behavior

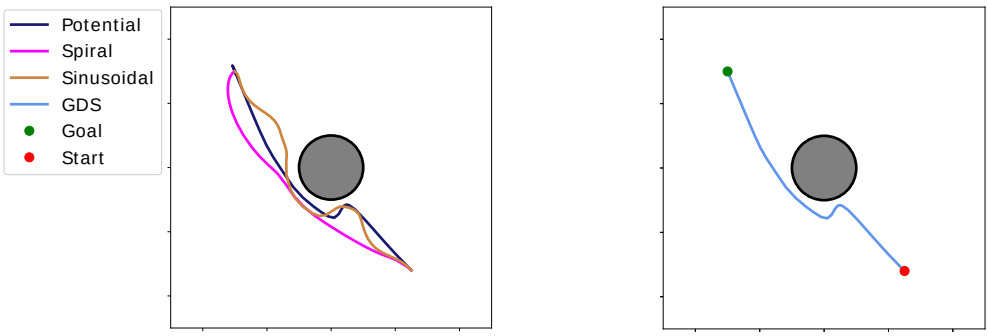
of each individual subtask is no longer explicitly regulated. By contrast, the approach with subtask CLF constraints allows the users to design and test the subtask policies from (3.32) independently. This allows for designing policies that can be applied to robots with different kinematic structures.

3.2.7 Experimental Results

We compare the proposed RMP^2–CLF framework with the original RMPflow framework by Cheng et al. [2018]. A video of the experimental results can be found here. The original RMPflow framework by Cheng et al. [2018] is referred to as RMPflow–GDS.

Simulated Single Robot Goal Reaching

We first present a simple simulated example, showing that, with a fixed choice a metric and potential function, RMP^2–CLF can generate richer behaviors (than the associated GDSs) under different choices of nominal controllers. We consider the 2D goal reaching task presented by Cheng et al. [2018]. A planar robot with double-integrator dynamics (2.2) is expected to move to a goal position without colliding into a circular obstacle. As is done by Cheng et al. [2018], there is a collision avoidance subtask and a goal attractor subtask. For the RMP^2–CLF framework, we use the collision avoidance RMP in [Cheng et al., 2018] and keep the choice of metrics and potential functions for the goal attractor RMP consistent with Cheng et al. [2018]. For the goal attractor RMP, we present several nominal controllers: (i) a pure potential-based nominal controller $\mathbf{M}\,\pi_{\text{pt}} = -\nabla\Phi$; (ii) a spiral nominal controller $\mathbf{M}\,\pi_{\text{sp}} = -\nabla\Phi + \|\dot{\mathbf{x}}\|\,\mathbf{v}$, where $\mathbf{v}$ is the potential-based controller rotated by $\pi/2$, i.e. $\mathbf{v} = -R(\pi/2)\,\nabla\Phi$ with $R(\cdot)$ being the rotation matrix; and (iii) a sinusoidal controller $\mathbf{M}\,\pi_{\text{sn}} = -\nabla\Phi + \sin(t/4)\,\|\dot{\mathbf{x}}\|\,\mathbf{v}$. For the minimally invasive controller, we use $\mathbf{P} = \mathbb{I}_{2\times 2}$ to minimize the Euclidean distance between the nominal controller and the solution to the optimization problem (3.32). We implement the RMPflow–GDS framework with the same choice of parameters as Cheng et al. [2018]. The trajectories under different nominal controllers are shown in Fig. 3.2. Although it may be possible that similar behaviors can be realized by a careful redesign of the metric and potential function for a GDS, the



(a) RMP^2–CLF (b) RMPflow–GDS [Cheng et al., 2018]

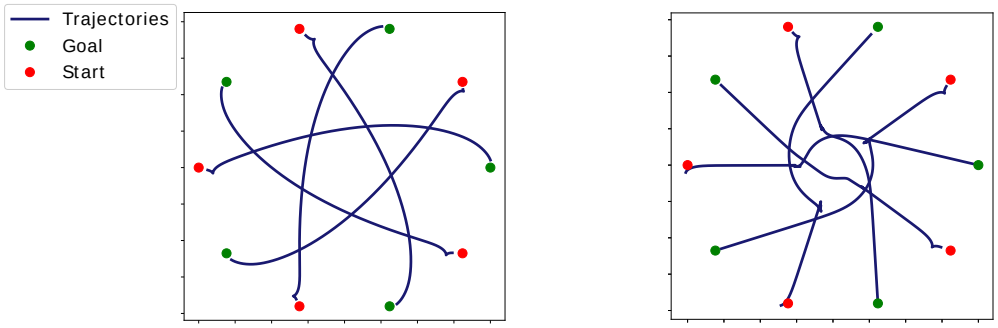
Figure 3.2: 2D goal reaching task with a circular obstacle (grey). (a) RMP^2–CLF with three choices of nominal controllers, resulting in different goal reaching behaviors. (b) RMPflow–GDS with the goal attractor given by a GDS. The behavior is limited by the choice of the metric and the potential function.

RMP^2–CLF framework can produce a rich class of behaviors without being concerned with the geometric properties of the subtask manifold.

Simulated Multi-robot Goal Reaching

RMP^2–CLF guarantees system stability even when the nominal controllers are not inherently stable or asymptotically stable. Therefore, the user can incorporate design knowledge given by, e.g. motion planners, human demonstrations or even heuristics, into the nominal controllers. To illustrate this, we consider a multi-robot goal reaching task, where the robots are tasked with moving to the opposite side of the arena without colliding. If the robots move in straight lines, their trajectories would intersect near the center of the arena. Due to the symmetric configuration, the system can easily deadlock with robots moving very slowly or stopping near the center to avoid collisions. This problem can be fixed if the symmetry is broken. One possible solution is to design nominal controllers for the goal attractors so that the robots move along curves.

We compare the spiral goal attractor RMP with the GDS goal attractor RMP designed



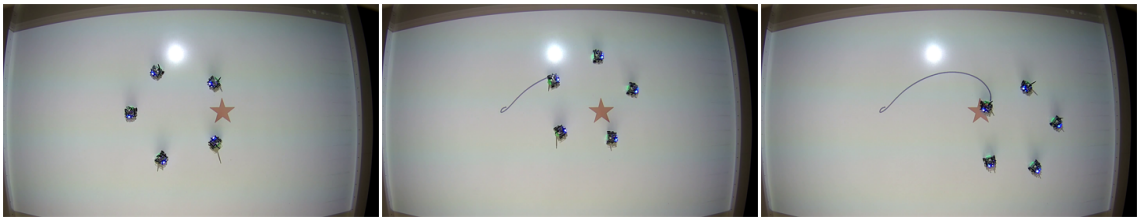
(a) RMP^2–CLF (b) RMPflow–GDS Cheng et al. [2018]

Figure 3.3: Multi-robot goal reaching task. (a) RMP^2–CLF with spiral nominal controllers. The robots move to their goal smoothly. (b) RMPflow–GDS with the goal attractor given by a GDS. Due to the symmetry of the configuration, the system suffers from deadlock when the robots are near the center: the robots oscillate around the deadlock configuration.

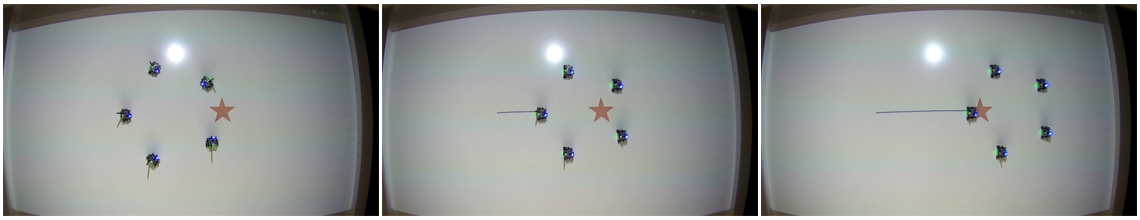
by Li et al. [2019b]. In both cases, we define collision avoidance for pairs of robots with the same choice of parameters. The trajectories of the robots under the spiral nominal controllers are shown in Fig. 3.3a. The spiral controllers produce smooth motion, whereas the GDS goal attractors produce jerky motion when the robots are near the center due to deadlock caused by the symmetric configuration (Fig. 3.3b).

3.2.8 Real Robot Implementation

We present an experiment conducted on the Robotarium Pickem et al. [2017], a remotely accessible swarm robotics platform. In the experiment, five robots are tasked with preserving a regular pentagon formation while the leader has an additional task of reaching a goal. We use the same RMP-tree structure and parameters for most leaf-node RMPs as described in the formation preservation experiment by Li et al. [2019b]. The only difference is that we replace the GDS goal attractor therein with the spiral nominal controller augmented with the CLF condition (3.32). Fig. 3.4 presents the snapshots from the formation preservation experiment. In Fig. 3.4a–Fig. 3.4c, we see that the leader approaches the goal with a



(a) CLF controller: $t = 0s$ (b) CLF controller: $t = 18s$ (c) CLF controller: $t = 38s$



(d) GDS controller: $t = 0s$ (e) GDS controller: $t = 8s$ (f) GDS controller: $t = 19s$

Figure 3.4: Multi-robot formation preservation task. The robots are tasked with preserving a regular pentagon formation while the leader has an additional task of reaching a goal position. (a) RMP^2–CLF with a spiral nominal controller. (b) RMPflow–GDS. The goal (red star) and the trajectories (blue curves) of the leader robot are projected onto the environment through an overhead projector. RMP^2–CLF shapes the goal-reaching behavior through a spiral nominal controller.

spiral trajectory specified by the nominal controller, while other subtask controllers preserve distances and avoid collision. This shows the efficacy of our controller synthesis framework. By contrast, the robot moves in straight lines under the goal attractor given by the GDS (see Fig. 3.4d-Fig. 3.4f). Although it could be possible to redesign the subtask manifold such that there exists a GDS that produces similar behaviors, the RMP^2–CLF framework provides the user additional flexibility to shape the behaviors without worrying about the geometric properties of the subtask manifolds.

3.3 Conclusion

In this chapter, we consider two types of structure in robotics problems *1)* the acceleration-based system dynamics which is governed by the kinematics of the robot; and *2)* the multi-objective nature of robotics problems. We introduce Riemannian motion policy (RMP), a framework that is especially suitable for encoding these types of structure. In the RMP framework, each subtask is associated with an RMP, $(\mathbf{M}_k, \mathbf{a}_k)$, where $\mathbf{M}_k$ is the importance weight matrix policy and $\mathbf{a}_k$ is the acceleration policy.

To combine subtask RMPs together into a configuration space policy $\pi(\mathbf{q}, \dot{\mathbf{q}})$, we propose RMP^2, an efficient algorithm that can be implemented by automatic differentiation libraries. We then show that RMP^2 can combine subtask RMPs in to produce a stable configuration space policy π, provided that subtask RMPs follow control Lyapunov function (CLF) conditions. This implies a computational framework which creates subtask RMPs that satisfy CLF conditions via quadratic programming (QP). In the next three chapters, we will see how RMP can be a valuable framework for encoding known problem structure in robot learning.

3.4 Related Work

Operational space control Operational space control (OSC) [Khatib, 1987] is originally proposed for force control of robot manipulators. It is used to compute the joint torques of a robot manipulator such that the induced task space dynamics is a spring-mass-damper system. It is later extended to velocity and acceleration-based control of manipulators [Nakanishi et al., 2008]. Despite its popularity, it could be challenging to directly apply OSC to multi-objective problems with multiple tasks (and hence task spaces) of interest.

Riemannian motion policies The Riemannian motion policy (RMP) framework is initially proposed Ratliff et al. [2018], and later refined by Cheng et al. [2018, 2020] such that the combined policy has stability guarantees. Compared to prior work, the main contributions of this thesis are: *1)* we propose a new algorithm for combining RMPs, called RMP^2, which is more computationally efficient and easier to implement; and *2)* we establish a

new stability results which apply to a broader class of RMPs, providing more flexibility for designing (and later, parameterizing) RMPs.

In addition to motion generation for robot manipulators, RMP has been successfully applied to other domains, including autonomous vehicle navigation [Meng et al., 2019], humanoid control [Wingo et al., 2020], multi-robot control [Li et al., 2019b], and task planning [Paxton et al., 2019]. Ratliff et al. [2021] and Van Wyk et al. [2022] extend the framework to Finsler geometry, and propose a related motion generation framework called geometric fabrics.

Null-space and hierarchical control The framework of null-space or hierarchical control [Peters et al., 2008, Escande et al., 2014] handles multi-objective problems by assigning priorities to the tasks, and hence to the controllers. The performance of the high-priority tasks can be guaranteed by forcing the lower-priority controllers to act on the null space of high-priority tasks. However, several problems surface as the number of tasks increases. One problem is the algorithmic singularities introduced by the usage of multiple levels of projections [Peters et al., 2008, Escande et al., 2014]. Most algorithms are designed under the assumption of singular-free conditions. But this assumption is unlikely to hold in practice, especially when there are a large number of tasks, and the system can easily become unstable if the algorithmic singularities occur. In addition, it is possible that secondary controllers, e.g. collision avoidance controllers, become the ones with high-priorities and the primary task can not be achieved. While several heuristics [Dietrich et al., 2012, Lee et al., 2012] have been proposed to shift the control priorities dynamically, whether such systems can be globally stabilized in presence of the algorithmic singularities is still an open question [Dietrich et al., 2018].

Stochastic motion policies Another related framework for multi-objective problem is to model single-task policies as stochastic policies, and combine them through the product-of-expert rule [Paraschos et al., 2013, Urain et al., 2021]. Stochastic policies can be better suited to model multi-modal tasks, and hence could be less prone to local optima. However, it is more challenging to establish stability results for the resulting policy.

Chapter 4

PARAMETRIZING RMP^2 POLICIES

So far in Chapter 3, we have been viewing RMP as a framework for control, in other words, we need to *hand-specify* all subtask RMPs and subtask maps so that the configuration space policy can achieve these subtasks. In this chapter and the next one, we will show that RMP can be used to construct a *structured* policy class, which we call RMP^2 policies, for robot learning. With RMP^2 policies, we can use configuration space policies generated by RMP^2 as a policy class, instead of directly using, e.g., a fully-connected neural network which maps joint positions and velocities to joint accelerations.

Similar to a fully connected neural network policy, RMP^2 policies have the following desirable properties which make them suitable for learning:

1. RMP^2 policies are differentiable. As we discussed in Section 3.2.2, RMP^2 can be implemented by automatic differentiation libraries. Therefore, RMP^2 policies can be learned end-to-end "like" a fully connected neural network.

2. RMP^2 policies are relatively fast to inference. It can run at around 300 Hz for a Franka Emika Panda robot with around 150 control points [Li et al., 2021].

3. RMP^2 policies are as expressive as an end-to-end fully connected neural network directly parameterizing the configuration space policy. This is because a configuration space policy is an RMP^2 policy with one identity subtask space.

Moreover, RMP^2 policies have additional benefits for learning, when compared with fully-connected neural network policies:

1. RMP^2 policies can make use of problem structure, including kinematics of the robot and the task decomposition. This often allows RMP^2 policies to learn more *efficiently*, in terms of number of samples needed to learn a good policy.

2. As is shown in Section 3.2.3, RMP^2 policies have stability properties, which can be used to enforce *safety* and *performance guarantees* of the learned policy.

In this chapter, we will discuss how to parameterize RMP^2 policies, namely, what parts of RMP^2 can be represented by, e.g., neural networks, and be updated during learning. Recall from Section 3.2 that the RMP^2 policy is given by,

$$\ddot{\mathbf{q}} = \left(\sum_{k=1}^{K} \mathbf{J}_k(\mathbf{q})^\top \mathbf{M}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k) \mathbf{J}_k(\mathbf{q}) \right)^\dagger \left(\sum_{k=1}^{K} \mathbf{J}_k(\mathbf{q})^\top \mathbf{M}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k) \left(\mathbf{a}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k) - \mathbf{c}_k(\mathbf{q}, \dot{\mathbf{q}}) \right) \right).$$

This means that there are two group of entities that can be represented as, e.g., neural networks. *Subtask maps:* One can also represent the subtask maps $\{\psi_k\}_{k=1}^K$ as neural networks. The subtask map affects $\mathbf{x}_k = \psi_k(\mathbf{q})$, $\dot{\mathbf{x}}_k = \mathbf{J}_k(\mathbf{q})\,\dot{\mathbf{q}}$, $\mathbf{J}_k(\mathbf{q})$ and $\mathbf{c}_k(\mathbf{q}, \dot{\mathbf{q}})$ in the equation above. We will cover an approach to parameterize a subtask map in Section 4.2. *Subtask RMPs:* Subtask RMPs $\{(\mathbf{M}_k, \mathbf{a}_k)\}_{k=1}^K$ can be also be represented as neural networks. In Section 4.3, we will discuss two methods of representing subtask RMPs as neural network, paramterizing through GDSs and a direct approach.

It should be noted that parameterizing subtask maps and RMPs are *not* mutually exclusive. One can parameterizing both a subtask RMP as well as its map. In Chapter 5, we will provide an example on jointly parameterizing subtask RMPs and maps.

Moreover, one does not need to (and probably would not wish to) represent *all* subtask maps and/or *all* subtask RMPs. In fact, if all subtask maps and RMPs are parameterized, we lose all benefit of using an RMP^2 policy as opposed to a fully-connected neural network. The main benefit of using RMP^2 policy is to encode known problem structure, e.g., robot kinematics and task decomposition into the policy class so that we do not need to "re-learn" such structure from data. Therefore, in this chapter, we make the following design choices so that we can make use as much known problem structure of possible.

1. **Main consideration on parameterization:** The main consideration behind the parameterization strategy is such that the stability results (Theorem 3.2.1) is meaningful. This means that we need the assumptions in Theorem 3.2.1 to hold, and we also need the stability property to translate to desirable properties, such as safety, motion feasibility and (through convergence to the invariant set) convergence to goal.

2. **Choice of subtasks to parameterize:** We only parameterize subtask maps or RMPs for the target reaching subtasks, i.e., subtasks which demand a certain point[1] on the robot to move in a certain way and eventually reach a target[2]. This means that we keep subtask maps and RMPs hand-specified for subtasks that ensures safety and feasibility of the robot motion, e.g., collision avoidance, joint limits and velocity limits, etc. With properly specified subtask maps and RMPs for those tasks, Theorem 3.2.1 implies safety (collision-free) and feasibility (respecting joint limits and velocity limits) of the robot motion [Cheng et al., 2018, 2020, Li et al., 2019a].

Remark For notational clarity, we will drop the subtask space subscript k in the remainder of this chapter. We will be referring to a particular subtask unless otherwise stated.

4.1 Target Reaching Subtasks

A target reaching subtask requires the a certain point of a robot to reach a target (in a particular way). Target reaching is an important component of a lot robotics problems. We have seen examples of target reaching subtasks earlier in Chapter 3, for example, we need the tip of the three link robot in Fig. 3.1a to reach the green target. In Fig. 4.1, we show two more examples of target reaching subtasks on a Franka Emika Panda robot. The first example requires the robot to, start from inside the cabinet, reach the handle of the cabinet door. In the second example, the robot closes the drawer through moving its end-effector.

Cheng et al. [2018, 2020] propose a few designs of target reaching RMPs, which allows the importance weight matrix $\mathbf{M}$ to adapt as the point of interest get close to the target. However, these target reaching RMPs are designed such that the point of interest move to the target in a *straight-line* fashion[3]. As is illustrated in Fig. 4.1, in practice, straight-line target reaching policy may not be desirable. For example, for the cabinet door reaching task, straight-line motion would make the robot to move *through* the cabinet door, causing

[1] Different subtasks may consider different points on the robot.

[2] Here we only consider the position of the point, but we can generalize the idea to handle orientation target as well.

[3] Although the point does not necessarily move in straight line under the configuration space policy generated by RMP^2 due to the existence of other subtasks.

(a) Cabinet door reaching

(b) Drawer closing

Figure 4.1: Target reaching, which requires the a certain point of a robot to reach a target in a particular way, is an important component of a lot robotics problems. (a) The robot reaches the cabinet door handle with its gripper from inside the cabinet. (b) The robot uses end-effector to close a drawer.

conflict with collision avoidance subtasks. For the drawer closing, we need the robot to first make contact with the drawer, then move horizontally to close the drawer. Moving in a straight-line manner could mean that the end-effector of the robot never makes a good contact with the face of the drawer. Hand-designing these target reaching subtask maps and RMPs could be tedious and difficult. Therefore, it is desirable to *parameterize* subtask maps and RMPs for the target reaching subtask so that we can *learn* a good target reaching behavior from data.

For target reaching, an important property is whether the point of interest eventually reach the target, no matter where it starts. Recall from Chapter 2 that this can be formalized to a notion of stability called *global asymptotic stability*. The main consideration behind most parameterizations of RMP^2 policies discussed in this chapter is to ensure that for the target reaching subtask, at least when not combined with other subtask, the goal is globally asymptotically stable. In the remaining of this chapter, we will explore a few options of ensuring global asymptotic stability of the goal.

4.2 Parameterizing Subtask Maps

When parameterizing the *subtask map* for a particular target reaching subtask, we always consider subtask map in the form of $\psi = (\varphi_\theta - \varphi_\theta(\mathbf{0})) \circ (\texttt{FK}(\cdot) - \mathbf{g})$, where φ_θ is a neural network parameterizing a map, FK is the forward kinematics map to the point of interest and $\mathbf{g}$ is the goal position. Such a parameterization of subtask maps allows us to make use of the kinematics of the robot. The shifting of $\varphi_\theta(\mathbf{0})$ and $\mathbf{g}$ is such that we can consider the stability property at $\mathbf{x} = 0$ regardless of the value of parameters θ and goal $\mathbf{g}$.

Although we can directly parameterize φ_θ as an arbitrary neural network, it is not desirable to do so. This is because an arbitrary map does not necessarily preserve global asymptotic convergence. This means that, even if the subtask RMPs ensures asymptotic stability with respect to the origin, our point of interest may not actually go to the goal. To see this, consider the map $\varphi_\theta \equiv \mathbf{0}$. Convergence to the origin in such a subtask space is not meaningful, as for any $\mathbf{q}$ we have $\mathbf{x} = \mathbf{0}$.

Fortunately, if the map φ_θ is a so-called *diffeomorphism* (we will formally define it soon), it does preserve global asymptotic convergence. In other words, as long as our subtask RMPs ensure that the origin is globally asympotic stable, then our point of interest will globally asymptotically converge to the goal. In Section 4.2.1, we will formally define diffeomorphisms and discuss why diffeomorphism can preserve stability properties. In Section 4.2.2, we introduce a neural network architecture inspired by Dinh et al. [2016] for parameterizing diffeomorphisms. This section is adapted from [Rana et al., 2020a].

4.2.1 Change of Coordinates for Dynamical Systems

Consider a bijective map $\varphi : \mathbb{R}^n \to \mathbb{R}^n$. The bijective map φ is a *diffeomorphism* if both the map φ and its inverse map φ^{-1} are continuously differentiable. We further assume that the diffeomorphism φ is bounded, namely that it maps bounded vectors to bounded vectors. A diffeomorphism can be used to describe a *change of coordinates* for differentiable manifolds.

As such, a dynamical system[4]

$$\dot{\mathbf{x}} = f(\mathbf{x}) \tag{4.1}$$

can be described under $\mathbf{y} \coloneqq \varphi(\mathbf{x})$ as,

$$\dot{\mathbf{y}} = \left[\frac{\partial \varphi}{\partial \mathbf{x}} f(\mathbf{x})\right]_{\mathbf{x}=\varphi^{-1}(\mathbf{y})} = \mathbf{J}_{\varphi}(\varphi^{-1}(\mathbf{y}))\, f(\varphi^{-1}(\mathbf{y})) \quad \coloneqq \tilde{f}(\mathbf{y}, \mathbf{u}), \tag{4.2}$$

where $\mathbf{J}_{\varphi}(\mathbf{x}) = \frac{\partial \varphi}{\partial \mathbf{x}}$ is the Jacobian of φ. In the special case where both the system dynamics f and the diffeomorphism φ are linear maps, this change of coordinates reduces to *change of basis* for linear dynamical systems Hespanha [2018].

The two dynamical systems (4.1) and (4.2) are descriptions of *the same internal dynamical system* evolving on a manifold under two coordinates $\mathbf{x}$ and $\mathbf{y}$. Therefore, the two systems share stability properties. Assume the existence of a Lyapunov function $V(\mathbf{x})$ showing that the equilibrium point $\mathbf{x}^*$ is globally asymptotically stable. Then, the diffeomorphism φ defines a Lyapunov function $\tilde{V} : \mathbf{y} \mapsto V(\varphi^{-1}(\mathbf{y}))$,

$$\begin{aligned} \dot{\tilde{V}}(\mathbf{y}) &= \left[\frac{\partial V}{\partial \mathbf{x}} \frac{\partial \varphi^{-1}}{\partial \mathbf{y}} \dot{\mathbf{y}}\right]_{\mathbf{x}=\varphi^{-1}(\mathbf{y})} = \left[\frac{\partial V}{\partial \mathbf{x}} \left(\mathbf{J}_{\varphi}(\mathbf{x})\right)^{-1} \tilde{f}(\mathbf{y})\right]_{\mathbf{x}=\varphi^{-1}(\mathbf{y})} \\ &= \left[\frac{\partial V}{\partial \mathbf{x}} \left(\mathbf{J}_{\varphi}(\mathbf{x})\right)^{-1} \mathbf{J}_{\varphi}(\mathbf{x})\, f(\mathbf{x})\right]_{\mathbf{x}=\varphi^{-1}(\mathbf{y})} = \left[\frac{\partial V}{\partial \mathbf{x}} \dot{\mathbf{x}}\right]_{x=\varphi^{-1}(\mathbf{y})} = \dot{V}(\varphi^{-1}(\mathbf{y})), \end{aligned} \tag{4.3}$$

where the second equality follows from the implicit function theorem. Therefore, the system after the change of coordinate (4.2) has a globally asymptotically stable equilibrium point $\mathbf{y}^* = \varphi(\mathbf{x}^*)$. Moreover, since the diffeomorphism is bijective, the converse is also true: if there exists a Lyapunov function $\tilde{V}$ under coordinate $\mathbf{y}$, equilibrium point $\mathbf{x}^*$ is globally asymptotically stable. Therefore, the point of interest $\texttt{FK}(\mathbf{q})$ converges asymptotically to $\mathbf{g}$ if the subtask RMPs renders the system converging asymptotically to the origin.

As an example, Fig. 4.2 shows the vector fields (yellow) and trajectories (red) generated by the same underlying system observed in two coordinate systems (related through a diffeomorphism which maps the origin to the origin). Under the coordinate for the left figure, the system evolves in straight line to the origin, while on the right, the system evolves in curved trajectories. However, the two systems share the stability property, i.e., the origin is globally asymptotically stable for both systems.

[4]In chapter Chapter 2, we consider dynamical system in the form of $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$. Here we assume that we have plugged in the control input as a function of the state.

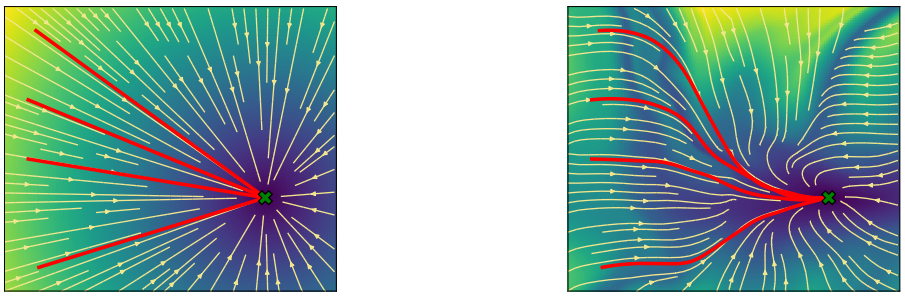


Figure 4.2: Vector fields (yellow) and trajectories (red) generated by two systems under a change of coordinates given by a diffeomorphism that maps the origin to the origin. The origin is marked as a green cross. Although the vector fields and trajectories look different, the two systems share stability properties: the origin is globally asymptotically stable for both systems.

The example above tells us that, with a sufficiently expressive diffeomorphism map φ_θ, we can achieve complex target reaching behavior, like the ones showing on the right, even with simple straight-line behavior on the subtask space, e.g., the ones proposed by Cheng et al. [2018, 2020]. Parameterizing the diffeomorphic subtask map allows us to learn systems which can generate complex target reaching motion with stability guarantees, while only needing to analyze the stability property of a simpler system on the subtask space. Next, we will propose a neural network architecture called *Euclideanizing flow*, which ensures that the map is a diffeomorphism.

4.2.2 Parameterizing Subtask Map via Euclideanizing Flow

Note that the composition of two diffeomorphisms is also a diffeomorphism. Therefore, we parameterize the map φ_θ as a composition of M (simpler) diffeomorphisms $\varphi_\theta = \varphi_1 \circ \varphi_2 \circ \cdots \circ \varphi_M$. For notational simplicity, we omit the subscript θ for each simpler diffeomorphism $\varphi_m : \mathbb{R}^n \to \mathbb{R}^n$, though readers should note that there are learnable parameters for each of the simpler diffeomorphisms $\{\varphi_m\}_{m=1}^M$.

We parameterize each map φ_m as a *coupling layer* [Dinh et al., 2016]. Let $\mathbf{y}_m \in \mathbb{R}^n$ denote the output from the m-th coupling layer φ_m, i.e., $\mathbf{y}_m = \varphi_m(\mathbf{y}_m)$, $\mathbf{y}_0 = \mathbf{x}$, and

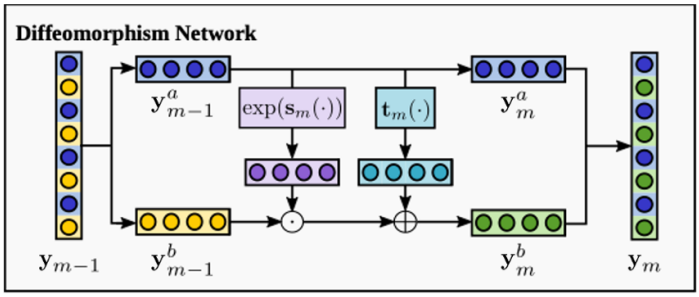


Figure 4.3: Architecture of the diffeomorphic mapping network for Euclideanizing flow. The network architecture is inspired by Dinh et al. [2016],

$\mathbf{y}_m = \mathbf{y}$. Each coupling layer, φ_m, involves splitting the input $\mathbf{y}_{m-1}$ followed by scaling and translating one of the parts. The `split` operation divides inputs into two parts, $\mathbf{y}^a_{m-1} \in \mathbb{R}^{\lfloor n/2 \rfloor}$ and $\mathbf{y}^b_{m-1} \in \mathbb{R}^{\lceil n/2 \rceil}$, constituting alternate input dimensions, whereby the pattern of alternation is reversed after each layer. Mathematically[5],

$$\mathbf{y}_m = \begin{bmatrix} \mathbf{y}^a_m \\ \mathbf{y}^b_m \end{bmatrix} = \begin{bmatrix} \mathbf{y}^a_{m-1} \\ \mathbf{y}^b_{m-1} \odot \exp\left(s_m(\mathbf{y}^a_{m-1})\right) + t_m(\mathbf{y}^a_{m-1}) \end{bmatrix} \coloneqq \varphi_m(\mathbf{y}_{m-1}), \tag{4.4}$$

where $\odot$ denotes pointwise product and exp denotes pointwise exponential. The functions $s_m : \mathbb{R}^{\lfloor n/2 \rfloor} \to \mathbb{R}^{\lceil n/2 \rceil}$ and $t_m : \mathbb{R}^{\lfloor n/2 \rfloor} \to \mathbb{R}^{\lceil n/2 \rceil}$ are the parameterized scaling and translation functions, respectively. The coupling layer is a bijective affine map by construction. Since each map is bijective, the composed map φ_θ is also guaranteed to be bijective. Further, the composed map φ_θ is continuously differentiable as long as the scaling and translation functions in each coupling layer are continuously differentiable. This formulation of a bijection has been previously employed for density estimation by Dinh et al. [2016], where the scaling and translation functions were given by deep convolutional neural networks.

In contrast to Dinh et al. [2016], for the scaling and translation functions, we use single-layer neural networks with the layer resembling an approximated kernel machine [Rahimi and Recht, 2008]. This special network structure leverage the advantages of both kernel

[5]Technically, $\mathbf{y}_m$ is not a direct concatenation of $\mathbf{y}^a_m$ and $\mathbf{y}^b_m$, it should be merged in an alternative manner.

machines and neural networks:

- The kernel functions act as regularizers to enforce desired properties on the learned function, e.g. smoothness, which can further improve the sample efficiency of the learning algorithm; and

- As a parameterized model, the composed diffeomorphism can be efficiently trained using learning techniques for neural networks.

We use a matrix-valued Gaussian separable kernel, defined as $K(\mathbf{z}, \mathbf{z}') \coloneqq \exp(-\frac{\|\mathbf{z}-\mathbf{z}'\|^2}{2l^2})\,\mathbb{I}$, where l is the length-scale of the kernel. The Gaussian kernel restricts the hypothesis class to the class of C^∞ vector-valued functions, imposing a stricter smoothness constraint than continuous differentiability, i.e. C^1. This is desirable since human motions are known to be maximizing smoothness (or minimizing jerk) [Flash and Hogan, 1985]. We approximate the kernel by N randomly sampled Fourier features [Rahimi and Recht, 2008], such that the scaling and translation functions are given by linear combinations of these features,

$$g(\mathbf{z}) = \phi(\mathbf{z})^\top \mathbf{w}, \qquad \text{with } \phi(\mathbf{z}) = \sqrt{\frac{2}{N}} \begin{bmatrix} \cos(\boldsymbol{\alpha}_1^\top \mathbf{z} + \boldsymbol{\beta}_1) \\ \cos(\boldsymbol{\alpha}_2^\top \mathbf{z} + \boldsymbol{\beta}_2) \\ \vdots \\ \cos(\boldsymbol{\alpha}_N^\top \mathbf{z} + \boldsymbol{\beta}_N) \end{bmatrix} \otimes \mathbb{I} \in \mathbb{R}^{(N\cdot\lceil n/2 \rceil)\times\lceil n/2 \rceil}, \quad (4.5)$$

where $\mathbf{w} \in \mathbb{R}^{(N\cdot\lceil n/2 \rceil)}$ is the learnable parameters. The projection vector $\phi(\mathbf{z})$ is composed of N randomly sampled Fourier features such that the kernel matrix is given by $K(\mathbf{z}, \mathbf{z}') \approx \varphi(\mathbf{z})^\top \varphi(\mathbf{z}')$. Concretely, the coefficients $\{\boldsymbol{\alpha}_i\}_{i=1}^N$ are sampled from a zero-mean Gaussian distribution $\mathcal{N}(\mathbf{0}, l^{-2}\,\mathbb{I})$, and the bias terms $\{\boldsymbol{\beta}_i\}_{i=1}^N$ are sampled from a uniform distribution $\mathrm{U}(0, 2\pi)$.

Under the formulation in (4.5), we define $s_m(\cdot) = \varphi(\cdot)^\top \mathbf{w}_{s_k}$ and $t_m(\cdot) = \varphi(\cdot)^\top \mathbf{w}_{t_k}$. The set of parameters in the diffeomorphic map is therefore given by $\theta \coloneqq \{(\mathbf{w}_{s_m}, \mathbf{w}_{t_m})\}_{m=1}^M$.

Concrete Examples

Here we show a few examples that Euclideanizing flow can be used to parameterize subtask maps such that simple straight-line motion on the subtask space can imply complex behav-

ior. We consider the LASA dataset by Khansari-Zadeh [2019], which consists of a library of 30 two-dimensional handwritten letters, each with 7 demonstrations. We consider a 2-d point robot, meaning that `FK` is the identity mapping. We also shift the letters such as the end of the trajectories are at the origin.

For each letter, we parameterize the subtask map as a Euclideanizing flow network with $M = 10$ coupling layers constituting of $N = 200$ random Fourier features with length-scale $l = 0.45$. We learn the subtask maps through the techniques which we will discuss in Chapter 5.

Fig. 4.4 shows the vector fields on the configuration space, which are given by straight-line vector fields on the subtask space, on a subset of letters from the LASA dataset. In all the plots, the trajectories (red) closely match the demonstrations (white). All trajectories converge to the goals shown in green crosses. These examples show that the parameterization described in this section is expressive enough to generate complex configuration space behaviors even from simple straight-line subtask space behaviors.

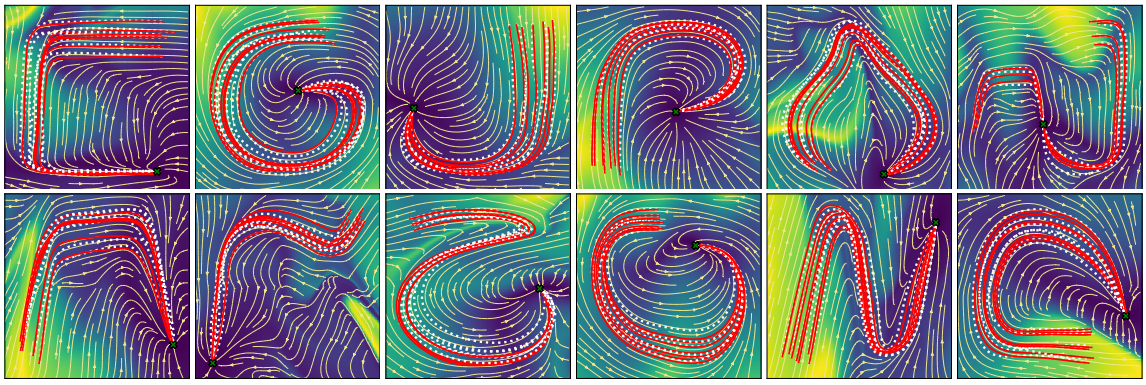


Figure 4.4: Vector fields of the dynamics learned on the LASA dataset, alongside demonstrations (white) and reproductions (red). The vector fields are generated by straight-line motions in the subtask space.

4.3 Parameterizing Subtask RMPs

Another way to parameterizing RMP^2 policies is through parameterizing each target reaching RMPs. In Section 4.3.1, we will introduce a few ways to parameterize major components of a GDS, including metric and potential function. In Section 4.3.2, we will briefly discuss directly parameterizing the importance weight and acceleration policies in an RMP. Section 4.3.1 is adapted from [Rana et al., 2019].

4.3.1 Parameterizing GDSs

Let $\mathbf{x}$ be a generalized coordinate on the subtask space. For simplicity, we consider Riemannian metrics $\mathbf{G}(\mathbf{x})$ that only depend on positions (not velocities). Recall from (3.18) in Chapter 3 that a GDS with positional dependent metric $\mathbf{G}(\mathbf{x})$ takes in the following form,

$$\mathbf{G}(\mathbf{x})\,\ddot{\mathbf{x}} + \mathbf{C}_{\mathbf{G}}(\mathbf{x},\dot{\mathbf{x}})\,\dot{\mathbf{x}} = -\nabla_{\mathbf{x}}\Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x},\dot{\mathbf{x}})\,\dot{\mathbf{x}},$$

and the corresponding RMP is given by

$$\begin{aligned}\mathbf{M}(\mathbf{x},\dot{\mathbf{x}}) &= \mathbf{G}(\mathbf{x}) \\ \mathbf{a}(\mathbf{x},\dot{\mathbf{x}}) &= \mathbf{G}(\mathbf{x})^{-1}\big(-\nabla_{\mathbf{x}}\Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x},\dot{\mathbf{x}})\,\dot{\mathbf{x}} - \mathbf{C}_{\mathbf{G}}(\mathbf{x},\dot{\mathbf{x}})\,\dot{\mathbf{x}}\big).\end{aligned} \tag{4.6}$$

We observe here that the desired motion generated by the system (3.18) is mainly governed by the component $-\mathbf{G}(\mathbf{x})^{-1}\nabla\Phi(\mathbf{x})$, which can be seen as setting $\mathbf{a}$ along the negative *natural gradient* of the potential function. The remaining components: the *damping* acceleration, $-\mathbf{G}(\mathbf{x})^{-1}\,\mathbf{B}(\mathbf{x},\dot{\mathbf{x}})\,\dot{\mathbf{x}}$, and the *curvature* curvature, $-\mathbf{G}(\mathbf{x})^{-1}\,\mathbf{C}_{\mathbf{G}}(\mathbf{x},\dot{\mathbf{x}})\,\dot{\mathbf{x}}$, simply ensure stable and geometrically consistent behavior.

Hence, we choose to focus on parameterizing the *metric* and *potential function* of the system (3.18). We pre-define the damping matrix as $\mathbf{B}(\mathbf{x},\dot{\mathbf{x}}) = \gamma_d\,\mathbf{G}(\mathbf{x})$, such that the damping acceleration $\mathbf{G}(\mathbf{x})^{-1}\,\mathbf{B}(\mathbf{x},\dot{\mathbf{x}}) \equiv \gamma_d\,\mathbb{I}$ is always independent of the choice of metric.

Hence, we require the parameterized Riemannian metric to be globally positive definite and the parameterized potential to have only one minima at the origin[6] (without any other stationary points). As is shown in Lemma 3.2.1, for $\mathbf{G}(\mathbf{x}), \mathbf{B}(\mathbf{x},\dot{\mathbf{x}}) \in \mathbb{R}^{m\times m}_{+}$, the system

[6]Recall that the subtask map for target reaching is shifted by $\mathbf{g}$.

(3.18) converges to the invariant set $\mathcal{C}_\infty \coloneqq \{(\mathbf{x}, \dot{\mathbf{x}}) : \nabla\Phi(\mathbf{x}) = 0, \dot{\mathbf{x}} = 0\}$. This means that the subtask system asympototically converges to the origin.

Parameterizing Riemannian Metric

To ensure the stability of the parameterized system, the Riemannian metric $\mathbf{G}(\mathbf{x}) = \mathbf{M}(\mathbf{x})$ needs to be positive definite. We introduce two parameterizations below. One is through parameterizing the Cholesky decomposition of $\mathbf{G}(\mathbf{x})$, and the other is through representing $\mathbf{G}(\mathbf{x})$ as the Gram matrix of a parameterized matrix $\mathbf{A}(\mathbf{x})$.

Cholesky decomposition parameterization: One way of ensuring positive definiteness is to parameterize the metric by its Cholesky decomposition, $\mathbf{G}(\mathbf{x}) = \mathbf{L}(\mathbf{x})\,\mathbf{L}(\mathbf{x})^\top$, where $\mathbf{L} \in \mathbb{R}^{m\times m}$ is a lower-triangular matrix with positive diagonal entries. Let $\mathbf{l}_d(\mathbf{x}) \in \mathbb{R}^m$ and $\mathbf{l}_o(\mathbf{x}) \in \mathbb{R}^{\frac{1}{2}(m^2-m)}$ denote the vector given by collecting the diagonal and off-diagonal entries of $\mathbf{L}(\mathbf{x})$, respectively. In order for $\mathbf{L}(\mathbf{x})$ to be a Cholesky decomposition of the positive definite matrix $\mathbf{G}(\mathbf{x})$, we require each entries of $\mathbf{l}_d(\mathbf{x})$ to be strictly positive.

We can represent the metric matrix as a neural-network with parameter θ. The neural network takes in the coordinate $\mathbf{x} \in \mathbb{R}^n$ and outputs $\mathbf{l}_d(\mathbf{x};\theta)$ and $\mathbf{l}_o(\mathbf{x};\theta)$ (Fig. 4.5). To ensure that $\mathbf{l}_d(\mathbf{x};\theta)$ is strictly positive for all $\mathbf{x} \in \mathbb{R}^n$, we take absolute value of the output of the linear layer and add it with a small positive offset $\epsilon > 0$. Hence, lower-triangular matrix $\mathbf{L}(\mathbf{x};\theta)$ is always invertible and the Riemannian metric $\mathbf{G}(\mathbf{x};\theta)$ is guaranteed to be positive definite.

Gram matrix parameterization: Another way is to parameterize the metric as

$$\mathbf{G}(\mathbf{x};\theta) = \mathbf{A}(\mathbf{x};\theta)^\top\,\mathbf{A}(\mathbf{x};\theta) + \epsilon\,\mathbb{I}_{m\times m},$$

where $\epsilon > 0$ is a small positive value ensuring the positive definiteness of $\mathbf{G}(\mathbf{x};\theta)$, as $\mathbf{A}^\top\mathbf{A}$ is always positive semi-definite. The neural network architecture is similar to Fig. 4.5, except that the network directly outputs a flattened $\mathbf{A}$ matrix in $\mathbb{R}^{m^2}$ (without splitting into diagonal and off-diagonal parts, etc.).

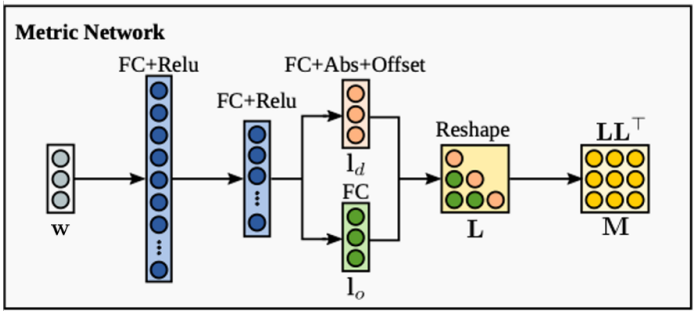


Figure 4.5: The structure of the neural network for metric learning. The first two layers are fully connected layers with ReLU activation functions. The diagonal and off-diagonal elements of the lower triangular matrix $\mathbf{L}(\mathbf{x};\theta)$ is then predicted through another fully connected layer. In order to ensure that the diagonal elements are strictly positive, the absolute value of the output of the layer is taken and a positive offset is added.

Parameterizing Potential Functions

The main consideration behind parameterizing a potential function is to ensure that it is continuously differentiable, with the only stationary point at the origin. Then, by Lemma 3.2.1, the system (3.18) globally asymptotically converges to the origin.

Diffeomorphism parameterization: One way to parameterize a potential function, which is inspired by diffeomorphism discussed in Section 4.2.1, is to parameterize it as $\Phi = \bar{\Phi} \circ \varphi_\theta$, where $\bar{\Phi}$ is a simple potential function with the only stationary point at the origin, e.g., $\bar{\Phi}(\mathbf{y}) = \frac{1}{2}\|\mathbf{y}\|^2$, and φ_θ is a parameterized diffeomorphism map (e.g., a Euclideanizing flow network introduced in Section 4.2.2) which maps origin to the origin. By the chain rule of calculus,

$$\nabla_{\mathbf{x}}\Phi = \mathbf{J}_{\varphi_\theta}^\top \nabla_{\mathbf{y}}\bar{\Phi}. \tag{4.7}$$

Since φ_θ is a diffeomorphism, the Jacobian $\mathbf{J}_{\varphi_\theta}$ is always non-singular. Therefore, $\nabla_{\mathbf{x}}\Phi = \mathbf{0}$ if and only if $\nabla_{\mathbf{y}}\bar{\Phi} = \mathbf{0}$. Therefore, the only stationary point of Φ is at the origin.

Fig. 4.6 shows the iso-contours of a simple potential function $\bar{\Phi}$ which generate equidistant iso-contours around the origin, and the corresponding potential function $\Phi = \bar{\Phi} \circ \varphi_\theta$.

The diffeomorphism φ_θ makes Φ a complex potential function which has only one stationary point, which is at the origin. Similarly, the iso-contours of potential functions learned from the LASA handwriting dataset [Khansari-Zadeh, 2019] is shown in Fig. 4.7.

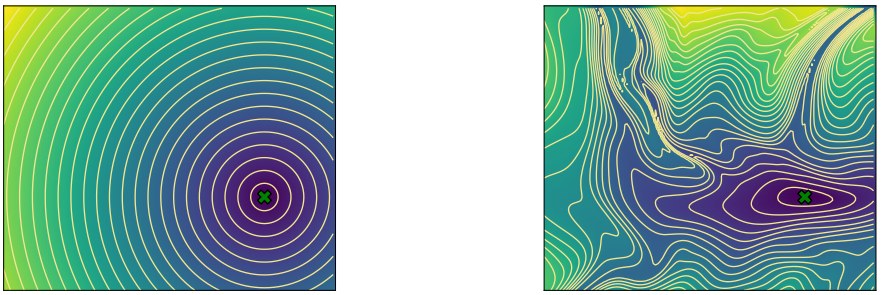


Figure 4.6: Iso-contours of $\bar{\Phi}$ and $\Phi = \bar{\Phi} \circ \varphi_\theta$. Despite that Φ has complex iso-contours, the origin its only stationary point.

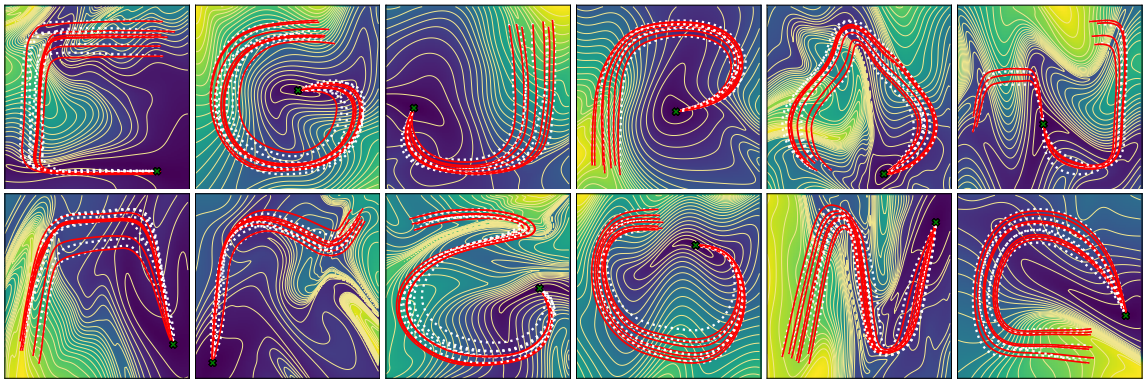


Figure 4.7: Iso-contours of the potential functions learned on the LASA dataset. The origin is the only stationary point for all potential functions

Nominal trajectory parameterziation: Suppose that we are given a nominal trajectory $\boldsymbol{\zeta} = (\boldsymbol{\zeta}_t)_{t=1}^T$ which specifies the desired behavior of the system (3.18). Such a trajectory can be given by, e.g., a human demonstration or a motion planner.

Given the nominal trajectory demonstration $\boldsymbol{\zeta}$, we can directly parameterize a potential $\Phi(\mathbf{x})$, which generates a dissipative field $-\nabla\Phi(\mathbf{x})$ that produces motions which: *(i)* converge smoothly towards the nominal trajectory, and *(ii)* follow the remaining trajectory after convergence. Furthermore, to ensure stability, we also enforce

$$\boldsymbol{\zeta}_T = \mathbf{0} \qquad \text{and} \qquad \Phi(\mathbf{x}) \to \infty \text{ as } \mathbf{x} \to \infty, \tag{4.8}$$

so that the trajectories can always be bounded following the dynamics (3.18). Fig. 4.8a shows a vector field generated by the negative gradient of an example potential field with aforementioned properties. We use an approach similar to that used by Khansari-Zadeh and Khatib [2017]. Specifically, the potential is given by a convex combination of potential elements centered at each trajectory point $\boldsymbol{\zeta}_t$,

$$\Phi(\mathbf{x}) \coloneqq \sum_{t=1}^{t=T} w_t(\mathbf{x})\phi_t(\mathbf{x}), \quad \text{where } w_t(\mathbf{x}) = \frac{k(\mathbf{x}, \boldsymbol{\zeta}_t)}{\sum_{t'=1}^{t'=T} k(\mathbf{x}, \boldsymbol{\zeta}_{t'})}, \quad k(\mathbf{x}, \boldsymbol{\zeta}_t) = e^{-\frac{\|\mathbf{x}-\boldsymbol{\zeta}_t\|^2}{2\sigma^2}}, \tag{4.9}$$

where $\sigma > 0$ is the length-scale of the kernel k. Furthermore, each contributing potential element here is a summation of two components: $\phi_t(\mathbf{x}) = \phi_t^{\perp}(\mathbf{x}) + \phi_t^0$, whereby the component $\phi_t^{\perp} : \mathbb{R}^m \mapsto \mathbb{R}_+$ is a strictly convex function with a global minima at $\boldsymbol{\zeta}_t$ and $\phi_t^0 \in \mathbb{R}_+$ is a bias term. As a consequence of the aforementioned decomposition, the gradient of the overall warped potential in (4.9) can also be decomposed as,

$$\nabla\Phi(\mathbf{x}) = \nabla\Phi^{\perp}(\mathbf{x}) + \nabla\Phi^{\parallel}(\mathbf{x}) \tag{4.10}$$

where the gradient component $\nabla\Phi^{\perp}(\mathbf{x})$ causes attractive pull towards the nominal trajectory while the component $\nabla\Phi^{\parallel}(\mathbf{x})$ produces accelerations along the nominal trajectory. The motion along the nominal trajectory is direct consequence of monotonically decreasing bias terms ϕ_t^0, such that the negative of the potential gradient aligns with the nominal trajectory. Due to this decomposition, we independently hand-design the function $\phi_t^{\perp}(\mathbf{x})$ as per our desired attractive accelerations towards the nominal trajectory. Next, we solve for the bias terms ϕ_t^0 such that the direction of motion governed by the potential at each point $\boldsymbol{\zeta}_t$ matches the nominal trajectory.

As a first step in this procedure, we choose to go with the following attractive potential element proposed by Cheng et al. [2018],

$$\phi_t^{\perp}(\mathbf{x}) = \frac{1}{\eta}\log\left(e^{\eta\|\mathbf{x}-\boldsymbol{\zeta}_t\|} + e^{-\eta\|\mathbf{x}-\boldsymbol{\zeta}_t\|}\right), \quad \nabla\phi_t^{\perp}(\mathbf{x}) = s_\eta\big(\|\mathbf{x}-\boldsymbol{\zeta}_t\|\big)\frac{\mathbf{x}-\boldsymbol{\zeta}_t}{\|\mathbf{x}-\boldsymbol{\zeta}_t\|} \tag{4.11}$$

where $\eta > 0$ defines the effective smoothing radius of the function at the origin and $s_\eta(0) = 0$ and $s_\eta(r) \to 1$ as $r \to 0$. For a sufficiently large η, this choice of potential function ensures that the attractive acceleration always has a unit magnitude except in the neighborhood of the center $\boldsymbol{\zeta}_t$ where it smoothly decreases to zero. A trivial alternative to this function is a quadratic as used by Khansari-Zadeh and Khatib [2017]. However, the gradient of a quadratic function increases linearly with distance which can cause undesirably large accelerations far away from the nominal trajectory.

Towards the second step in the procedure, we solve for the bias terms $\{\phi_t^0\}$ through,

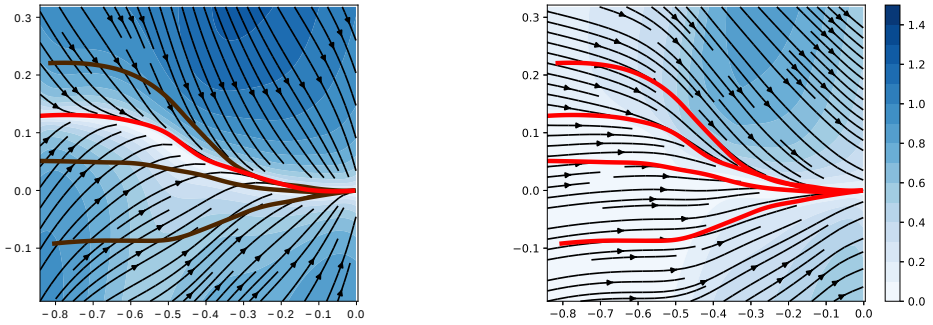
$$\begin{aligned} \min_{\{\boldsymbol{\phi}_t^0\}_{t=1}^T} \quad & \frac{1}{T}\sum_{t=1}^{t=T}\left\|\dot{\boldsymbol{\zeta}}_t + \nabla\Phi\big(\boldsymbol{\zeta}_t;\{\phi_t^0\}\big)\right\|^2 + \lambda\left\|\phi_t^0\right\|^2 \\ \text{s.t.} \quad & 0 \le \phi_T^0 \le \phi_{t+1}^0 \le \phi_t^0, \qquad \forall t = 1,\ldots,(T-1) \\ & \nabla_{\mathbf{x}}\Phi(\mathbf{0}) = \mathbf{0}, \end{aligned} \tag{4.12}$$

where $\lambda > 0$ is the regularization parameter. Furthermore, the optimization constraints enforce the potential to decrease monotonically along the nominal trajectory with a stationary point at the origin. The aforementioned optimization problem can be solved efficiently with off-the-shelf solvers, e.g. CVX [Grant et al., 2009].

In Fig. 4.8a, we show an example of solving for a potential function given a nominal trajectory highlighted in red. The system converges asymptotically to the origin. The vector field shows that all trajectories are attracted to the nominal trajectory and follows along the nominal trajectory. When combined with a learned Riemannian metric described earlier, the GDS can closely follow all four demonstration trajectories (Fig. 4.8b).

4.3.2 Directly Parameterizing RMPs

Although parameterizing an RMP through GDS can give us desirable stability properties, it is still possible to directly parameterize a subtask RMP $\big(\mathbf{M}(\mathbf{x},\dot{\mathbf{x}}), \mathbf{a}(\mathbf{x},\dot{\mathbf{x}})\big)$. To parameterize



(a) Learned potential function

(b) Learned potential function and metric

Figure 4.8: (a) Vector field introduced by the learned potential function: every trajectory are attracted to the nominal trajectory (red). (b) Vector field generated by the learned potential and Riemannian metric: all four demonstrations can be reproduced by the GDS (3.18).

the positive definite importance weight $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})$, either strategies for parameterizing the Riemannian metric described in Section 4.3.1 can be used. A fully-connected neural network can be used to represent $\mathbf{a} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$.

In this case, the subtask RMP does not have the global convergence property as provided by other parameterizations, but direct parameterization may still be desirable as it can be evaluated faster than other parameterizations. For example, in Chapter 6, we will use the direct parameterization in reinforcement learning, as a lot of inference (and back-propagation) is required throughout learning.

Suppose that there exsits a Riemannan metric $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ such that $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})$ satisfies (3.14), and there exists a potential function Φ such that the Lyapunov function (3.15) is decreasing, then the system is still guaranteed to be collision-free and feasible when combined with properly designed collision avoidance and joint limit subtask RMPs.

It is also possible to leverage the computational framework proposed in Section 3.2.6 and use a neural network to parameterize the nominal acceleration policy π_k (instead of $\mathbf{a}_k$). Then, $\mathbf{a}_k$ (or $\mathbf{f}_k$) can be given by solving the QP (3.32) which has a closed-form solution.

4.4 Conclusion

In this chapter, we start to think about RMP^2 policies as a structured policy class for learning. We discuss that RMP^2 policies might be more desirable than fully-connected neural networks, as RMP^2 policies can make use of known problem structure, such as robot kinematics, task decomposition, and partial solution (e.g., for collision avoidance or respecting joint limits).

There are two type of entities that can be parameterized in an RMP^2 policy, the subtask maps and subtask RMPs. To ensure the safety and feasibility of robot motion, we make the design choice of only parameterizing subtask maps and RMPs corresponding to target reaching subtasks, while keeping subtask maps and RMPs for, e.g., collision avoidance, joint limits, etc. hand-specified. Our parameterization for subtask map is based on the insight that diffeomorphism preserves stability properties, i.e., global convergence to goal. Finally, we discuss how to parameterize RMPs through parameterizing GDSs, and reason about the trade-offs between parameterizing GDSs and directly representing RMPs as fully-connected neural networks.

4.5 Related Work

A few other strategies of parameterizing RMP^2 policies have been proposed in the literature. Meng et al. [2019] propose a parameterization of RMPs similar to the direct approach discussed in this chapter. They leverage the symmetry and only predicts the lower triangular entries of the Riemannian metric. However, their parameterization does not guarantee that the learned Riemannian metric is positive definite. Mukadam et al. [2019] keep subtask policies and maps as hand-specified, but add scalar learnable weights for each subtask. This parameterization can preserve information from hand-designed policies and maps, but sometimes lack expressivity. Lee et al. [2020] propose to sequentially concatenate hand-designed subtask policies and model-free reinforcement learning policies. Shaw et al. [2022] combine RMP with variable impedance control. More recently, Xie et al. [2023] and Zhi et al. [2023] propose parameterization strategies for geometric fabrics [Ratliff et al., 2021, Van Wyk et al., 2022], an extension of the RMP framework.

It should be noted that the parameterization strategy discussed in this chapter can be applicable beyond the RMP framework. For example, Lutter et al. [2018] propose a neural network architecture for dynamics modeling similar to our Riemannian metric parameterization using its Cholesky decomposition. Urain et al. [2020] apply a similar idea of learning diffeomorphic mapping for learning stochastic policies.

Chapter 5

LEARNING RMP^2 POLICIES FROM HUMAN DEMONSTRATIONS

Learning from (human) demonstrations (LfD) [Argall et al., 2009] provides a paradigm for introducing robots to skills required for executing complex tasks. Hand-specifying such skills can otherwise be hard or infeasible, especially for inexperienced end-users. A given task may require *coordinated* and *adaptive* movements of different parts of the robot. For example, consider a wiping task. In addition to requiring the end-effector to maintain contact with the surface, it may be desirable to maintain a specific elbow orientation in order to effectively wipe the surface. Movements of the elbow and the end-effector are also inter-dependent, requiring coordination. All robot parts must also adapt to environmental changes, such as displacement of the surface to be wiped, or introduction of a new obstacle. Thus, each part of the robot can be viewed as executing multiple individual subtasks, dictating desired behaviors.

While previous work has explored learning goal-directed reactive policies from demonstrations [Khansari-Zadeh and Billard, 2011, Neumann and Steil, 2015, Ravichandar and Dani, 2019], these approaches cannot combine multiple policies, each dictating the behavior in a certain subtask space, while maintaining stability properties. By contrast, RMP^2 policies are particularly suitable as a policy class for multi-objective problems with multiple subtask spaces. As is discussed in Chapter 4, RMP^2 policies have inherent stability properties even when multiple subtask policies are being learned.

In this chapter, we present two case studies of learning RMP^2 policies from human demonstrations with multiple subtasks of interest. The first one learns each subtask policy individually, while the other one learns all subtask maps and policies jointly. Section 5.1 and Section 5.3 of this chapter are adapted from [Rana et al., 2020b]. Section 5.2 is adapted from [Rana et al., 2019].

5.1 Problem Statement

Consider the problem of kinesthetic teaching, where the human provide demonstrations by moving the robot, providing a number of trajectory demonstrations in the joint configuration space. Additionally, we allow the human to specify a number of *subtask spaces*[1], where the motion on the these subtask spaces is relevant to the achieving the overall task. For example, for a goal-reaching subtask, the user may specify, as a subtask space, the 3-dimensional Euclidean workspace of the end-effector position. The goal for our learning problem is to learn a parameterized policy π_θ which can generate motion similar to the human demonstrations when *viewed in these subtask spaces*.

Remark 5.1.1 (Velocity-based System). Despite that the RMP framework is defined for acceleration-based systems, in this chapter, we define learning problems on a simpler velocity-based system. This gives a simpler policy composition rule for RMPs $(\mathbf{M}_k, \mathbf{v}_k)$, where $\mathbf{v}_k$ is the desired *velocity* policy for the k-th subtask.

$$\dot{\mathbf{q}} = \pi_\theta(\mathbf{q}) = \min_{\mathbf{v}\in\mathbb{R}^d} \quad \sum_{k=1}^{K} \frac{1}{2} \parallel \mathbf{J}_k \mathbf{v} - \mathbf{v}_k \parallel^2_{\mathbf{M}_k} . \tag{5.1}$$

The configuration space policy $\dot{\mathbf{q}}$ can be computed based on a similar (but simpler) procedure as RMP[2]. This simplification to velocity-based system is mostly for computational efficiency and numerical stability as there is no need to compute the curvature terms. As is mentioned in Chapter 4, with the same choice of Riemannian metric and potential function, an acceleration-based system can generate similar motion as a velocity-based system given a proper damping term. In fact, the resulting robot motion in Section 5.2 is generated by an acceleration-based system, despite that the importance weights and potential functions are learned from velocity-based systems.

Consider N demonstration trajectories in the configuration space of the robot, each composed of T_i position-velocity pairs, denoted by $\{\{(\mathbf{q}^d_{i,t}, \dot{\mathbf{q}}^d_{i,t})\}_{t=1}^{T_i}\}_{i=1}^{N}$. On possible way of learning is to directly regress joint space velocity so that it matches the joint velocity of the

[1] In most existing learning from demonstrations literatureParaschos et al. [2017], Pastor et al. [2009], Khansari-Zadeh and Billard [2011], Neumann and Steil [2015], chaandar Ravichandar and Dani [2019], Rana et al. [2020a], only a single (subtask) space is considered, and it is usually either the configuration space, or the (3-d or 6-d) end-effector workspace.

demonstrations, i.e.,

$$\theta_{\mathcal{C}}^{\star} = \arg\min_{\theta} \underbrace{\sum_{i=1}^{N} \sum_{t=1}^{T_i} \left\| \dot{\mathbf{q}}_{i,t}^{d} - \pi^{\theta}\left(\mathbf{q}_{i,t}^{d}\right) \right\|^2}_{\mathcal{L}_{\mathcal{C}}(\theta)}. \tag{5.2}$$

Presumably, if the learned joint velocity policy perfectly matches the demonstrated joint velocities, it should also perfectly matches the demonstrations in the subtask spaces. However, in practice, there usually does not exist a policy such that $\mathcal{L}_{\mathcal{C}}(\theta) = 0$. This is because, when providing demonstrations, the human primarily cares about the motion on spaces that are directly relevant to achieving the task (i.e. the subtask spaces). As a result, the demonstrations can be conflicting in the joint space (providing vastly different velocities at the same joint position), even though they can be consistent in the subtask spaces. Therefore, directly regressing on the joint space trajectories (5.2) will produce large error both in the joint space and in the workspace.

In this chaper, we present two solutions for learning RMP^2 policies from human demonstrations. In Section 5.2, each subtask policy is *independently* learned to imitate the demonstrated trajectories mapped to the corresponding space, i.e.,

$$\theta_k^{\star} = \arg\min_{\theta_k} \sum_{i=1}^{N} \sum_{t=1}^{T_i} \left\| \mathbf{J}_k \, \dot{\mathbf{q}}_{i,t}^{d} - (\mathbf{M}_{\theta_k})^{-1} \nabla \Phi_{\theta_k} \right\|^2. \tag{5.3}$$

The combination of these individually-learned policy only happens after learning, i.e., during the execution of the policy. This strategy is simple and computational efficient. However, this strategy has the following limitations. First, the learned Riemannian metric, i.e., the importance weight policy, are not learned to provide the proper trade-offs between policies, as each policy is learned independently. Due to this, additional manual scaling of the importance weight matrices is needed especially when combined with hand-designed policies. Second, the geometric constraints (e.g. induced by the kinematics of the robot) between tasks are not considered during learning, which contributions to errors during execution.

In Section 5.3, we proposed an alternative objective function that *jointly* learns all

subtask maps and policies:

$$\begin{aligned}\theta^\star &= \arg\min_\theta \sum_{i=1}^{N}\sum_{t=1}^{T_i}\sum_{k=1}^{K} \lambda_k \left\| \mathbf{J}_k\, \dot{\mathbf{q}}^d_{i,t} - \mathbf{J}_k\, \pi_\theta(\mathbf{q}^d_{i,t}) \right\|^2 \\ &= \arg\min_\theta \sum_{i=1}^{N}\sum_{t=1}^{T_i}\sum_{k=1}^{K} \lambda_k \left\| \dot{\mathbf{q}}^d_{i,t} - \pi_\theta(\mathbf{q}^d_{i,t}) \right\|^2_{\mathbf{J}_k^\top \mathbf{J}_k},\end{aligned} \tag{5.4}$$

where $\lambda_k > 0$ is the user-specified weight for the k-th subtask. In contrast to (5.2), our proposed objective only penalizes the deviation of velocities in the subtask spaces. This approach requires differentiating the simplified RMP^2 algorithm in (5.1). By properly formulating the learning problem (5.4), this approach is able to take full advantage over the policy structure during learning.

As is discussed in Chapter 4, the policy parameterization π_θ also allows for certain subtask policies, e.g., collision avoidance and joint limit RMPs, to be *fixed* during learning. This provides the user with the freedom to manually design these subtask policies.

5.2 Case Study 1: Learning Individual Subtask RMPs

In this section, we present a case study of learning individual subtask policies *independently* and then later combine these learned policies during execution, potentially along with other hand-designed subtask policies. In this chapter, we focus on parameterizing subtask policies as GDSs, as introduced in Section 4.3.1. We choose to approach learning of GDSs in a two-step manner. Specifically, we first learn a potential function from a single nominal demonstration and then learn an Riemannian metric which warps the gradient of the potential such that the subtask policy accurately reproduces the demonstrations.

For notational clarity, we will drop the subtask space subscript k in the remainder of this section. However we will always refer to a particular subtask unless otherwise stated.

5.2.1 Learning Potential Function from a Nominal Demonstration

In order to learn a subtask space potential function, we first select a nominal demonstration (in subtask space) $\boldsymbol{\zeta}$ which is the closest to all other demonstrations in terms of geometric features. One such metric of dissimilarity is Dynamic Time Warping (DTW) distance. The

nominal demonstration is therefore given by the demonstration with the least mean DTW distance from other demonstrations.

Given the nominal trajectory demonstration $\boldsymbol{\zeta}$, we learn a potential $\Phi(\mathbf{x})$ from the nominal trajectory using the parameterization described in Section 4.3.1. Recall that the goal is to learn a potential function with a gradient field $-\nabla\Phi(\mathbf{x})$ that produces motions which: *1)* converge smoothly towards the nominal trajectory, and *2)* follow the remaining trajectory after convergence. The potential gradient is decomposed as,

$$\nabla\Phi(\mathbf{x}) = \nabla\Phi^{\perp}(\mathbf{x}) + \nabla\Phi^{\parallel}(\mathbf{x}) \tag{5.5}$$

where the gradient component $\nabla\Phi^{\perp}(\mathbf{x})$ causes attractive pull towards the demonstration while the component $\nabla\Phi^{\parallel}(\mathbf{x})$ produces accelerations along the direction of motion of the demonstration. The motion along the trajectory is given by $\nabla\Phi^{\parallel}(\mathbf{x})$. We use CVX [Grant et al., 2009] to solve the learning problem.

5.2.2 Learning Riemannian Metric from Multiple Demonstrations

While a single demonstration can imply certain traits of motions, multiple demonstrations can further capture certain properties of the skill that can not be encoded by a single trajectory. For example, if all the demonstrations stay close to each other in a particular region of the state space, it informs that the motion is highly constrained in the region. In such part of the state space, the reproduced trajectories should follow the demonstrations closely so that the skill specifications are satisfied. We choose to learn an Riemannian metric $\mathbf{G}(\mathbf{x};\theta)$ on the subtask space (manifold) to warp the learned potential function. The importance weight policy expands or contracts the space so that the attractive component is no longer uniform along the trajectories.

We follow Section 4.3.1 and parameterize the Riemannian metric through its Cholesky decomposition. We represent the metric matrix as a neural-network with parameter θ. The neural network takes in the coordinate $\mathbf{x} \in \mathbb{R}^n$ and outputs $\mathbf{l}_d(\mathbf{x};\theta)$ and $\mathbf{l}_o(\mathbf{x};\theta)$ (Figure 4.5). To ensure that $\mathbf{l}_d(\mathbf{x};\theta)$ is strictly positive for all $\mathbf{x} \in \mathbb{R}^n$, we take absolute value of the output of the linear layer and add it with a small positive offset $\epsilon > 0$. Hence, lower-triangular matrix $\mathbf{L}(\mathbf{x};\theta)$ is always invertible and the Riemannian metric $\mathbf{G}(\mathbf{x};\theta)$ is

guaranteed to be positive definite. The Riemannian metric learning problem seeks to find the parameters of the neural network such that final natural gradient descent system reproduces the demonstrated subtask space velocities $\{\{\dot{\mathbf{x}}^d_{i,t}\}_{t=1}^{T_i}\}_{i=1}^{N}$ along the subtask space demonstrations $\{\{\mathbf{x}^d_{i,t}\}_{t=1}^{T_i}\}_{i=1}^{N}$,

$$\begin{aligned} \arg\min_{\theta} \quad & \sum_{i=1}^{N}\sum_{t=1}^{T_i} \mathcal{L}\big(-\mathbf{G}(\mathbf{x}^d_{i,t};\theta)^{-1}\nabla\Phi(\mathbf{x}^d_{i,t}),\ \dot{\mathbf{x}}^d_{i,t}\big) \\ \text{s.t.} \quad & \mathbf{G}(\mathbf{x}^d_{i,t};\theta) = \mathbf{L}(\mathbf{x}^d_{i,t};\theta)\,\mathbf{L}(\mathbf{x}^d_{i,t};\theta)^\top, \end{aligned} \tag{5.6}$$

where $\mathcal{L}(\cdot,\cdot)$ can be any regression loss function, e.g. L2 loss, smooth L1 loss, and their regularized version. It is noteworthy that $\mathbf{G}(\cdot;\theta)$ need not be explicitly computed during training since $\mathbf{G}(\mathbf{x};\theta)^{-1} = \mathbf{L}(\mathbf{x};\theta)^{-\top}\mathbf{L}(\mathbf{x};\theta)^{-1}$, and $\mathbf{L}(\mathbf{x};\theta)^{-1}$ can be efficiently computed through forward-substitution due to the fact that $\mathbf{L}(\mathbf{x};\theta)$ is a lower-triangular matrix.

Although the training is done on the first-order system. While executing the learned policy, the curvature term $\boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x},\dot{\mathbf{x}};\theta)$ needs to be computed as well. By definition of the curvature term (3.13), its calculation involves computing the partial derivatives $\frac{\partial G_{ij}(\mathbf{x};\theta)}{\partial x_k}$, which can be computed through back-propagation using automatic differentiation libraries.

5.2.3 Experimental Evaluation

We evaluated the proposed approach on two manipulation tasks, including *door reaching* and *drawer closing*. Both tasks were demonstrated on a 7-DOF Franka Emika robot with configuration space coordinate $\mathbf{q} \in \mathbb{R}^7$. For each skill, a human subject provided 6 demonstrations via kinesthetic teaching, starting from various joint angles of the robot. The demonstrations were recorded in the joint space.

The desired tasks require coordinated motion of the entire hand of the robot with respect to the target objects. To encode the motion of the entire wrist, we pick $k = 3$ control points on the hand; one defined at the center of the gripper and one each at the tips of the two-fingered gripper. We consider three three 3-dimensional subtasks, one per control point, under maps $\{\psi_k\}_{k=1}^{3}$. Specifically, a subtask map is defined as a composition $\psi_k := \texttt{FK}_k(\cdot) - \mathbf{g}_k)$. Here $\texttt{FK}_k : \mathbb{R}^7 \mapsto \mathbb{R}^3$ maps the joint angles to the kth control point position under the robot's forward kinematics, and $\mathbf{g}_k$ is the target location in the k-th subtask

space[2]. For each subtask, we independently learn a subtask policy in the form (3.18).

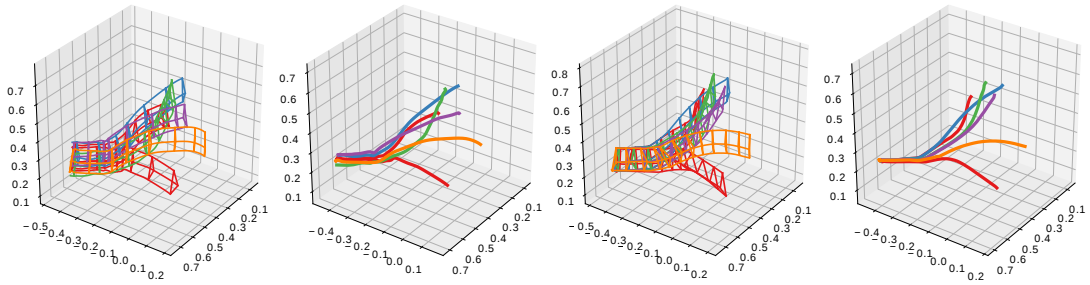
To execute the skill in a new environment, additional hand-specified joint limit RMPs, and obstacle avoidance RMPs are also added as subtask RMPs. These hand-specified RMPs dictate the kinematic and environmental constraints. All subtask policies, learned and hand-specified, are combined via (5.1) to produce a configuration space policy.

The *drawer closing* task required the robot to reach the handle of a drawer from a given initial position and push it closed. Figure 5.1a–5.1b shows the demonstrations transformed in the 3 aforementioned subtask spaces. It should be noted that the closing motion not only requires a straight line motion by the end-effector after making contact with the drawer handle, but also requires the wrist to align with the face of the handle. Figure 5.1c–5.1d shows the reproductions from the same initial positions as the demonstrations while Figure 5.2 shows an instance of reproduction on the real robot. Among all the 6 trials, the robot is able to successfully reproduce the demonstrations and close the drawer.

The *door reaching* task required the robot to start from inside a cabinet, going around the cabinet door and reaching for the door handle outside the door. The robot is required to stop at a standoff position a small distance away from the drawer handle. This task requires a highly constrained motion that results in the end-effector moving along a C-shaped arc. Furthermore, as before, the entire wrist trajectory is important here since the robot is required to reach the handle at a certain relative angle. Figure 5.4a–5.4b shows the demonstrated trajectories while Figure 5.4c–5.4d shows the reproductions from the same set of initial positions. Figure 5.5 shows snapshots of a task reproduction on the real robot. Once again we notice all the reproductions to successfully achieve the task. Furthermore, to test the reactive behavior of our policy, we displace the door during execution. As noticeable in Figure 5.6, the robot successfully react to the the change in goal location and hence complete the task.

Another important feature our approach inherits from RMPflow is obstacle avoidance. We desire the generated motions to be collision-free in order to be feasible in any new environment. To test the obstacle avoidance behavior, we place a cylindrical obstacle in

[2]The target location $\mathbf{g}_k$ is attached to the target object. This enables the policy to react to object displacement.



(a) (b) (c) (d)

Figure 5.1: Trajectories for the *drawer closing* task. (a), (c): The motion of 3 control points on the robot hand, shown as vertices of a triangle, along the demonstrated and reproduced trajectories respectively. (b), (d): The demonstrated and reproduced end-effector (center of hand) trajectories.

Figure 5.2: The *drawer closing* task. The robot successfully closes the drawer from a new initial configuration.

the environment such that it hinders the robot's motion towards the drawer for the *drawer closing* task. We notice successful completion of the task as the robot is able to go around the obstacles in all 6 trials (Figure 5.3).

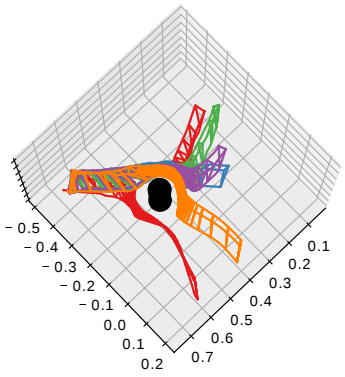
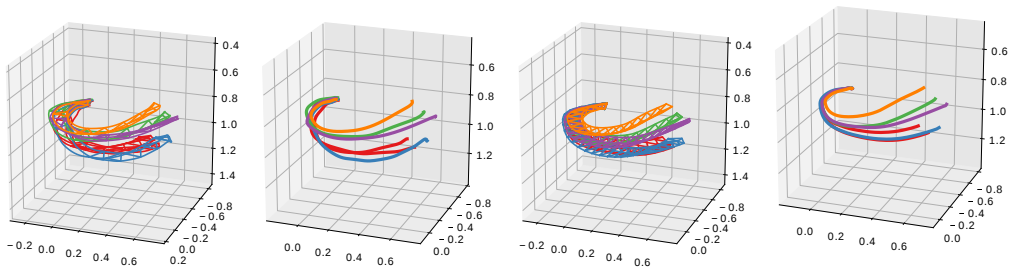


Figure 5.3: Reproductions of the *drawer closing* task with a cylindrical obstacle (in black) in the scene. The positions of the 3 control points on robot hand are shown as vertices of a triangle.



(a) (b) (c) (d)

Figure 5.4: Trajectories for the *cabinet door reaching* task. (a), (c): The motion of 3 control points on the robot hand, shown as vertices of a triangle, along the demonstrated and reproduced trajectories respectively. (b), (d): The demonstrated and reproduced end-effector (center of hand) trajectories.

Figure 5.5: The cabinet *door reaching task*. The robot manages to reach the cabinet door handle despite of the new initial configuration and new door configuration.

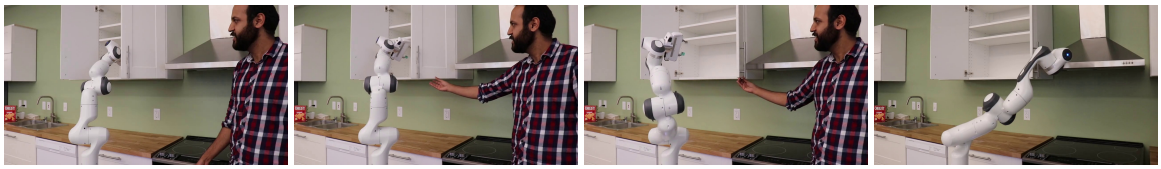


Figure 5.6: Reactivity. The door handle location is displaced during execution and the robot can be seen to adapting to the new target.

5.3 Case Study 2: Jointly Learning Multiple Subtask RMPs

In this section, we provide details of our approach to *jointly* learning multiple subtask policies and maps from human demonstrations. Given the observation that the trajectory demonstrations are the most informative when considered in the subtask spaces, we propose an alternative objective function that direct penalize the deviation in the subtask spaces:

$$\begin{aligned}\theta^\star &= \arg\min_\theta \sum_{i=1}^{N}\sum_{t=1}^{T_i}\sum_{k=1}^{K} \lambda_k \left\| \mathbf{J}_k \, \dot{\mathbf{q}}^d_{i,t} - \mathbf{J}_k \, \pi_\theta(\mathbf{q}^d_{i,t}) \right\|^2 \\ &= \arg\min_\theta \underbrace{\sum_{i=1}^{N}\sum_{t=1}^{T_i}\sum_{k=1}^{K} \lambda_k \left\| \dot{\mathbf{q}}^d_{i,t} - \pi_\theta(\mathbf{q}^d_{i,t}) \right\|^2_{\mathbf{J}_k^\top \mathbf{J}_k}}_{\mathcal{L}(\theta)},\end{aligned} \tag{5.7}$$

where $\lambda_k > 0$ is the user-specified weight for the k-th subtask. In contrast to (5.2), our proposed objective only penalizes the deviation of velocities in the subtask spaces. Let us now consider policy learning with RMP^2 policies. As the subtask spaces are provided by the demonstrator, we can represent the joint space velocity as the solution to (5.1). We can then optimize for the objective function (5.4) through, e.g. gradient descent:

$$\begin{aligned}\theta &\leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta) \\ &\leftarrow \theta - \alpha \sum_{i=1}^{N}\sum_{t=1}^{T_i} \frac{\partial \pi_\theta(\mathbf{q}^d_{i,t})}{\partial \theta} \nabla_{\pi_\theta(\mathbf{q}^d_{i,t})} \mathcal{L}(\theta),\end{aligned} \tag{5.8}$$

where the term $\frac{\partial \pi_\theta(\mathbf{q}^d_{i,t})}{\partial \theta}$ can be computed by back-propagating through the simplified RMP^2 algorithm solving (5.1).

5.3.1 Policy Parameterization

We seek to paramterize subtask maps and policies $\{(\psi_k, \mathbf{M}_k, \mathbf{v}_k)\}_{k=1}^K$ so that the resulting configuration space policy is stable. Similar to the stability results in Chapter 3, the resulting motion is stable as long as the subtask policy follows natural gradient flow. Therefore, we choose to parameterize the subtask map and policy $(\psi_k, \mathbf{M}_k, \mathbf{v}_k)$ through the tuple $(\psi_k, \Phi_k, \mathbf{G}_k)$, where Φ_k is the potential function and $\mathbf{G}_k = \mathbf{M}_k$ is the Riemannian metric. The velocity policy is then given by $\mathbf{v}_k = (\mathbf{G}_k)^{-1}\nabla\Phi_k$. To ensure stability of the combined

policy in the configuration space, we need the Riemannian metric $\mathbf{G}_k$ to be always positive definite. Additionally, we require Φ_k to have a unique minimum at a desired goal location.

The main challenge for representing the subtask policy is finding a expressive parameterization of the potential function without introducing spurious attractive points. While direct parameterization with such property is in general challenging, in Chapter 4 we show that we can parameterize diffeomorphisms to *latent* subtask spaces, where a simple potential function is defined there. We adopt this approach as it shows expressiveness in representing complex motions without introducing undesired local minima. In particular, we use $\psi_k = (\varphi_k - \varphi_k(\mathbf{0})) \circ (\texttt{FK}_k(\cdot) - \mathbf{g}_k)$, where φ_k is given by the diffeomorphic mapping network in Fig. 4.3.

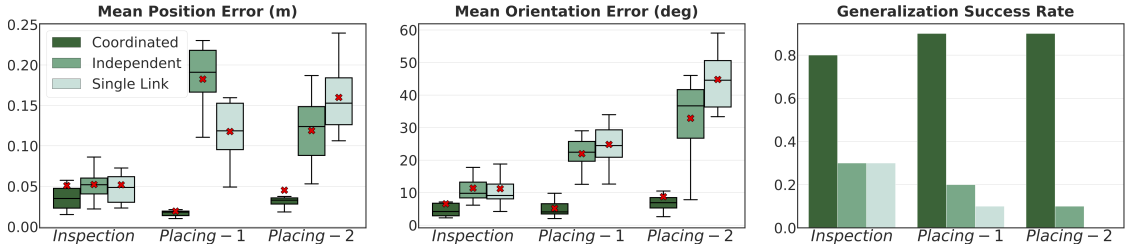
Then, in the latent space, we can use a simple pre-specified potential function, e.g. $\Phi_k(\mathbf{x}_k) = 0.5\|\mathbf{x}_k\|^2$. Similar to Section 5.2, we represent a latent subtask Riemannian metric $\mathbf{G}_k$ by its Cholesky decomposition parameterized by a matrix-valued neural network. Then, by properties of diffeomorphisms (see Chapter 4), the generated motion is guaranteed stable.

5.3.2 Experimental Results

We evaluated our approach on three manipulations tasks[3] on a 7-DOF Rethink Sawyer robot with configuration space coordinates $\mathbf{q} \in \mathbb{R}^7$. We consider 3 tasks including *inspection*, *placing-1*, and *placing-2*. For details about the task specifications, the reader is referred to Figs. 5.9–5.11. For each task, a human subject provided multiple configuration space demonstrations via kinesthetic teaching: 14 demonstrations for *inspection*, 9 for *placing-1*, and 12 for *placing-2*.

Subtasks. Each of the tasks is decomposed into learnable subtasks assigned to 3 robot body parts, whereby each body part is represented by a unique *control point* (see Fig. 5.8 for details). Given our choice of control points, the subtask policies effectively control the end-effector pose (i.e. position and orientation). However, we stress that our learning approach is not only limited to learning policies for robot poses. In fact, one may instead, for instance,

[3]Accompanying video is available at: `https://youtu.be/hwcxzLnxZPQ`.



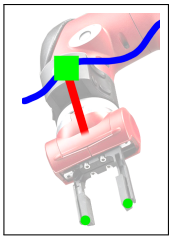
(a) (b) (c)

Figure 5.7: Comparison of our approach against baselines based on (a) mean position error, (b) mean orientation error, and (c) generalization success rate over 10 executions.

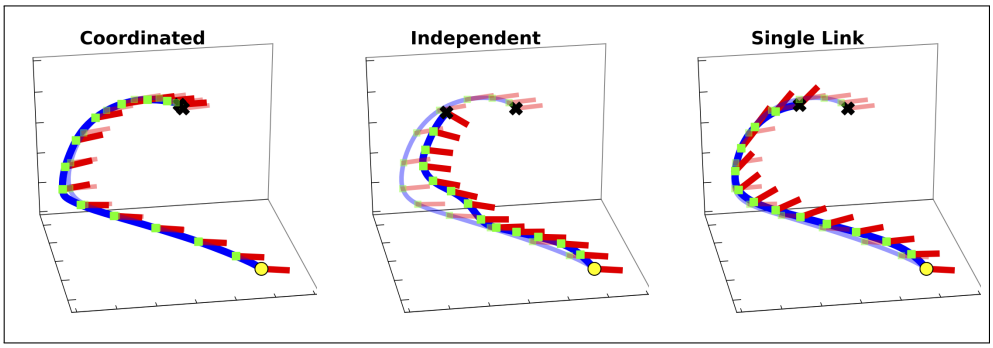
choose to learn motion policies dictating a partial pose (by removing a control point), or pose alongside the robot elbow (by adding an additional control point). Furthermore, there is a hand-specified default subtask policy pulling the end-effector in straight-line towards a desired goal pose. It is governed by a convex potential and a constant inertia matrix of $10\,\mathbb{I}_{7\times 7}$. Additionally, to ensure the configuration space importance weight matrix $\mathbf{M}$ is always non-singular and well-conditioned, we add a small offset $\epsilon = 0.02$ to its diagonal entries.

Baselines. To evaluate the performance of our approach, we establish two baselines: *1)* an *independent* learning version whereby the subtask policies are learned independently, as described in Section 5.2, and *2)* a *single link* learning version where just a single control point (i.e. end-effector) is chosen and the associated subtask policy is again learned independently.

Learning Details The subtask policies are defined by a set of diffeomorphisms $\{\varphi_k\}_{k=1}^{3}$ and a set of latent Riemannian metrics $\{\mathbf{G}_k\}_{k=1}^{3}$. In our parameterization, each diffeomorphism is composed of 10 chained diffeomorphisms, each parameterized by 200 random Fourier features with length-scale of 0.45. On the other hand, each latent Riemannian metric $\mathbf{G}_k$ is parameterized by neural network with two hidden layers with 128 and 64 dimensions respectively. The optimization problem in (5.4) is solved with Adam optimizer Kingma and



(a)



(b)

Figure 5.8: (a) Visualization of the 3 control points (in green), with the end-effector control point denoted by a square while the two control points for gripper tips are given by dots. Overlaid is an end-effector position trajectory (in blue), and a line directed from the end-effector to the center of the gripper (in red) denoting instantaneous end-effector orientation. (b) Plots showing pose trajectories starting from an initial end-effector pose (yellow circle) governed by our approach (Coordinated) and baselines (Independent and Single Link). Also shown in the background, is the demonstration starting from the same initial pose. The final positions are denoted by black crosses.

Ba [2014] with a learning rate of 1×10^{-4} and weight decay 1×10^{-8}.

Results Fig. 5.8 shows example reproductions of end-effector pose trajectories under the aforementioned variants of our algorithm. Our coordinated learning approach is observed to successfully reproduce the demonstrated motions. However, the baselines either fail to reproduce the position profile or the orientations. To quantitatively evaluate the capacity of our approach to reproduce demonstrations, we employ two error metrics, mean position error and mean orientation error. We evaluate position errors in terms of the Euclidean distance i.e. $\texttt{error}(\boldsymbol{p}_1(t), \boldsymbol{p_2}(t)) = \|\boldsymbol{p}_1(t) - \boldsymbol{p}_2(t)\|_2$, where $\mathbf{p}_1(t)$ and $\mathbf{p}_2(t)$ are end-effector positions on the demonstrated and reproduced trajectory at time stamp t, respectively. On the other hand, for orientation errors we employ $\texttt{error}(\boldsymbol{o}_1(t), \boldsymbol{o}_2(t)) = \arccos(|\boldsymbol{o}_1(t) \cdot \boldsymbol{o}_2(t)|)$, where $\boldsymbol{o_1}(t)$ and $\boldsymbol{o}_2(t)$ are unit quaternions representing end-effector orientations. For each comparison metric, we take the mean of the errors accumulated over the duration of a trajectory. Fig. 5.7 reports these comparisons as box plots. For the two *placing* tasks, our approach outperforms the baselines by a significant margin. A major contributor towards this difference in performance is the existence of default subtask policies. When learned without accounting for the existing policies, the learned policies may not be able to sufficiently bias against the default behavior. Furthermore, we observe that the independent learning version occasionally performs worse than the single link variant. This is perhaps because the independently learned subtask policies may conflict with each other. This does not manifest as much in the single link case, since there is only one learned subtask policy.

Lastly, we test the generalization performance of our approach. For this evaluation, we roll out our motion policy from 10 new initial configurations. A rollout is considered successful if all the goals of the task are met without any collisions. Fig. 5.7 (c) reports the success rates. Once again, our end-to-end learning approach outperforms the baselines. We also observe that, while the difference in terms of quantitative errors between our approach and the baselines is small on the *inspection* task, there are vast differences in performances given by generalization success rates. This is perhaps because, even when not trained end-to-end, the robot's kinematic constraints may enforce certain level of coordination between subtasks, thus resulting in low reproduction errors. However, for highly constrained tasks

like the ones we explore in this section, even small errors can result in task execution failures. A subset of rollouts from our learned policies, starting from the same configurations as demonstrations, are visualized in Figs. 5.9–5.11 (*bottom*).

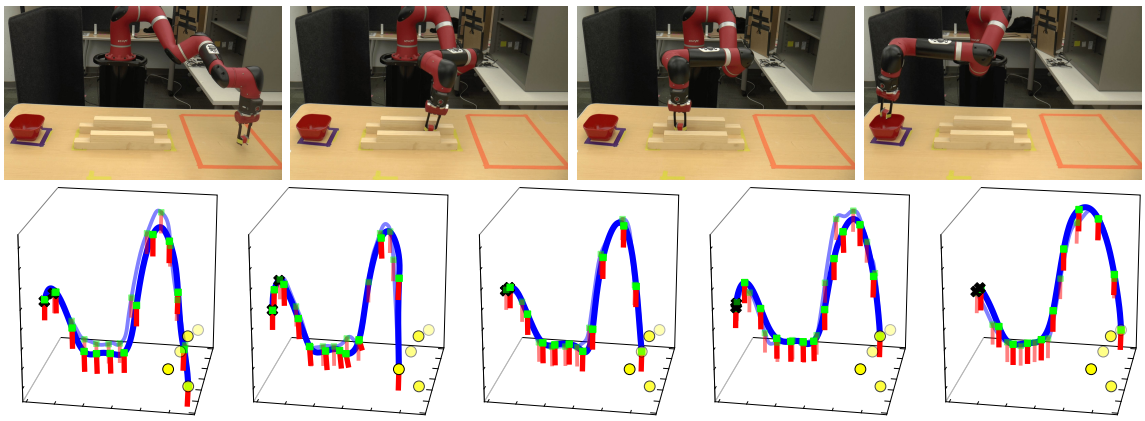


Figure 5.9: The *inspection* task required the robot to pick an object from one side of the table and place it in a bowl on the other side. In the middle, the robot was required to pass a constrained pathway. *Top:* A series of snapshots showing a robot executing learned behavior. *Bottom:* Plots of a subset of motion reproductions from different initial poses, overlaid on corresponding demonstrations. The yellow circles represent the initial end-effector positions, each corresponding to one of the rollouts.

5.4 Conclusion

In this chapter, we use the parameterizations introduced in Chapter 4 to learn RMP^2 policies from human demonstrations. We introduce two strategies for learning with multiple subtasks of interest, *1) individually* learn subtask policies and *2) jointly* learn subtask policies and maps. The former is more simple and computational efficient while the later often tends to produce better policies. We show that with RMP^2 policies, we can learn with a small set (around 10) demonstrations. The learned policies generalize well to unseen initial configurations and obstacles, and are robust to disturbances.

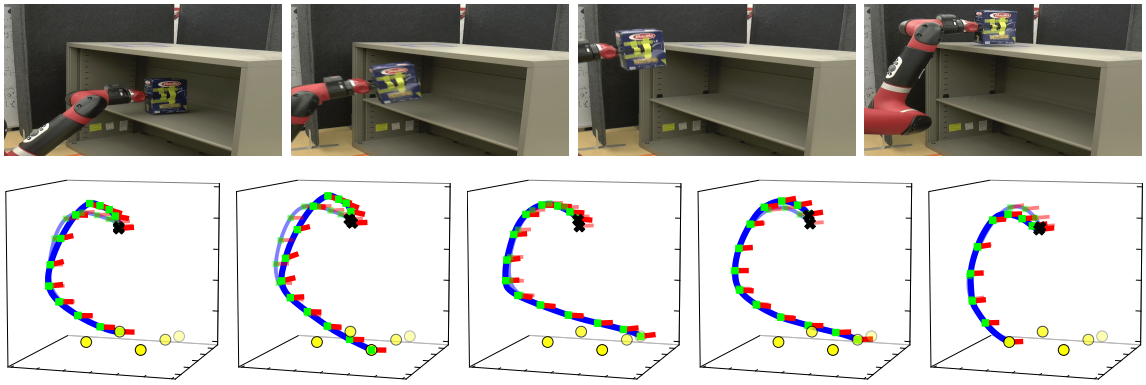


Figure 5.10: The *placing-1* task required the robot pick an object from a lower shelf and place it on at a goal location on the top-most shelf at a certain orientation. *Top:* A series of snapshots showing a robot executing learned behavior. *Bottom:* Plots of a subset of motion reproductions from different initial poses, overlaid on corresponding demonstrations. The yellow circles represent the initial end-effector positions, each corresponding to one of the rollouts.

5.5 Related Work

5.5.1 Learning Stable Policies from Human Demonstrations

Over the past decade, a number of approaches have been proposed towards learning goal-directed time-invariant dynamical systems from human demonstrations [Osa et al., 2018]. An early approach is SEDS [Khansari-Zadeh and Billard, 2011]. SEDS assumes the demonstrations comply with a Lyapunov (or energy) function given by the squared-distance to the goal, and thus is restricted to motions which monotonically converge to the goal over time. A relaxation to the stability criterion was proposed in CLF-DM [Khansari-Zadeh and Billard, 2014], which instead assumes Lyapunov functions in the form of a weighted sum of asymmetric quadratic functions (WSAQF). The parameters of the Lyapunov function are learned independently from the (unstable) dynamics, and then used in an online fashion to generate stabilizing controls. The online correction scheme may interfere significantly with the learned dynamics [Neumann and Steil, 2015]. A more recent approach, CDSP [chaan-

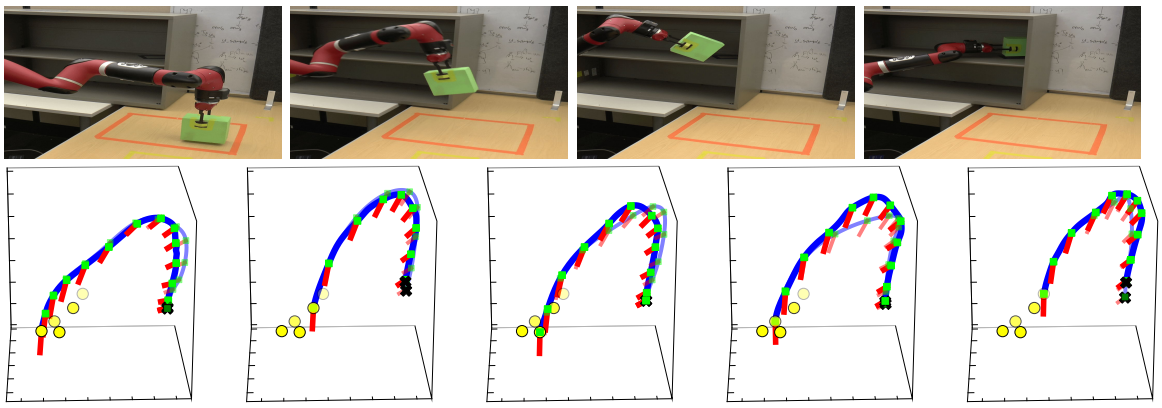


Figure 5.11: The *placing-2* task required the robot pick an object from a table, significantly rotate its end-effector, and place the object on a shelf. *Top:* A series of snapshots showing a robot executing learned behavior. *Bottom:* Plots of a subset of motion reproductions from different initial poses, overlaid on corresponding demonstrations. The yellow circles represent the initial end-effector positions, each corresponding to one of the rollouts. Note that the viewing angle in the plots is different from that in the robot execution snapshots.

dar Ravichandar and Dani, 2019] instead enforces incremental stability, a notion concerned with relative displacement of motions. Instead of learning a Lyapunov function, CDSP proposed learning a positive-definite contraction metric. However, CDSP restricts the class of contraction metrics to (potentially limiting) sum of squared polynomials.

There are a few approaches that learn stable dynamical systems via learned diffeomorphisms. However, by definition, a diffeomorphism must be invertible, making the learning problem non-trivial. One approach [Perrin and Schlehuber-Caissier, 2016], realizes a diffeomorphism as a composition of locally weighted translations. The authors only apply this approach to the problem of learning a single (or average) demonstration. Another strategy, τ-SEDS [Neumann and Steil, 2015], learns diffeomorphisms from multiple demonstrations. However, τ-SEDS defines a diffeomorphism by the square-root of a WSAQF, thus restricting the hypothesis class.

5.5.2 Imitation Learning for Robotics Problems

Behavior Cloning [Pomerleau, 1988] contended with the covariate shift issue manually by using data augmentation. One way to address this issue is through Inverse Reinforcement Learning (IRL) [Ng and Russell, 2000, Abbeel and Ng, 2004a] and related entropy maximization technique [Ziebart et al., 2008a]. Those algorithms required planners in the inner loop which added both computational complexity and required the existence of such globally optimal planners. There has been work [Levine and Koltun, 2012, Englert et al., 2017] into leveraging KKT conditions and optimization theory, but all of these methods focus on exploiting the structure of the MDP rather than directly training a policy.

A separate thread of research introduced SMiLE [Ross and Bagnell, 2010] and DAgger [Ross et al., 2011], with later extensions [Ross and Bagnell, 2014, Sun et al., 2017]. These algorithms focused on the same policy learning paradigm of BC, but directly addressed the covariate shift issue both theoretically and pragmatically using a technique called data set aggregation. These algorithms assumed access to an oracle expert that can be queried, which is an information theoretically stronger assumption than the standard BC algorithms which leverage only data collected in advance.

Venkatraman et al. [2014] observed the infeasibility of this assumption for some settings, focusing in particular on learning dynamical systems, and devised a method of using the data itself as an oracle expert. This method, termed Data As Demonstrator (DAD), alleviates the covariate shift issue by generating synthetic data to ensure that prediction returns to demonstrated states. However, the technique may generate data for two separate demonstrations that conflict with one another, an issue the authors acknowledged.

Adversarial Imitation Learning (AIL) such as GAIL [Ho and Ermon, 2016a] and AIRL [Fu et al., 2017], which frames imitation learning as learning a policy that minimizes a divergence between the state-action distributions of the expert trained policy, has also recently become a popular approach for imitation learning. Unfortunately, these methods are sample inefficient since a large number of policy interactions with the environment are required and inherit training stability challenges characteristic of adversarial learning making them difficult to apply in practice to robot learning.

Chapter 6

REINFORCEMENT LEARNING WITH RMP^2 POLICIES

We can view the acceleration-based control problem discussed in Chapter 3 as a (deterministic) Markov decision process (MDP): the state is the position-velocity $s \coloneqq (\mathbf{q}, \dot{\mathbf{q}})$, the action is the acceleration $a \coloneqq \ddot{\mathbf{q}}$, the transition is given by the integration rule[1], and the reward is defined to encourage desired behaviors for a task (such as smooth, goal-reaching and collision-free motions). Our goal is to find an acceleration policy π such that the system following $\pi(a|s) = \pi(\ddot{\mathbf{q}}|\mathbf{q}, \dot{\mathbf{q}})$ would exhibit good performance for the tasks of interest. This means that with a well-defined reward, we can learn RMP^2 policies through (online) *reinforcement learning*.

RL is a generic learning formulation, which does not make explicit assumptions on the dynamics and policy classes[2]. Therefore, it is possible to directly apply off-the-shelf RL algorithms to learn RMP^2 policies. However, RL algorithms are known to be sensitive to design choices, e.g., hyper-parameters, reward functions and exploration strategies. There is also flexibility on formulating the RL problem, as the boundary between an agent and the environment can be drawn in different ways [Sutton and Barto, 2018]. RL is also known to be sample inefficient. In this chapter, we discuss a few considerations on reinforcement learning with RMP^2 policies, and present a few successful examples of RL with RMP^2 policies. This chapter is adapted from [Li et al., 2021].

---

[1]The acceleration-based control system is a continuous-time system. However, in practice, the control (action) is only given at discrete time steps. See Remark 2.2.1 for a discussion on continuous-time versus discrete-time systems.

[2]Certain approaches, e.g., policy gradient approaches and actor-critic approaches assume that the policy class is differentiable, which is the case for RMP^2 policies.

6.1 Considerations on Reinforcement Learning with RMP^2 Policies

6.1.1 Boundary between the Policy and the Environment

When formulating an RL problem, the boundary between the agent, i.e., the policy, and the environment can be drawn in multiple ways [Sutton and Barto, 2018]. Practically, different choices of agent-environment boundary can cause drastic differences in sample efficiency, exploration quality, computational speed, etc.

Perhaps the only requirement is that all learnable parameters should be included as part of the policy (instead of the environment). This means that, when parameterizing RMP^2 policies with subtask RMPs (when not parameterizing subtask maps), we have the flexibility as either consider the RMP^2 algorithm as part of the policy or the environment.

When choosing RMP^2 as part of the policy, one would likely need to differentiate through the algorithm, which is possible since RMP^2 is implemented in automatic differentiation libraries. This means that when updating the policy through, e.g., policy gradient, one need to back-propagate through the computational graph constructed by RMP^2, which could be time consuming[3] with a large number of updates. However, the action dimensional is fixed to be d, the dimension of the configuration space, which is relatively small. This means that it could be more sample efficient if we consider RMP^2 as part of the policy.

On the other hand, if we consider RMP^2 as part of the environment, we do not need to back-propagate through RMP^2, which makes each policy update faster. In this case, the action space will be the space of all learnable subtask RMPs (importance weight and accelerations). When learning a large number of subtask RMPs, this may result in a large action space, which makes exploration harder and requires more samples during learning.

Therefore, if a large number of subtask RMPs is parameterized, it could be beneficial to consider RMP^2 as part of the policy and differentiate through it, because otherwise it will result in a high-dimensional action space for RL. On the other hand, when there is only a single low-dimensional subtask RMP to be learned, it might be convenient to consider RMP^2 as part of the environment so that policy update is faster.

---

[3]The time for differentiating through RMP^2 is a constant factor more than the time required for computing RMP^2, which is still reasonably fast.

6.1.2 Accelerate Learning with Residual Policies

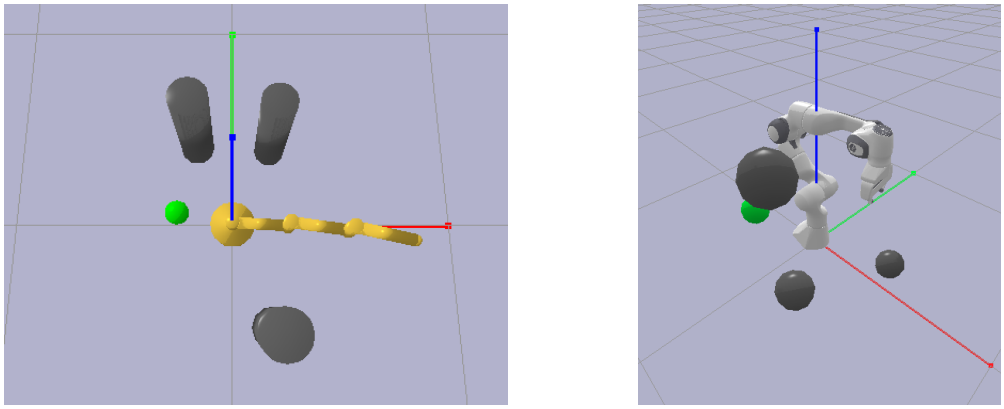
For many robotics tasks, hand-crafted RMPs [Cheng et al., 2018] can provide a possibly sub-optimal but informative prior solution to the task or some subtasks (e.g. avoiding collision, respecting joint limits, etc.). In many cases, making use of these hand-crafted RMPs within RMP^2 policies can benefit learning, as it could provide a reasonable initialization for the learner and, for RL, an initial state visitation distribution with higher rewards.

Learning Residual Acceleration Policy: Perhaps the most straightforward solution is to learn the residual (configuration space) acceleration policy to a (hand-specified) RMP^2 policy using universal functional approximators, e.g. neural networks [Johannink et al., 2019]. As we show in the experiments, this can often provide a significant improvement to the performance compared to randomly initialized policies, especially when the tasks are more challenging.

Learning Residual RMP: Another option is to learn a residual subtask RMP with a universal functional approximator (i.e. the subtask RMP is initialized as the hand-crafted RMP). In this way, the structure of the RMP^2 policy is preserved. As is shown in the experiments, learning with residual RMP can perform significantly better than using a randomly initialized neural network policies, and can sometimes learn faster than learning residual acceleration policy.

6.2 Three-link Robot Reaching

We first consider a three-link robot simulated by the PyBullet physics engine [Coumans and Bai, 2017], which we saw in Chapter 3. The robot has 3 links, each of length 0.25 m. The workspace of the robot is a 2-dimensional disk of radius 0.75 m. The z-coordinates of all links are different so that the links cannot collide into one another. The objective here is for the robot to move the tip of the last link (i.e., the end-effector of the robot) to a randomly generated goal location while avoiding randomly generated cylinders, as shown in Fig. 6.1 (a). Acceleration-based control is realized through a low-level PD tracking controller, where



(a) (b)

Figure 6.1: PyBullet simulation for the three-link robot reaching task (a) and the Franka robot reaching task (b).

the acceleration motion policy π generates the desired reference state for the PD controller. The robot does not have joint limits but have joint velocity limit of 1.0 rad/s for all joints.

6.2.1 Environment Setups

The robot is initialized at a random configuration within a small range (± 0.1rad) around the zero-configuration (all links pointing right) and at a random low joint velocity within $[-0.005, 0.005]$ rad/s.

The (2-dimensional) center of the base of each obstacle is sampled from an annulus centered at the origin with outer radius 0.9 m and inner radius 0.4 m, and the radius is sampled uniformly from $[0.05, 0.1]$ m. The height of the cylinder is fixed at 0.5 m, which would result in collision if the x, y-coordinates of any points on the robot intersect with the cylinder. The consideration behind this obstacle configuration is that their intersection with the workspace is usually non-zero, and they would not result in a configuration where the goal is not achievable. The initial configurations of the environment also ensures a minimum of 0.1 m between the goal and any obstacles as well as that between the initial configuration of the robot and any obstacles (otherwise, the initialization is rejected, and

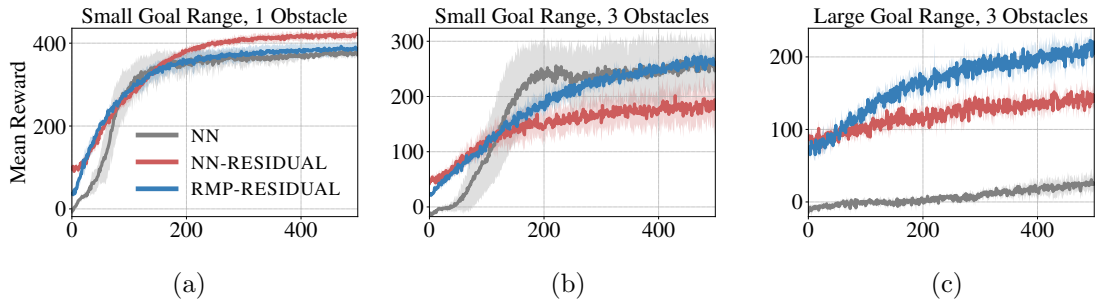


Figure 6.2: Mean episode reward over iterations for the three-link robot reaching task.

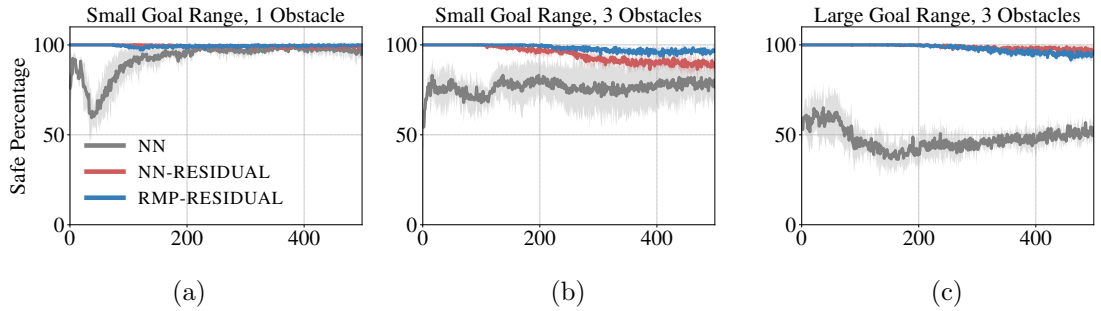


Figure 6.3: Percentage of safe episodes over iterations for the three-link robot reaching task.

the goal and obstacle(s) are re-sampled).

We consider three environment setups for the three-link robot with increasing difficulty:

- `Env 1` (small goal range; 1 obstacle): the goal is sampled from the intersection of an octants (sector with central angle $\pi/2$) at the origin and an annulus with outer radius 0.275 m and inner radius 0.475 m. One obstacle is sampled from the procedure described above;

- `Env 2` (small goal range; 3 obstacles): the goal is sampled from the same region as Env 1. Three obstacles are sampled through the same procedure;

- `Env 3` (large goal range; 3 obstacles): the goal is sampled from the intersection of the

left half-disk and an annulus with outer radius 0.125 m and inner radius 0.625 m.

`Env 2` is more difficult than `Env 1` as there are more obstacles. `Env 3` is the most complicated scenario as it has a larger range of random goals, which is known to be challenging for RL algorithms [Aumjaud et al., 2020]. Moreover, although the goals here are generally closer than the previous 2 environment setups, this also increase the frequency of the scenarios where the obstacles are directly obstructing the way to the goal, requiring a more sophisticated policy.

Inspired by Kumar et al. [2020b], we define the reward function as, $r = \exp\left(\frac{\|\mathbf{x}-\mathbf{g}\|_2^2}{2\sigma^2}\right) + \sum_{i=1}^{N} \max\left(0, 1 - \frac{d_i}{\delta}\right) - \lambda\|\tau\|^2$, where $\mathbf{x}$ is the position of the end-effector of the robot, $\mathbf{g}$ is the goal position, d_i is the distance between the robot and the ith obstacle, and τ is the torque applied to the robot by the low-level PD controller. The scalars σ, δ, and λ are the characteristic length scale of the goal reward, characteristic length scale of the obstacle cost, and the multiplier for the actuation cost. We choose $\sigma = 0.1$, $\delta = 0.05$, and $\lambda = 1 \times 10^{-5}$. Further, we clip the reward if it is smaller than -5 so that the reward is in the range of $[-5, 1]$ for each step.

The horizon of each episode is 600 steps, giving the episode reward a range of $[-3000, 600]$. For each step, the acceleration command is applied to the low-level PD-controller, and the simulation proceeds for 0.0125s (simulation time). Therefore, each episode is 7.5s (simulation time) of policy rollout. The episode can end early if the robot collides with an obstacle.

6.2.2 Policy classes

We compare the results for learning with the following 3 types of policies, all implemented in TensorFlow [Abadi et al., 2015].

NN: A randomly initialized 3-layer neural network policy with ReLU activation function and hidden layers of sizes 256 and 128, respectively. The input to the neural network policy consists of $[\sin(\mathbf{q}), \cos(\mathbf{q}), \dot{\mathbf{q}}, \mathbf{g} - \mathbf{x}, \{\mathbf{v}_i\}_i, \{\mathbf{o}_i\}_i]$, where $\mathbf{v}_i$ is the vector pointing from the i-th obstacle and its closest point on the robot, and $\mathbf{o}_i$ denotes the center position and radius of the i-th obstacle. The output of the policy is the 3-dimensional joint acceleration.

NN-RESIDUAL: A residual neural network policy to a hand-designed RMP^2 policy. The input and neural network architecture here are the same as the randomly-initialized case above. However, the joint acceleration now is the sum of the output of the residual policy and the hand-designed RMP^2 policy. The hand-designed RMP^2 policy consists of a joint damping RMP, a joint velocity limit RMP, collision avoidance RMPs, and a target reaching RMP [Cheng et al., 2018].

RMP-RESIDUAL: A residual RMP policy on the 2-dimensional end-effector space, which is the residual to a hand-designed goal attractor RMP (same as the one used for NN-RESIDUAL). The residual RMP policy is parameterized as a 3-layer neural network with hidden layer sizes 128 and 64. We use elu activation function for the neural network. The input to the residual RMP policy consists of $[\mathbf{x}, \dot{\mathbf{x}}, \mathbf{g}, \{\mathbf{o}_i\}_i]$, which are the end effector position, velocity, and the information of goal and obstacle(s). The output of the neural network is (the concatenation of) a matrix $\mathbf{A}_r \in \mathbb{R}^{2\times 2}$ and a residual acceleration vector $\mathbf{a}_r \in \mathbb{R}^2$. Let $(\mathbf{M}_a, \mathbf{a}_a)$ be the output for the hand-designed attractor, the output of the residual RMP policy is then, $\mathbf{M} = (\mathbf{A}_r + \texttt{chol}(\mathbf{M}_a))(\mathbf{A}_r + \texttt{chol}(\mathbf{M}_a))^\top$ and $\mathbf{a} = \mathbf{a}_r + \mathbf{a}_a$, where $\texttt{chol}(\cdot)$ is the lower-triangular Cholesky decomposition of the matrix. This parameterization ensures that the importance weight $\mathbf{M}$ is always positive-semidefinite, and the output is close to the hand-designed RMP when the neural network is initialized with weights close to zero. The output of the residual RMP policy, is combined with other hand-designed RMPs (the same set of RMPs with NN-RESIDUAL) with RMP^2 to produce the joint acceleration. Since we are learning a low-dimensional RMP, as is discussed in Section 6.1.1, it is more convenient to consider RMP^2 as part of the environment so that the policy update is faster.

6.2.3 Training Details

We train the three policies under the three environment setups by proximal policy optimization (PPO) [Schulman et al., 2017], implemented by RLlib [Liang et al., 2018], for 500 training iterations. For each iteration, we collect a batch of 67312 interactions with the environment (112 episodes if there is not collision). We use a learning rate of 5×10^{-5}, PPO clip parameter of 0.2. To compute the policy gradient, we use generalized advantage

estimation (GAE) [Schulman et al., 2015] with $\lambda = 0.99$. The value function is parameterized by a neural network with 2 hidden layers (of sizes 256 and 128, respectively) and tanh activation function.

6.2.4 Experiment Results

The mean episode reward and percentage of safe episodes over training iterations for the three environment setups are shown in Fig. 6.2 and Fig. 6.3, respectively. The curve and the shaded region show the average, and the region within 1 standard deviation from the mean over 4 random seeds.

For `env 1` with small goal range and 1 obstacle, all three policies manage to achieve high reward of around 400 (meaning that, on average, the robot stays very close to the goal for at least 400 steps out of the 600 steps) with a similar number of iterations. The random-initialized neural network (NN) conducts a large number of unsafe exploration, especially during the first 200 iterations, as shown in Fig. 6.3 (a). The other two policies manage to learn a good policy without many collisions as the red and blue curves stay close to 100% throughout training.

For `env 2` with 3 obstacles, the randomly-initialized neural network policy (NN) learns slightly faster than the other two polices, though it has a higher variance (shown by the large gray shaded region in Fig. 6.2 (b)). Notably, although the reward seems high, the resulting policies are not safe, as shown in Fig. 6.3 (b). This reflects the difficulty of reward design: as high reward policies do not necessarily have desirable behaviors. On the contrary, the residual RMP policy (RMP-RESIDUAL) achieves similar reward but is able to keep the number of collisions low. The residual neural network policy (NN-RESIDUAL) improves rather slowly, possibly due to the lack of knowledge about the internal structure of RMP^2.

For `env 3` with large goal range and 3 obstacles, the neural netowrk policy (NN) struggles to achieve reasonable performance under 500 training iterations, and the collision rate remains high. The performance improvement for the reisual neural network policy (NN-RESIDUAL) is also slow, with a slope similar to the neural network policy, though it starts with a higher rewards. The residual RMP policy manages to improve the performance by

more than one-fold.

Remark 6.2.1. One may notice that the initial reward for the residual policies is different for each experiment setup. For example, the initial rewards for `env 3` is significantly higher than `env 2`. This is because we used the *same* hand-designed RMP policy for all three setups, and in `env 3`, the goals are generally initialized closer to the robot than the other two steps (as discussed, this does not make `env 3` easier however).

6.3 Franka Robot Reaching

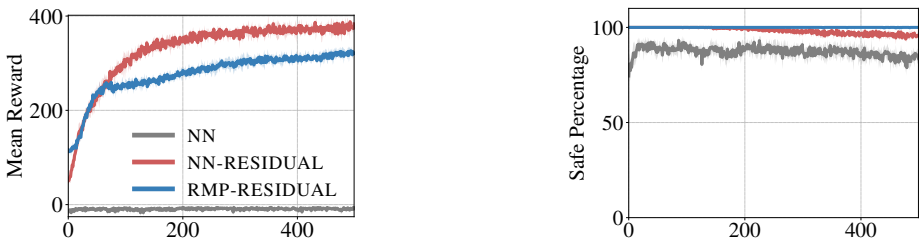


Figure 6.4: Mean episode reward (a) and safe episode percentage (b) over training iterations for the Franka reaching task.

We additionally train the three aforementioned policies on a simulated 7-degree-of-freedom Franka manipulator on PyBullet. The environment setup is similar to the three-link robot reaching task, although the initial configuration is a centered position, as shown in Fig. 6.1 (b). We randomly sample 3 ball obstacles, where the center is sampled a half-torus of major radius 0.5 m, minor radius 0.3 m, and height 0.5 m; and the radius is uniformly sampled from $[0.05, 0.1]$ m. The goal is sampled from the same half-torus, but the distance between the goal and initial end-effector position needs to be at least 0.5 m. The Franka reaching task is more challenging than the three-link robot reaching task as the states are of higher-dimension; however, it is easier in the sense that it has higher degrees of freedom to avoid collision with obstacles.

The mean episode reward and safe percentage of the three polices for the Franka reaching task is shown in Fig. 6.4. Again, the neural network policy (NN) struggles to learn a good policy and avoid collision with obstacles. The residual neural network policy (NN-RESIDUAL) achieves slightly higher reward than the residual RMP policy (RMP-RESIDUAL), possibly because it has higher degree-of-freedoms to control the robot and the learning of the residual RMP policy (RMP-RESIDUAL) has not fully converged. Notably, the residual RMP policy (RMP-RESIDUAL) maintain zero collisions throughout the training process.

6.4 Conclusion

In this chapter, we explore learning RMP^2 policies using online reinforcement learning. We show that there are multiple ways of encoding structure into RMP^2 policies, including using hand-designed subtask policies, e.g., collision avoidance and joint limit policies, using hand-designed subtask maps for goal reaching, and learning residual policies to hand-designed subtask policies. Through encoding problem structure, RMP^2 policies can succeed in tasks that have proved challenging for general function approximator policies, result in significantly fewer safety violations, and are less sensitive to reward designs.

6.5 Related Work

In contrast to hand-designed control techniques, reinforcement learning approaches [Schulman et al., 2015, 2017, Haarnoja et al., 2018, Mnih et al., 2015] make minimal assumptions and promise to improve policy performance through interactions with the environment. However, such approaches often require many interactions to achieve reasonable results, especially when learning policies under sparse reward signals through reinforcement learning. Furthermore, most learning algorithms are sensitive to distribution shifts and have poor out-of-distribution generalization ability. For example, a policy that is trained to go to the left of a room in training often does not know how to go to the right, because such examples are never presented in training.

In practice, we desire motion policy optimization algorithms that possess *both* the non-statistical guarantees from the control-based approaches and the flexibility of the learning-

based approaches. A promising direction is the use of structured policies [Choi et al., 2020, Chow et al., 2019, Dalal et al., 2018, Urain et al., 2021]. Here the main idea is to apply learning to optimize only within policy parameterizations that have provable control-theoretic properties (such as stability and safety guarantees). In other words, it uses data to optimize the hyperparameters of a class of controllers with provable guarantees. Our work is closely related to this line of work. We use RMP^2 policies as the structured policy class, which provides inherent stability and safety guarantees, while being able to encode other useful structure, such as kinematics, task decompositions, etc. in the robotics problems.

There is another line of work on constrained reinforcement learning [Achiam et al., 2017, Stooke et al., 2020] which seeks to provide safety guarantees to the learned policy through solving a Lagrangian relaxation of the constrained optimization problem. These approaches often only provide guarantees for policies *after* learning. The agent can incur high costs *during* learning, which is undesirable for robotics applications. Turchetta et al. [2020] consider a scenario where there exists an intervention oracle, allowing them to provide safety guarantee throughout learning. However, the construction of such an intervention oracle can be highly problem dependent.

Part II

LEVERAGING STRUCTURE IN ROBOTICS DATA

Chapter 7

LEVERAGING EXPLICIT STRUCTURE IN ROBOTICS DATA

In this chapter and the following one, we will focus on an offline learning paradigm. We are given a set of offline robotics data, and the goal is to produce a policy with good performance *without* any online interaction with the environment.

In practice, offline robotics data is oftentimes not homogeneous. Robotics data may come from different sources, and hence may provide different information about the underlying task. We may have multi-task data, where all data provides useful dynamics information, but a subset of the data does not contain reward information about the underlying task. On the other hand, we can have multi-embodiment data, where dynamics information is only meaningful in a subset of the data. Finally, a subset of data can have both dynamics and reward information. In this chapter, we will be reasoning about this structure in robotics data that is given by *what information each subset of data provides, whether it is dynamics or reward*.

To exploit this structure of robotics data, we propose a new paradigm for offline sequential decision making, called offline policy learning from observations (PLfO). Offline PLfO aims to learn policies using datasets with substandard qualities: *1)* only a subset of trajectories is labeled with rewards, *2)* labeled trajectories may not contain actions, *3)* labeled trajectories may not be of high quality, and *4)* the data may not have full coverage. Such imperfection is common in real-world learning scenarios, and offline PLfO encompasses many existing offline learning setups, including offline imitation learning (IL), offline IL from observations (ILfO), and offline reinforcement learning (RL). In this work, we present a generic approach to offline PLfO, called Modality-agnostic Adversarial Hypothesis Adaptation for Learning from Observations (MAHALO). Built upon the pessimism concept in offline RL, MAHALO optimizes the policy using a performance lower bound that accounts for uncertainty due to the dataset's insufficient coverage. We implement this idea by adversarially

Table 7.1: Different problem formulations for sequential decision making on offline datasets. Our PLfO formulation is the most general and can leverage the broadest range of data, which makes it the most realistic. The other formulations can be reduced to PLfO with additional restrictions on data. * denotes that data can only be used partially, with either action or reward removed.

	(s, a, r, s')	(s, a, s')	Expert (s, a, s')	Expert (s, s')	Non-expert (s, r, s')
Offline IL	✗*	✓	✓	✗	✗
Offline ILfO	✗*	✓	✗*	✓	✗
Offline RL	✓	✗	✗	✗	✗
Offline RL w/ Unlabeled Data	✓	✓	✗	✗	✗
Offline PLfO (Proposed)	✓	✓	✓	✓	✓

training data-consistent critic and reward functions, which forces the learned policy to be robust to data deficiency. We show that MAHALO consistently outperforms or matches specialized algorithms across a variety of offline PLfO tasks in theory and experiments. This chapter is adapted from [Li et al., 2023a].

7.1 Introduction

Online reinforcement learning (RL) has shown great promise in solving simulated tasks [Silver et al., 2016, Mnih et al., 2015]. However, exploratory interactions with the environment, which are central to online RL, often can not be afforded in risk-sensitive applications, such as robotics [Ibarz et al., 2021] and healthcare [Gottesman et al., 2018]. In these domains, it is more practical to consider an offline setting [Levine et al., 2020], where data is collected by behavioral policies satisfying certain criteria.

There are two main approaches to solving decision making problems offline: offline imitation learning (IL) [Chang et al., 2021, Kidambi et al., 2021, Kim et al., 2021] and offline RL [Fujimoto et al., 2019, Levine et al., 2020]. Offline IL generally does not assume

access to the reward. Theses approaches learn with a small set of expert demonstrations and potentially a separate dynamics dataset with unknown quality. Offline IL seeks to mimic expert behavior while avoiding distribution shift caused by using offline datasets. Offline imitation learning from observations (ILfO) [Kidambi et al., 2021, Ma et al., 2022b] further relaxes the requirements of expert actions. ILfO allows learning from experts with different action spaces [Edwards et al., 2020], or when the expert has a different action modality or a embodiment [Cao and Sadigh, 2021, Radosavovic et al., 2021].

Offline RL, on the other hand, does not require expert-level demonstrations. It instead assumes that each transition in the offline dataset is labeled with reward. The goal of offline RL is to learn a policy which *1)* always improves upon the behavioral policy [Fujimoto et al., 2019, Laroche et al., 2019], and *2)* can outperform any other policies whose state-action distribution is covered by data [Xie et al., 2021].

However, in real-world applications, it is expensive to either acquire expert-level demonstrations (even if they are observation-only), or label every transition with reward. In this chapter, we propose a more general and realistic formulation called offline *Policy Learning from Observations* (PLfO). Our goal is to learn from datasets where *1)* a subset of trajectories is labeled with rewards, *2)* labeled trajectories may not contain actions, *3)* labeled trajectories may not be of high quality, and *4)* the overall data may not have full coverage. The flexibility of this formulation allows us to directly take advantage of more data sources, such as dynamics data collected for other tasks and reward data collected by a non-expert agent with a different action space.

Offline PLfO considers two offline datasets: the reward dataset $\mathcal{D}_R = \{(s, r, s')\}$ and dynamics dataset $\mathcal{D}_\mathcal{A} = \{(s, a, s')\}$, where the dynamics dataset is consistent with the Markovian dynamics that the learner aims to solve (i.e. it is collected by agents that have the same embodiment as the learner). In offline RL setting, the reward and dynamics datasets are aligned, since they are from the same underlying dataset $\mathcal{D} = \{(s, a, r, s')\}$. Recent work [Yu et al., 2022] relaxes this requirement by assuming that only a subset of transitions are labeled with rewards, i.e., the set of state transitions contained in $\mathcal{D}_R$ is a subset of $\mathcal{D}_\mathcal{A}$. On the contrary, in offline PLfO, we make no assumption on how these two datasets are related to each other.

Offline ILfO can also be viewed as a special case of offline PLfO. Although ILfO does not assume knowledge of reward, it makes an implicit assumption that expert trajectories attain high returns, while making no assumptions on reward information elsewhere (e.g., on the dynamics data $\mathcal{D}_A$). In other words, from the perspective of offline PLfO, expert demonstrations essentially act as the reward-labeled dataset $\mathcal{D}_R$ for ILfO. Practically, we can simply label the expert demonstrations with the maximum reward. This observation is in line with existing work [Fu et al., 2018a, Eysenbach et al., 2021, Smith et al., 2023]. We refer readers to Table 7.1 for a summary of comparison between offline PLfO and existing formulations.

The key challenge to offline PLfO is the mismatch among the reward dataset, the dynamics dataset, and the test-time distribution. We present a generic approach to offline PLfO, called Modality-agnoistic Adversarial Hypothesis Adaptation for Learning from Observations (MAHALO). Built upon the concept of pessimism from offline RL literature [Jin et al., 2021, Liu et al., 2020, Kumar et al., 2020a, Xie et al., 2021, Cheng et al., 2022], MAHALO optimizes for a performance lower bound accounting for insufficient data coverage on reward and dynamics. It can be realized by modifying existing offline RL algorithms based on adversarial training, such as [Xie et al., 2021, Cheng et al., 2022, Uehara and Sun, 2022, Rigter et al., 2022, Xie et al., 2022]. In particular, we present a model-free instantiation of MAHALO built upon ATAC [Cheng et al., 2022], an offline RL algorithm based on a Stackelberg game of relative pessimism. In MAHALO, we consider the actor policy as the leader in the Stackelberg game, and adversarially train critic and reward functions so that they are data-consistent and can detect potential deficiency of the actor policy. As a result, the policy can be robust to the missing data coverage.

The contribution of this work is two-fold. First, we propose offline PLfO, a novel formulation which relaxes data assumption for policy learning with offline data. This general formulation encompasses most existing offline formulations, including, but not limited to, offline IL, ILfO, RL, and RL with unlabeled data. Second, we present MAHALO, a solution to offline PLfO based on pessimism. We further present a model-free realization of MAHALO. In theory and experiments, we show that MAHALO consistently outperforms or matches performance with more specialized algorithms across various offline PLfO scenarios

and tasks.

7.2 A Unified Formulation for Offline RL and IL from Observations

In this section, we first introduce the generic setup of offline policy learning from observations (PLfO). Then we discuss practical scenarios where the data cannot be fully leveraged in offline RL and IL setups but is within the PLfO setup.

7.2.1 Problem Formulation

In offline PLfO, we assume access to pre-collected offline data consisting of transitions. In contrast to typical offline RL, we allow our data to include transitions which contain *either* reward or action. In other words, in offline PLfO, we consider two datasets, a reward dataset $\mathcal{D}_R = \{(s, r, s')\}$ and a dynamics dataset $\mathcal{D}_\mathcal{A} = \{(s, a, s')\}$. We note that these two datasets may not necessarily have an intersection.

We assume that both datasets are compliant with the underlying MDP. For the dynamics dataset, we follow standard compliance assumption in offline RL literature [Jin et al., 2021]: *1)* for any $(s, a, s') \in \mathcal{D}_R$, we have $s' \sim P(\cdot|s, a)$; and *2)* the state-action pairs (s, a) in $\mathcal{D}_\mathcal{A}$ are sampled from the discounted state-action occupancy d^μ of a behavioral policy $\mu : \mathcal{S} \to \Delta(\mathcal{A})$. We slightly abuse notation to use μ to also denote the discounted state-action occupancy d^μ, i.e., $\mu = d^\mu$. For the reward dataset, for any $(s, r, s') \in \mathcal{D}_R$, we assume that the reward function R is defined on the state transition (s, s') and $r = R(s, s')$. We do not make assumption on the underlying distribution of state transitions $(s, s') \sim \nu$, e.g., s' in (s, s') may not be sampled from $P(\cdot|s, a)$ for some action a. With a slight abuse of notation, we will also use ν to denote the underlying distribution of transition tuple containing reward (s, r, s'). Like offline RL, we do not assume the coverage of these datasets on states and actions. The goal of offline PLfO is to learn a policy π which obtains high expected discounted return $J(\pi)$ in MDP $\mathcal{M}$ while using reward and dynamics datasets of limited coverage.

7.2.2 Relation to Existing Formulations

Offline PLfO is a general formulation encompassing many existing problem setups. This means that an offline PLfO algorithm can solve any of the following problems, or the combination of them, via simple reductions.

Offline RL Offline RL studies the problem of policy learning from a reward-labeled transition dataset $\mathcal{D} = \{(s, a, r, s')\}$. The goal of offline RL is to learn the best policy that can be explained by data, while not making assumptions on the data coverage quality. An offline RL algorithm ideally is able to learn the optimal policy of the MDP $\mathcal{M}$, as long as the dataset covers the states and actions that the optimal policy would visit.

Offline RL can be reduced to offline PLfO where the reward dataset and dynamics dataset are generated from the same underlying offline dataset $\mathcal{D} = \{(s, a, r, s')\}$. The alignment assumption makes offline RL in general an easier problem than offline PLfO since there is no mismatch between reward and dynamics data.

Offline RL with unlabeled data Yu et al. [2022], Singh et al. [2020], Hu et al. [2023] consider a formulation where a subset of dynamics data is labeled with reward. In this scenario, the reward data is well-covered by dynamics data. The main challenge here is to leverage unlabeled dynamics data while not suffering from insufficient reward coverage.

Offline IL Offline IL (such as behavior cloning) studies the problem of policy learning using only the transition dataset without reward labels. The transition data is a union of near-optimal expert data $\mathcal{D}_E$ and (optionally) a separately collected data of unknown quality $\mathcal{D}_X$. The data in offline IL does not have full coverage, and the principle of mimicking the expert data in IL also effectively encourages the learner to stay within the the data distribution. In fact, Cheng et al. [2022] show that offline IL can be viewed as an offline RL problem with the largest reward uncertainty: By running an offline RL algorithm that optimizes for the relative performance between the learner and the behavioral policy under data uncertainty, an IL mimicking behavior would naturally occur.

Although IL generally assumes no reward information, it makes an implicit assumption

on expert performance. In other words, IL is a learning problem where only positive (i.e., high return) examples are given. This observation is also in line with existing work which takes a density matching perspective on IL [Ho and Ermon, 2016b, Kim et al., 2021] and reward learning from demonstrations [Fu et al., 2018a, Eysenbach et al., 2021]. Recent work such as [Kim et al., 2021, Chang et al., 2021, Smith et al., 2023] has considered offline IL with a separately-collected dynamics dataset $\mathcal{D}_X$ of unknown quality.

Offline IL can be reduced to offline PLfO: The reward dataset $\mathcal{D}_R = \{(s, R_{\max}, s')\}$ comes from expert demonstrations, where $R_{\max}$ is the maximum reward. The dynamics dataset contains both the expert demonstrations $\mathcal{D}_E$ and, if given, the separately-collected dynamics data $\mathcal{D}_X$.

Offline ILfO Offline ILfO is similar to offline IL, except that the expert demonstrations only contain state transitions, i.e., $\mathcal{D}_E = \{(s_E, s'_E)\}$. As such, offline ILfO can be viewed as offline PLfO with reward dataset $\mathcal{D}_R = \{(s_E, R_{\max}, s'_E)\}$ and dynamics dataset $\mathcal{D}_\mathcal{A} = \mathcal{D}_X$. Compared to the previous setups, offline ILfO faces the challenge of insufficient action coverage, as the expert state transitions (s_E, s'_E) may not be in the dynamics dataset $\mathcal{D}_\mathcal{A}$.

7.2.3 Practical Scenarios

We now consider a few practical examples of reward and dynamics data sources. As we will see, it is likely that the coverage of reward data mismatches with that of dynamics data. In such scenario, offline PLfO formulation is well-suited as it can leverage *all* available data.

Sources of dynamics data Since dynamics data is task-agnostic, dynamics data can be obtained from running data-collection policies on the MDP, as well as any other MDPs with the same dynamics. For example, in a multi-task setting [Yu et al., 2020a], dynamics data can be acquired when solving different task rewards. However, it is expensive to label dynamics data with rewards for *every* task. Realistically, data often only has reward information of the particular task that the data collection is for, which means that some state transitions only have actions, but not rewards.

Sources of reward data One practical strategy to reward labeling is to label randomly sampled trajectories from a pre-collected dynamics dataset [Yu et al., 2022]. This means that a large subset of the dynamics data remains unlabeled. Another setting is to re-use reward data collected by another agent (with the same state space). It is possible that the action information is unavailable or unusable in the underlying MDP. For example, the other agent can have different embodiment or use a different control modality. Additionally, reward information can be implicitly provided through expert demonstrations, as is discussed in Section 7.2.2.

7.3 Modality-agnostic Adversarial Hypothesis Adaptation for Learning from Observations

We propose a generic approach to offline PLfO, called Modality-agnostic Adversarial Hypothesis Adaptation for Learning from Observations (MAHALO). It is inspired by the idea of pessimism in offline RL [Jin et al., 2021].

7.3.1 Solution Concept to Offline PLfO

In order to tackle the heterogeneous uncertainty in offline PLfO, we leverage the concept of version space in the offline RL literature [Cheng et al., 2022, Xie et al., 2021, Rigter et al., 2022, Uehara and Sun, 2022, Xie et al., 2022] to construct a performance lower bound for optimizing policies in MAHALO. Here, a version space, denoted as $\mathcal{J} = \{\hat{J} : \Pi \to \mathbb{R}\}$, is the space of policy performance hypotheses that remain feasible after observing data in $\mathcal{D}_{\mathcal{A}}$ and $\mathcal{D}_R$. For example, if a bipedal robot has experienced that falling down receives zero rewards, then any hypothesis in the version space would give a zero reward to any policy that makes the robot fall, but the hypotheses in the version may disagree on which reward to give for other behaviors.

In MAHALO we use the version space $\mathcal{J}$ to encapsulate uncertainty due to heterogeneous missing coverage. Because the version space $\mathcal{J}$ by definition includes the true performance function J, we can use $\mathcal{J}$ to construct a policy performance lower bound. For example, for a policy π we can compute its absolute performance lower bound naturally as $\min_{\hat{J}\in\mathcal{J}} \hat{J}(\pi)$. Thus we can optimize policies through solving a saddle-point problem: $\max_{\pi\in\Pi} \min_{\hat{J}\in\mathcal{J}} \hat{J}(\pi)$

which provides a way to systematically optimize policy performance accounting for missing information in data.

For the case where the data are fully labeled with rewards and actions, offline RL literature has proposed several designs of the version space $\mathcal{J}$: with MDP models or value functions, in conjunction with absolute or relative pessimism [Cheng et al., 2022, Xie et al., 2022, 2021, Rigter et al., 2022, Uehara and Sun, 2022]. Here we generalize this technique to offline PLfO. In particular, we will design model-free versions of MAHALO, as model-free methods are simpler to implement, use less hyperparameters, and have demonstrated superior empirical performance in offline RL [Yu et al., 2021]. In principle, MAHALO can be realized by *any* version-space offline RL algorithm, including those based on models.

7.3.2 Model-Free Realization of MAHALO

We now present a model-free realization of MAHALO based on the concept of relative pessimism in offline RL [Cheng et al., 2022]. For clarity, we first present the formulation at the population level. We will later provide theoretical analysis for the finite-sample scenario in Section 7.3.2. Another realization of MAHALO based on absolute pessimism [Xie et al., 2021] is introduced and analyzed in [Li et al., 2023a].

We formulate offline PLfO as a Stackelberg game, with the actor policy $\pi \in \Pi$ as the leader. The followers consist of critic $f \in \mathcal{F}$ and reward function $g \in \mathcal{G}$.

$$\hat{\pi} \in \arg\max_{\pi \in \Pi} \mathcal{L}_\mu(\pi, f^\pi) \tag{7.1}$$

$$\text{s.t.} \quad f^\pi \in \arg\min_{f \in \mathcal{F}, g \in \mathcal{G}} \mathcal{L}_\mu(\pi, f) + \alpha \mathcal{E}_\nu(g) + \beta \mathcal{E}_\mu(\pi, f, g),$$

with $\alpha \geq 0, \beta \geq 0$ being hyperparameters, and

$$\mathcal{L}_\mu(\pi, f) \coloneqq \mathbb{E}_\mu\big[f(s, \pi) - f(s, a)\big], \tag{7.2}$$

$$\mathcal{E}_\nu(g) \coloneqq \mathbb{E}_\nu\big[\big(g(s, s') - r\big)^2\big], \tag{7.3}$$

$$\mathcal{E}_\mu(\pi, f, g) \coloneqq \mathbb{E}_\mu\big[\big((f - \bar{g} - \mathcal{P}^\pi f)(s, a)\big)^2\big]. \tag{7.4}$$

where $f(s, \pi) \coloneqq \mathbb{E}_{a \sim \pi(\cdot|s)}[f(s, a)]$ and $\bar{g}(s, a) \coloneqq \mathbb{E}_{s' \sim P(\cdot|s,a)}[g(s, s')]$. This optimization problem can be viewed as a regularized version of the constrained problem with a version

space $\mathcal{J} = \{\hat{J} : \hat{J}(\pi) = J(\mu) + \frac{1}{1-\gamma}\mathcal{L}_\mu(\pi, f), \mathcal{E}_\nu(g) \leq \epsilon_\alpha, \mathcal{E}_\mu(\pi, f, g) \leq \epsilon_\beta\}$ for some $\epsilon_\alpha, \epsilon_\beta \geq 0$ related to α, β.[1] We adopt the regularized version, as it is easier to implement numerically.

To understand the above formulation, let us start by considering the last two terms in the followers' objective function. Recall that ν is the underlying distribution of reward data, so $\mathcal{E}_\nu(g)$ measures whether the candidate reward function g is data consistent. $\mathcal{E}_\mu(\pi, f, g)$ quantifies whether the candidate critic f is Bellman-consistent on the dynamics data for policy π and reward function g. Therefore, with sufficiently large α and β, the reward g^π and critic f^π are both (approximately) consistent with the reward and dynamics data. On the other hand, $\mathcal{L}_\mu(\pi, f)$ is the relative performance of between the candidate policy π and behavioral policy μ, with value estimated by critic f. The followers minimizes this quantity, meaning that f^π provides a *pessimistic* relative evaluation of policy π (which will be formally shown in Proposition 7.3.1). The learned actor policy $\hat{\pi}$ maximizes this lower bound to improve over the the behavioral policy.

We would like to stress on the importance of making the reward function additionally minimize for the Bellman error in the followers' objective (7.1). By minimizing (7.4), the reward and critic functions work *together* to provide a pessimistic evaluation of the policy. In other words, the Bellman error (7.4) connects the learned reward function to the pessimistic loss (7.2), since the reward function can change in a way to make the critic more pessimistic.

The Stackelberg game for MAHALO in (7.1) is similar to ATAC [Cheng et al., 2022]. The main difference is that ATAC uses the observed reward in the Bellman error term as it assumes the reward is available for every transition. MAHALO, on the other hand, trains the reward function to be consistent with the reward data (7.3) and Bellman-consistent with a pessimistic critic function (7.4) on the dynamics data.

Below we discuss three desirable properties of MAHALO. First, given large dynamics and reward datasets, MAHALO can outperform any policy whose state-action distribution is well-covered by both datasets. Second, MAHALO ensures safe policy improvement [Fujimoto et al., 2019, Laroche et al., 2019], i.e., the learned policy $\hat{\pi}$ is no worse than the

[1] This analogy between the constrained and the regularized versions can be derived following the principle in [Xie et al., 2021].

behavioral policy μ given sufficient data. Third, MAHALO can automatically adapt to the structure within data. When applied to more restrictive formulations such as offline RL, offline IL, and offline ILfO, MAHALO shows similar behavior as specialized algorithms.

Theoretical Properties

We analyze the solution to a finite-sample version of (7.1) based on dynamics dataset $\mathcal{D}_{\mathcal{A}}$ and reward dataset $\mathcal{D}_R$:

$$\begin{aligned} \hat{\pi} &\in \arg\max_{\pi \in \Pi} \mathcal{L}_{\mathcal{D}_{\mathcal{A}}}(\pi, f^{\pi}) \\ \text{s.t.} \quad f^{\pi} &\in \arg\min_{f \in \mathcal{F}, g \in \mathcal{G}} \mathcal{L}_{\mathcal{D}_{\mathcal{A}}}(\pi, f) + \alpha \mathcal{E}_{\mathcal{D}_R}(g) + \beta \mathcal{E}_{\mathcal{D}_{\mathcal{A}}}(\pi, f, g), \end{aligned} \tag{7.5}$$

where $\mathcal{L}_{\mathcal{D}_{\mathcal{A}}}(\pi, f)$ and $\mathcal{E}_{\mathcal{D}_R}(g)$ are the empirical estimates of $\mathcal{L}_{\mu}(\pi, f)$ and $\mathcal{E}_{\nu}(g)$, respectively, and

$$\begin{aligned} \mathcal{E}_{\mathcal{D}_{\mathcal{A}}}(\pi, f, g) \coloneqq \mathbb{E}_{\mathcal{D}_{\mathcal{A}}}\big[\big(f(s,a) - g(s,s') - \gamma f(s', \pi)\big)^2\big] \\ - \min_{f' \in \mathcal{F}} \mathbb{E}_{\mathcal{D}_{\mathcal{A}}}\big[\big(f'(s,a) - g(s,s') - \gamma f(s', \pi)\big)^2\big]. \end{aligned} \tag{7.6}$$

The quantity $\mathcal{E}_{\mathcal{D}_{\mathcal{A}}}(\pi, f, g)$ is the estimated Bellman error Antos et al. [2008]. We show in [Li et al., 2023a] that $\mathcal{E}_{\mathcal{D}_{\mathcal{A}}}(\pi, f, g)$ can be used to approximate the Bellman error $\mathcal{E}_{\mu}(\pi, f, g)$ defined in (7.4).

For clarity purposes, we make a perfect realizability and completeness assumption below. Our analysis can be easily modified to consider an approximate version.

Assumption 7.3.1 (Realizability & Completeness). We assume $\mu \in \Pi$, $R \in \mathcal{G}$, and for all $\pi \in \Pi$, $Q^{\pi} \in \mathcal{F}$. In addition, $\inf_{f' \in \mathcal{F}} \|f' - \bar{g} - \mathcal{P}^{\pi} f\|_{\mu} = 0$, $\forall g \in \mathcal{G}, f \in \mathcal{F}$.

Due to the nature of offline learning, we will compare the performance of the learned policy $\hat{\pi}$ with a policy π whose induced state-action occupancy d^{π} is "well-covered" by data distributions. Similar to Xie et al. [2021], Cheng et al. [2022], Uehara and Sun [2022], we define error transfer coefficients to measure distribution shift from the data distributions based on critic class $\mathcal{F}$, and reward class $\mathcal{G}$. This is a weaker notion than, e.g., density ratio [Munos and Szepesvári, 2008].

Definition 7.3.1 (Error transfer coefficients)**.** The Bellman error transfer coefficient between $\rho \in \Delta(\mathcal{S} \times \mathcal{A})$ and $\mu \in \Delta(\mathcal{S} \times \mathcal{A})$ under policy π, critic class $\mathcal{F}$ and reward class $\mathcal{G}$ is defined as

$$\mathcal{C}(\rho; \mu, \mathcal{F}, \mathcal{G}, \pi) \coloneqq \sup_{f \in \mathcal{F}, g \in \mathcal{G}} \frac{\|f - \bar{g} - \mathcal{P}^\pi f\|_{2,\rho}^2}{\|f - \bar{g} - \mathcal{P}^\pi f\|_{2,\mu}^2}. \tag{7.7}$$

Similarly, the reward error transfer coefficient between ρ and $\nu \in \Delta(\mathcal{S} \times \mathcal{S})$ under reward class $\mathcal{G}$ is defined as

$$\mathcal{C}(\rho; \nu, \mathcal{G}) \coloneqq \sup_{g \in \mathcal{G}} \frac{\|\bar{g} - \bar{R}\|_{2,\rho}^2}{\|g - R\|_{2,\nu}^2}. \tag{7.8}$$

We use $d_{\mathcal{F},\mathcal{G},\Pi}$ to denote the joint statistical complexity of critic class $\mathcal{F}$, reward class $\mathcal{G}$ and policy class Π, and use $d_{\mathcal{G}}$ to denote the statistical complexity of reward class $\mathcal{G}$ (e.g., when $\mathcal{F}$, $\mathcal{G}$ and Π are all finite, we have $d_{\mathcal{F},\mathcal{G},\Pi} = \mathcal{O}(\log |\mathcal{F}||\mathcal{G}||\Pi|/\delta)$ where δ is the failure probability). We refer the readers to [Li et al., 2023a] where we establish statistical complexity using covering number. We now state the main theoretical property of MAHALO of relative pessimism in (7.5).

Theorem 7.3.1. *Under Assumption 7.3.1, let $\hat{\pi}$ be the solution to* (7.5) *and let $\pi \in \Pi$ be any comparator policy. Let $C_1 \geq 1, C_2 \geq 1$ be constants, $\rho \in \Delta(\mathcal{S} \times \mathcal{A})$ be a distribution that satisfy $\mathcal{C}(\rho; \mu, \mathcal{F}, \mathcal{G}, \pi) \leq C_1$ and $\mathcal{C}(\rho; \nu, \mathcal{G}) \leq C_2$. Define $\epsilon_\mu \coloneqq V_{max}^2 d_{\mathcal{F},\mathcal{G},\Pi}/|\mathcal{D}_\mathcal{A}|$, $\epsilon_\nu \coloneqq R_{max}^2 d_\mathcal{G}/|\mathcal{D}_R|$ and $\epsilon \coloneqq (\sqrt{C_1 \epsilon_\nu} + \sqrt{C_2 \epsilon_\mu})^2$. Choosing $\alpha = \Theta\left(V_{max}^{1/3} \epsilon^{1/3}/\epsilon_\nu\right)$ and $\beta = \Theta\left(V_{max}^{1/3} \epsilon^{1/3}/\epsilon_\mu\right)$, with high probability,*

$$\begin{aligned} & J(\pi) - J(\hat{\pi}) \\ \leq & \mathcal{O}\left(\frac{1}{1-\gamma}\left(\frac{C_1^{1/3} V_{max} (d_{\mathcal{F},\mathcal{G},\Pi})^{1/3}}{|\mathcal{D}_\mathcal{A}|^{1/3}} + \frac{C_2^{1/3} R_{max} (d_\mathcal{G})^{1/3}}{|\mathcal{D}_R|^{1/3}}\right)\right) \\ & + \underbrace{\frac{\langle d^\pi \setminus \rho, \bar{g}^\pi + \mathcal{P}^\pi f^\pi - f^\pi \rangle}{1-\gamma}}_{\textit{off-support error (dynamics)}} + \underbrace{\frac{\langle (d^\pi \ominus \mu) \setminus \rho, |\bar{R} - \bar{g}^\pi| \rangle}{1-\gamma}}_{\textit{off-support error (reward)}}, \end{aligned} \tag{7.9}$$

where $(d^\pi \ominus \mu) \coloneqq d^\pi \setminus \mu + \mu \setminus d^\pi$ with $(d_1 \setminus d_2)(s,a) \coloneqq \max(d_1(s,a) - d_2(s,a), 0)$ and $\langle \iota, f \rangle \coloneqq \sum_{(s,a) \in \mathcal{S} \times \mathcal{A}} \iota(s,a) f(s,a)$.

The first term in (7.9) is the statistical error, which vanishes as $|\mathcal{D}_\mathcal{A}|, |\mathcal{D}_R| \to \infty$. The second and third terms measure how much the comparator policy π is outside of the support

of dynamics and reward data distributions. In other words, our learned policy $\hat{\pi}$ can compete with any policy π that is well-supported by both dynamics and reward data. Compared with Theorem 5 of ATAC by Cheng et al. [2022], we have an extra term about the statistical error of the estimated reward, and an off-support reward error, because we do not assume access to rewards on all transitions.

Despite using partial reward labels, MAHALO has a robust policy improvement property, similar to ATAC [Cheng et al., 2022], which guarantees that, for a known range of hyperparameters, the learned policy $\hat{\pi}$ is no worse than the behavioral policy by more than statistical errors.

Proposition 7.3.1 (Robust Policy Improvement)**.** *Under Assumption 7.3.1, let $\hat{\pi}$ be the solution to* (7.5)*. Let ϵ_μ and ϵ_ν be as defined in Theorem 7.3.1. We have, for any fixed $\alpha \geq 0$ and $\beta \geq 0$, with high probability,*

$$\begin{aligned} & J(\mu) - J(\hat{\pi}) \\ \leq & \mathcal{O}\left(\frac{V_{max}}{1-\gamma}\sqrt{\frac{d_{\mathcal{F},\mathcal{G},\Pi}}{|\mathcal{D}_\mathcal{A}|}} + \frac{\alpha R_{max}^2 d_\mathcal{G}}{(1-\gamma)|\mathcal{D}_R|} + \frac{\beta V_{max}^2 d_{\mathcal{F},\mathcal{G},\Pi}}{(1-\gamma)|\mathcal{D}_\mathcal{A}|}\right). \end{aligned} \tag{7.10}$$

As $|\mathcal{D}_\mathcal{A}|$ and $|\mathcal{D}_R| \to \infty$, we can see that the solution actor policy $\hat{\pi}$ is guaranteed to be no worse than the behavioral policy μ with *any* choices of fixed $\alpha, \beta \geq 0$.

Adaption to Structure within Data

Since MAHALO can be used to solve offline PLfO, MAHALO can be used to solve more restrictive problems via simple reductions. Then, a natural question is: Can MAHALO achieve similar performance as specialized algorithms? The answer is yes. Below we show that this is because MAHALO can adapt to the hidden structure within data despite being agnostic to the relationship between dynamics and reward datasets.

Offline RL In offline RL, since the reward and dynamics datasets are aligned, we have $\mathcal{E}_\nu(g) = \mathbb{E}_\mu[\mathbb{E}_{s' \sim P(\cdot|s,a)}[(g(s,s') - R(s,s'))^2]]$. The expected Bellman error of critic f (with the true reward R), $\mathcal{E}_\mu(\pi, f, R)$, can be upper bounded by $2\mathcal{E}_\nu(g) + 2\mathcal{E}_\mu(\pi, f, g)$. With sufficiently large α and β, for any actor policy π, its corresponding critic f^π is an approximately

Algorithm 3 MAHALO (realized by ATAC)

Input: Batch datasets $\mathcal{D}_R$, $\mathcal{D}_\mathcal{A}$; policy π, critics f_1, f_2; coefficients α, $\beta \geq 0$ and $\tau, w \in [0, 1]$.
Initialize target networks $\bar{f}_1 \leftarrow f_1$, $\bar{f}_2 \leftarrow f_2$.
for $k = 1, 2, \ldots, K$ **do**
  Sample minibatches $\mathcal{D}_R^{\text{mini}}$ and $\mathcal{D}_\mathcal{A}^{\text{mini}}$ from $\mathcal{D}_R$ and $\mathcal{D}_\mathcal{A}$.
  Compute critic loss $l_{\text{critic}}(f_i) \leftarrow \mathcal{L}_{D_\mathcal{A}^{\text{mini}}}(\pi, f_i) + \beta \mathcal{E}^w_{D_\mathcal{A}^{\text{mini}}}(\pi, f_i, g)$, for $i \in \{1, 2\}$
  Compute reward loss $l_{\text{reward}}(g) \leftarrow \alpha \mathcal{E}_{\mathcal{D}_R^{\text{mini}}}(g) + \beta \sum_{i=\{1,2\}} \mathcal{E}^w_{D_\mathcal{A}^{\text{mini}}}(\pi, f_i, g)$.
  Update critic network $f_i \leftarrow \text{Proj}_\mathcal{F}(f_i - \eta_{\text{fast}} \nabla l_{\text{critic}})$ for $i \in \{1, 2\}$.
  Update reward network $g \leftarrow \text{Proj}_\mathcal{G}(g - \eta_{\text{fast}} \nabla l_{\text{reward}})$.
  Compute actor loss $l_{\text{actor}} \leftarrow -\mathcal{L}_{\mathcal{D}_\mathcal{A}^{\text{mini}}}(\pi, f_1)$.
  Update actor network $\pi \leftarrow \text{Proj}_\Pi(\pi - \eta_{\text{slow}} \nabla l_{\text{actor}})$
  Update target $\bar{f} \leftarrow (1 - \tau)\bar{f} + \tau f$ for $(f, \bar{f}) \in \{(f_i, \bar{f}_i)\}_{i=1,2}$.
end for

Bellman-consistent critic function. Therefore, MAHALO behaves similarly as ATAC [Cheng et al., 2022]. This can also be seen from Theorem 7.3.1. Since $|\mathcal{D}_\mathcal{A}| = |\mathcal{D}_R|$, the statistical error is dominated by the first term, which is the same as the statistical error as ATAC. In offline RL, good dynamics coverage implies good data coverage, i.e., we have $(d^\pi \ominus \mu) \setminus \rho \leq d^\pi \setminus \rho$. This gives us a similar off-support error term.

Offline RL with unlabeled data For the sake of simplicity, we consider a tabular setting. In this case, regardless of the policy $\pi \in \Pi$, with sufficiently large α, the reward function g^π such that $g^\pi(s, s') \approx R(s, s')$ when (s, s') is within coverage of reward data ν; the value of $g^\pi(s, s')$ on other states would adapt pessimistically according to the learner policy. The critic f^π is effectively conducting a pessimistic policy evaluation for π in such a reward function. This means that MAHALO in the tabular setting has a similar behavior as UDS [Yu et al., 2022], a strategy where zero reward is given to *all* unlabeled data. The difference, though, is that MAHALO still assigns an accurate reward to unlabeled dynamics transitions (s, a, s') when (s, s') is within coverage of ν. This implies that MAHALO induces

less bias than UDS, even though UDS knows more about the underlying data generation process.

Offline IL and ILfO In the simplest offline IL setting where only the set of expert demonstrations $\mathcal{D}_E$ is given, Proposition 7.3.1 (with $\pi_E = \mu$) shows that the learned policy $\hat{\pi}$ is no worse than the expert policy π_E (in terms of $R(s, s') = R_{\max}\mathbb{1}[(s, s') \in \text{supp}(\nu)]$) up to statistical errors. Now consider the scenario where we have access the expert demonstrations $\mathcal{D}_E$ (which may or may not have actions)[2] and a separately collected dynamics dataset $\mathcal{D}_X$ with unknown quality. Theorem 7.3.1 (with $R(s, s') = R_{\max}\mathbb{1}[(s, s') \in \text{supp}(\nu)]$) shows that the learned policy would stay within the support of the expert distribution, similar to Wang et al. [2019], Smith et al. [2023].

Summary MAHALO is an general, data-agnostic algorithm for solving offline PLfO problems. When applied to more restrictive settings when data presents additional structure, MAHALO can automatically adapt to such structure, and achieves behavior on par with existing specialized algorithms. This means that MAHALO can leverage broad sources of data and no special care, e.g. data alignment or management, needs to be taken during data collection.

7.3.3 Implementation

The MAHALO realization above can be implemented by making a few simple modifications to ATAC [Cheng et al., 2022]. The resulting algorithm is presented in Algorithm 3, with the modifications marked in magenta. This implementation of MAHALO is based on a reduction of two-player game to no-regret policy optimization [Cheng et al., 2022]. We use ADAM [Kingma and Ba, 2014] optimizer with a faster learning rate η_{fast} for the critic and reward functions, and a smaller learning rate η_{slow} for the actor.

[2]When $\mathcal{D}_E$ contains action, we can alternatively replace $\mathcal{L}_{\mathcal{D}_A}(\pi, f)$ with $\mathcal{L}_{\mathcal{D}_E}(\pi, f)$, which would ensure robust policy improvement to the expert policy.

Actor and Critic Update

We use a strategy for updating the critic similar to ATAC. ATAC uses a surrogate to the Bellman error term in Eq. (7.4) called double Q residual algorithm (DQRA) loss, which combines double Q heuristics [Fujimoto et al., 2018], the residual algorithm [Baird, 1995], and target networks [Mnih et al., 2015]. The critic is parameterized by two networks $\{f_1, f_2\}$, each with a delayed target $\{\bar{f}_1, \bar{f}_2\}$. The target value is computed by taking the minimum of the two targets $\bar{f}_{\min}(s,a) = \min_{i\in\{1,2\}} \bar{f}_i(s,a)$. DQRA uses a convex combination of the temporal difference (TD) losses of the critic network and target networks to stabilize learning. For $i \in \{1,2\}$, the DQRA loss is defined as

$$\begin{aligned}\mathcal{E}^{w}_{\mathcal{D}^{\text{mini}}_{\mathcal{A}}}(\pi, f_i, g) \coloneqq (1-w)\mathcal{E}^{\text{td}}_{\mathcal{D}^{\text{mini}}_{\mathcal{A}}}(\pi, f_i, f_i, g) \\ + w\mathcal{E}^{\text{td}}_{\mathcal{D}^{\text{mini}}_{\mathcal{A}}}(\pi, f_i, \bar{f}_{\min}, g),\end{aligned} \tag{7.11}$$

where $w \in [0,1]$ is the weight and the TD loss is given by

$$\mathcal{E}^{\text{td}}_{\mathcal{D}^{\text{mini}}_{\mathcal{A}}}(\pi, f, f', g) \coloneqq \mathbb{E}_{\mathcal{D}^{\text{mini}}_{\mathcal{A}}}[(f(s,a) - g(s,s') - \gamma f'(s',\pi))^2].$$

Note that here we use predicted reward $g(s,s')$ since reward r is not observed in $\mathcal{D}_{\mathcal{A}}$. After each gradient update, we apply an ℓ_2 projection on critic network weights (not on bias terms) (line 7) as is done in ATAC [Cheng et al., 2022]. We update the actor using the same way as ATAC. The actor policy optimizes for a *single* critic f_1 and uses a Lagrangian relaxation of minimum entropy constraint similar to SAC [Haarnoja et al., 2018].

Reward Update

The major difference between Algorithm 3 and ATAC is the reward function update. We estimate the reward prediction loss empirically from minibatches sampled from the reward dataset $\mathcal{D}_R$: $\mathcal{E}_{\mathcal{D}^{\text{mini}}_R}(g) = \mathbb{E}_{\mathcal{D}^{\text{mini}}_R}[(g(s,s')-r)^2]$. The reward loss is the weighted sum of reward prediction loss $\mathcal{E}_{\mathcal{D}^{\text{mini}}_R}(g)$ and DQRA losses $\sum_{i=\{1,2\}} \mathcal{E}^{w}_{D^{\text{mini}}_{\mathcal{A}}}(\pi, f_i, g)$ (line 6). The DQRA losses connect the reward to the critic which minimizes also the performance difference; as a result, the learned reward function is also pessimistically estimated. In other words, the critic and the reward functions jointly form a hypothesis that adversarially adapts to the learner's policy.

Table 7.2: Five scenarios of PLfO considered in our experiments.

Scenarios	Mixed Quality Data	Expert
ILfO	state + action	state
IL	state + action	state + action
RLfO	state + action	state + reward
RL-expert	state + action	state + action + reward
RL-sample	state + action + (sampled trajs) reward	-

7.4 Experiments

We aim to answer the following questions: (a) Is MAHALO effective in solving different instances of offline PLfO problems? (b) Can MAHALO achieve similar performance as other specialized algorithms? (c) Whether MAHALO can obtain comparable performance to oracle algorithms with full reward and dynamics information? (d) In what situation is the pessimistic reward function of MAHALO critical in achieving good performance?

Scenarios To answer question (a), we design five instances of offline PLfO inspired by practical scenarios. **ILfO:** The learner is presented with a relatively small set of expert-level state-only trajectories, and a dynamics dataset of mixed quality. The dynamics data contains trajectories collected by policies with different performance-level. **IL:** Similar to ILfO, but we additionally provide expert actions to the learner. **RLfO:** Similar to ILfO, but the learner is additionally given the reward along the expert trajectories. This simulates the scenario where we provide manual labels to the expert demonstrations. **RL-expert:** The learner is presented with the expert action in addition to what is given in RLfO. **RL-sample:** The learner is provided with the mixed quality dynamics data, with a subset of trajectories labeled with reward. The information available in each scenario is summarized in Table 7.2.

Baselines To address question (b), we consider a few specialized baseline algorithms for each scenario. We consider behavior cloning from observation (**BCO**) [Torabi et al., 2018] as a baseline for ILfO, and behavior cloning (**BC**) for IL. We also include **SMODICE** [Ma et al., 2022b], a state-of-the art offline ILfO algorithm as a baseline algorithm for ILfO, and a variation of SMODICE which uses a state-action discriminator as a baseline for offline IL. Since IL, RL-expert and RL-sample can be viewed as RL with unlabeled data, we present two baselines for these settings: running an offline RL algorithm, we use **ATAC** [Cheng et al., 2022] since it is the closest to MAHALO, only on labeled data and **UDS** [Yu et al., 2022]. We note that UDS requires knowing the common transitions between reward and dynamics datasets. We implement UDS with ATAC, which is slightly different than Yu et al. [2022], where CQL [Kumar et al., 2020a] is used. Since learning inverse dynamics is a common approach to learning with observations [Torabi et al., 2018, Edwards et al., 2020], we implement a baseline called action prediction (**AP**). It pretrains an inverse dynamics model on the dynamics dataset, predicts the missing actions in the reward dataset, and runs ATAC [Cheng et al., 2022] on the reward dataset. For question (d), we implement a baseline algorithm called reward prediction (**RP**). It pretrains a reward function on the reward dataset. This reward function is then fixed for offline RL training using ATAC [Cheng et al., 2022]. Comparing MAHALO with RP can give us information on whether the adversarial training of reward function in MAHALO is effective. Finally, to answer question (c), we train ATAC on a fully-labeled dynamics dataset (**Oracle**) to evaluate if MAHALO can achieve comparable performance with a privileged offline RL algorithm (which has access to more information).

We evaluate MAHALO and above-mentioned algorithms on two sets of environments: locomotion tasks from the D4RL benchmark [Fu et al., 2020] and robot manipulation tasks from the Meta-World domain [Yu et al., 2020a].

7.4.1 Evaluation on D4RL

We consider three environments from D4RL [Fu et al., 2020]: hopper-v2, walker2d-v2, and halfcheetah-v2. The rewards in the three environments promote moving forward. In hopper

Table 7.3: Results on D4RL benchmark Fu et al. [2020]. We show the average normalized score over 50 evaluation trials across 10 random seeds. (See [Li et al., 2023a] for standard errors.). Algorithms with scores greater than 90% of the best score (excluding Oracle) are in bold. † ATAC only uses data with both dynamics and reward information. + Oracle has access to reward for all dynamics data.

Scenario	Dataset	MAHALO	RP	AP	UDS	ATAC†	BCO	BC	SMODICE	Oracle+
ILfO	hopper	**104.66**	**97.48**	45.97	-	-	46.80	-	67.72	-
	walker	**88.60**	77.15	61.11	-	-	63.02	-	1.52	-
	halfcheetah	**61.24**	36.00	4.87	-	-	5.16	-	**59.64**	-
IL	hopper	**104.06**	**97.88**	53.35	32.21	63.56	-	32.12	74.75	-
	walker	**89.03**	77.71	63.53	8.45	78.35	-	18.78	0.94	-
	halfcheetah	**54.99**	23.20	3.74	25.82	3.64	-	22.36	**58.40**	-
RLfO	hopper	**106.47**	**105.65**	47.01	-	-	-	-	-	103.39
	walker	**96.65**	**97.26**	63.30	-	-	-	-	-	98.52
	halfcheetah	50.38	**68.66**	3.35	-	-	-	-	-	63.57
RL-	hopper	87.73	**105.56**	51.54	**98.63**	65.45	-	-	-	103.39
expert	walker	**103.18**	**98.31**	56.27	72.97	66.40	-	-	-	98.52
	halfcheetah	48.43	**64.37**	3.47	13.68	3.41	-	-	-	63.57
RL-	hopper	**103.08**	**101.66**	71.92	0.95	71.50	-	-	-	103.34
sample	walker	**95.00**	**94.59**	5.94	0.00	0.48	-	-	-	95.73
	halfcheetah	**68.30**	**68.71**	13.26	20.36	19.38	-	-	-	69.91

and walker2d, the agent is required to stay within a health height range, otherwise the episode terminates. We construct a mixed quality dynamics dataset with 3.4 M transitions through concatenating the random, medium, medium-replay, and full-replay datasets. The expert data is consisted of 10k transitions ($\sim$ 10 trajectories) generated by randomly sampling trajectories from the expert dataset. For RL-sample, we sample 34k transitions (1% of overall data).

The normalized return for five scenarios for each dataset is listed in Table 7.3. MAHALO achieves top performance in almost every task except halfcheetah. MAHALO also shows comparable performance with the privileged oracle. Reward prediction (RP) also performs well in all RL tasks. It, however, does not perform as well in IL tasks, since the reward

function of RP can not generalize beyond the constant training reward. UDS does not perform well in these scenarios potentially due to being overly pessimistic. SMODICE uniformly performs worse than MAHALO in hopper and walker tasks. We hypothesize that the fixed discriminator reward function of SMODICE becomes overly pessimistic and discourages policy learning when the expert and dynamics distribution mismatches with one another.

7.4.2 Evaluation on Meta-World

We additionally evaluate MAHALO and other baseline algorithms on five robot manipulation tasks from Meta-World [Yu et al., 2020a]. For Meta-World tasks, we use a non-positive reward function that promotes agent to reach the goal, which is a terminal state, as fast as possible. We generate a mixed quality dataset by adding different level of noises to a scripted policy provided by Meta-World for each task. We collect 100 trajectories each for zero-mean Gaussian noise with standard deviations $[0.1, 0.5, 1.0]$, and use these 300 trajectories as the mixed quality dataset. We separately collect 100 expert trajectories. For, RL-sample, we randomly sample trajectories to label 50% of the dataset. Note that we use more reward data for Meta-World since the state space (which includes the space of goals) is larger.

We observe that MAHALO is one of the overall best-performing algorithms. ATAC also achieves strong results in IL and RL-expert. This is because ATAC, in these scenarios, is only presented with expert data (we do not see similar effect in D4RL tasks since the expert dataset is much smaller there). UDS shows similar performance as MAHALO in RL scenarios. They, however, perform worse than MAHALO in IL. Reward prediction (RP) performs worse than MAHALO in most cases, especially in IL and ILfO scenarios. This shows that the pessimistic reward function in MAHALO is critical in achieving robust performance across different tasks and scenarios. SMODICE is not able to train reliably, and often diverges during training. We therefore take the success rate of the best performing policy during training instead of the final one. We find that the best policy of SMODICE underperforms the final policy of most algorithms, including MAHALO, potentially due to

this instability in training.

7.5 Conclusion

In this chapter, we consider the explicit structure of robotics data given by its composition. A subset of robotics data, such as data collected on a different task, may only provide dynamics information, while another subset may only contain reward information. By reasoning about such structure, we propose a new offline learning formulation, called offline PLfO, which relaxes structural assumptions made by existing offline learning paradigms such as offline RL, offline RL with unlabeled data, offline IL, and offline ILfO, etc. We propose a learning algorithm called MAHALO based on the idea of pesssimism. MAHALO can be used to solve any offline PLfO problems, which means that it can be applied to most existing paradigms. Empirically, we show that MAHALO can achieve comparable or better performance than state-of-the-art algorithms on those existing learning paradigms.

7.6 Related Work

Offline RL Existing work on offline RL can be broadly classified into model-based approaches [Yu et al., 2020b, Kidambi et al., 2020, Yu et al., 2021, Uehara and Sun, 2022, Rigter et al., 2022] and model-free approaches [Jin et al., 2021, Fujimoto and Gu, 2021, Xie et al., 2021, Cheng et al., 2022, Kumar et al., 2020a, Kostrikov et al., 2021]. The main challenge to offline RL is the mismatch between the offline dataset and test-time distribution. Two common strategies to offline RL are behavior regularization [Fujimoto and Gu, 2021, Wu et al., 2019, Kostrikov et al., 2021] (restricting policy to be close to the behavioral policy) and pessimism [Jin et al., 2021, Liu et al., 2020, Xie et al., 2021, Cheng et al., 2022, Kumar et al., 2020a]. Our work follows more closely with model-free approaches built upon the concept of pessimism. In particular, we draw inspirations from approaches which conduct pessimistic policy evaluation on a version space of data-consistent hypotheses [Xie et al., 2021, Cheng et al., 2022, Uehara and Sun, 2022, Rigter et al., 2022]. However, we consider a more general formulation than offline RL, where reward or action can be missing from a subset of data. Edwards et al. [2020] approach RL from observation by combining offline RL with a separately learned inverse dynamics model. It however makes implicit

assumption that the dynamics data is abundant.

Offline RL with unlabeled data Yu et al. [2022] and Singh et al. [2020] consider a setting that reward can be missing from a subset of data, and uses a strategy of giving zero reward to these transitions. We show that MAHALO has a similar behavior when applied to this setting, while incurring less bias by correctly labeling rewards to transitions which are in support of the reward data. Recently, a strategy called PDS [Hu et al., 2023] is proposed to construct a pessimistic reward function using a ensemble of neural networks. MAHALO differs from PDS as the pessimistic reward function is constructed together with a pessimistic critic function. We also provide theoretical analysis of MAHALO in a general function approximator setting while PDS only has theoretical guarantees for linear MDPs. Li et al. [2023c] consider a related setting where rewards can be mislabeled or imperfect. This is slightly different to our problem formation, where the reward (and action) for each transition is either correct or missing.

Offline IL Our work is also related to offline IL, where the expert dataset and the optional, often separately-collected, dynamics dataset can have different coverage. Zolna et al. [2020] adapt GAIL [Ho and Ermon, 2016b] to an offline setting. Kim et al. [2021] seek to match the state-action occupancy using distribution correction estimation. Chang et al. [2021] minimize the state-action occupancy divergence using a pessimistic model. More recently, DWBC Xu et al. [2022a] propose to approach offline IL through weighted behavior cloning, where the weights are given by a discriminator. CLARE [Yue et al., 2023] extends maximum entropy inverse reinforcement learning to an offline setting with an unknown quality dynamics dataset. Smith et al. [2023] propose to run offline RL with a binary reward indicating whether the transition is generated by the expert. These approaches require knowing the actions in expert demonstrations, which is a more restrictive assumption than our formulation.

IL from observations ILfO further relaxes the assumption of observing expert actions. Earlier approaches [Torabi et al., 2018, 2019a,b, Yang et al., 2019, Schmeckpeper et al.,

2021] focus more on the online setting, where additional data collection is allowed when training the policy. There are a few recent papers that consider the offline ILfO setting. For example, Zhu et al. [2020], Ma et al. [2022b] take an off-policy distribution matching approach, and Kidambi et al. [2021] use a model-based approach to minimax IL. These work, however, assumes a near-optimal expert. MAHALO, however, can work with both near-optimal expert trajectories and non-expert state-only trajectories labeled with reward.

Learning with dynamics mismatch Learning from observation is often closely related to learning with dynamics mismatch [Liu et al., 2019, Desai et al., 2020, Cao and Sadigh, 2021, Radosavovic et al., 2021, Cao et al., 2021, Ma et al., 2022b], as it is hard to directly use action information from a different MDP. Although MAHALO can potentially handle this setting, we consider it beyond the scope of this thesis and defer it to future work.

Learning reward functions When applied to offline IL/ILfO settings, our approach also has connection with work on reward learning from demonstrations [Ng et al., 2000, Abbeel and Ng, 2004b, Ziebart et al., 2008b, Fu et al., 2018a,b, Singh et al., 2019, Eysenbach et al., 2021, Konyushkova et al., 2020, Kim et al., 2020], as MAHALO produces a reward function. The main difference between MAHALO and this line of work is that we do not treat reward learning and policy learning as two separate processes. We view the learned reward function rather as a by-product of the algorithm.

Table 7.4: Success rate of the final policy (with the exception of SMODICE*) on Meta-World [Yu et al., 2020a]. The success rate is computed over 50 evaluation episodes. We report the average success rate across 10 random seeds. (See Li et al. [2023a] for standard errors). We consider an episode success if it is able to reach the goal within 128 steps. * SMODICE often diverges during training; we therefore take the success rate of its best performing policy during training instead of the final one.

Scenario	Dataset	MAHALO	RP	AP	UDS	ATAC†	BCO	BC	SMODICE*	Oracle+
ILfO	reach	**65.0**	**62.6**	13.0	-	-	11.6	-	10.6	-
	push	**62.4**	11.6	12.4	-	-	14.6	-	0.4	-
	plate-slide	**100.0**	22.0	**94.0**	-	-	75.4	-	0.0	-
	handle-press	75.4	32.4	**96.6**	-	-	**87.8**	-	16.6	-
	button-press	**100.0**	**100.0**	**93.6**	-	-	**93.8**	-	0.4	-
IL	reach	24.6	23.6	38.0	21.6	**98.0**	-	62.2	19.8	-
	push	**92.6**	11.8	35.2	79.2	50.0	-	**91.4**	0.2	-
	plate-slide	80.4	34.4	**89.8**	76.2	**89.4**	-	**85.3**	0.0	-
	handle-press	71.4	34.8	**100.0**	35.2	**97.0**	-	75.2	20.2	-
	button-press	**100.0**	**99.8**	**96.2**	**100.0**	**99.6**	-	**100.0**	0.2	-
RLfO	reach	**86.4**	**88.0**	15.0	-	-	-	-	-	51.6
	push	**58.2**	32.0	20.2	-	-	-	-	-	91.8
	plate-slide	**100.0**	**100.0**	83.2	-	-	-	-	-	100.0
	handle-press	77.8	84.4	**96.0**	-	-	-	-	-	78.6
	button-press	**100.0**	**100.0**	**92.6**	-	-	-	-	-	100.0
RL-expert	reach	39.2	54.0	42.0	57.8	**98.4**	-	-	-	51.6
	push	**95.6**	88.6	40.4	**90.4**	**99.6**	-	-	-	91.8
	plate-slide	**100.0**	**100.0**	89.6	**99.4**	85.4	-	-	-	100.0
	handle-press	72.6	82.0	**97.6**	81.2	**99.0**	-	-	-	78.6
	button-press	**100.0**	**100.0**	**97.0**	**100.0**	**100.0**	-	-	-	100.0
RL-sample	reach	**86.6**	**87.4**	63.0	**87.4**	**85.4**	-	-	-	88.8
	push	40.6	**47.8**	29.2	35.8	35.0	-	-	-	46.0
	plate-slide	**100.0**	**100.0**	**97.6**	**100.0**	**99.6**	-	-	-	100.0
	handle-press	78.4	81.0	**100.0**	76.8	82.8	-	-	-	83.4
	button-press	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	-	-	-	100.0

Chapter 8

EXPLOITING IMPLICIT STRUCTURE IN ROBOTICS DATA

In many cases, structure is not given to the learning algorithm explicitly, but rather presented *implicitly* in data due to biases in data collection processes. For example, it can be hard to enumerate all the safety constraints for a robot in a complex environment, while it is very common during data collection to stop the robot when it is about to take dangerous actions. It turns out that such implicit structure in robotics data can sometimes be (unintentionally) leveraged by learning algorithms to give us surprising results.

In this chapter, we present a novel observation about the behavior of offline reinforcement learning (RL) algorithms: on many benchmark datasets, offline RL can produce well-performing and safe policies even when trained with "wrong" reward labels, such as those that are zero everywhere or are negatives of the true rewards. This phenomenon cannot be easily explained by offline RL's return maximization objective. Moreover, it gives offline RL a degree of robustness that is uncharacteristic of its online RL counterparts, which are known to be sensitive to reward design.

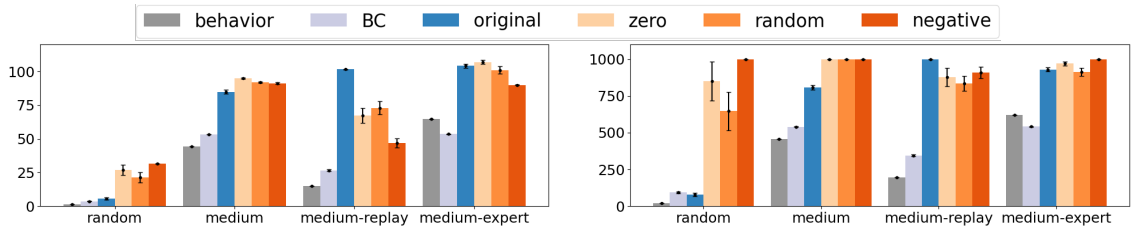
We demonstrate that this surprising robustness property is attributable to an interplay between the notion of *pessimism* in offline RL algorithms and certain implicit biases in common data collection practices. As we prove in this chapter, pessimism endows the agent with a *survival instinct*, i.e., an incentive to stay within the data support in the long term, while the limited and biased data coverage further constrains the set of survival policies. Formally, given a reward class — which may not even contain the true reward — we identify conditions on the training data distribution that enable offline RL to learn a near-optimal and safe policy from any reward within the class. We argue that the survival instinct should be taken into account when interpreting results from existing offline RL benchmarks and when creating future ones. Our empirical and theoretical results suggest a new paradigm for RL, whereby an agent is "nudged" to learn a desirable behavior with imperfect reward

but purposely biased data coverage.

8.1 A Surprising Finding

In offline reinforcement learning (RL), an agent optimizes its performance given an offline dataset. Despite being its main objective, we find that return maximization is not sufficient for explaining some of its empirical behaviors. In particular, *in many existing benchmark datasets, we observe that offline RL can produce surprisingly good policies even when trained on utterly wrong reward labels.*

In Fig. 8.1, we present such results on the `hopper` task from D4RL [Fu et al., 2020], a popular offline RL benchmark, using a state-of-the-art offline RL algorithm ATAC [Cheng et al., 2022]. The goal of an RL agent in the `hopper` task is to move forward as fast as possible while avoiding falling down. We trained ATAC agents on the original datasets and on three modified versions of each dataset, with "wrong" rewards: *1) zero*: assigning a zero reward to all transitions, *2) random*: labeling each transition with a reward sampled uniformly from $[0, 1]$, and *3) negative*: using the negation of the true reward. Although these wrong rewards contain no information about the underlying task or are even misleading, the policies learned from them in Fig. 8.1 (left) often perform significantly better than the behavior (data collection) policy and the behavior cloning (BC) policy. They even outperform policies trained with the true reward (denoted as *original* in Fig. 8.1) in some cases. This is puzzling, since RL is notorious for being sensitive to reward mis-specification [Leike et al., 2017, Hadfield-Menell et al., 2017, Ibarz et al., 2018, Pan et al., 2022]: in general, maximizing the wrong rewards with RL leads to sub-optimal performance that can be worse than simply performing BC. In addition, these wrong-reward policies demonstrate a "safe" behavior, which keeps the `hopper` from falling down for a longer period than other comparators in Fig. 8.1 (right). This is yet another peculiarity hard to link to return maximization, as none of the wrong rewards encourage the agent to stay alive. As we will show empirically in Section 8.4, these effects are not unique to ATAC or the `hopper` task. They occur with multiple offline RL algorithms, including ATAC [Cheng et al., 2022], PSPI [Xie et al., 2021], IQL [Kostrikov et al., 2021], CQL [Kumar et al., 2020a] and the Decision Transformer (DT) [Chen et al., 2021], on dozens of datasets from D4RL [Fu et al., 2020] and Meta-



(a) Normalized scores

(b) Episode lengths

Figure 8.1: On the `hopper` task from D4RL [Fu et al., 2020], ATAC [Cheng et al., 2022], an offline RL algorithm, can produce high-performance and safe policies even when trained on wrong rewards.

World [Yu et al., 2020a] benchmarks.

This robustness of offline RL is not only counter-intuitive but also cannot be explained fully by the literature. Offline RL theory Cheng et al. [2022], Xie et al. [2021], Jin et al. [2021], Liu et al. [2020] provides performance guarantees only when the data reward is the true reward. Although offline imitation learning (IL) Uehara and Sun [2022], Chang et al. [2021], Ma et al. [2022b] makes no assumptions about reward, it only shows that the learned policy can achieve performance comparable to that of the behavior policy, not beyond. Robust offline RL [Zhang et al., 2022, Wu et al., 2023, Zhou et al., 2021, Ma et al., 2022a, Panaganti et al., 2022] shows that specialized algorithms can perform well when the size of data perturbation is small. In contrast, we demonstrate that standard offline RL algorithms can produce good policies even when we completely change the reward. Constrained offline RL algorithms Le et al. [2019], Xu et al. [2022b], Lee et al. [2022a] can learn safe behaviors when constraint violations are both explicitly labeled and optimized. However, in the phenomena we observe, no safety signal is given to off-the-shelf, unconstrained offline RL algorithms. Recently, [Shin et al., 2023] observes a similar robustness phenomena of offline RL, but does not provide a complete explanation.

In this work, we provide an explanation for this seemingly surprising observation. In theory, we prove that this robustness property is attributed to the interplay between the

use of pessimism in offline RL algorithms and the implicit bias in typical data collection processes. Offline RL algorithms often use pessimism to avoid taking actions that lead to unknown future events. We show that this risk-averse tendency bakes a *"survival instinct"* into the agent, an incentive to stay within the data coverage in the long term. On the other hand, the limited coverage of offline data further constrains the set of *survival policies* (policies that remain in the data support in the long term). When this set of survival policies correlates with policies that achieve high returns w.r.t. the true reward (as in the example in Fig. 8.1), robust behavior emerges.

Our theoretical and empirical results have two important implications. First and foremost, offline RL has a survival instinct that leads to inherent robustness and safety properties that online RL does not have. Unlike online RL, offline RL is *doubly robust*: as long as the data reward is correct or the data has a positive implicit bias, a pessimistic offline RL agent can perform well. Moreover, offline RL is safe as long as the data only contains safe states; thus safe RL in the offline setup can be achieved without specialized algorithms. Second, because of the existence of the survival instinct, the data coverage has a profound impact on offline RL. While a large data coverage improves the best policy that can be learned by offline RL with the true reward, it can also make offline RL more sensitive to imperfect rewards. In other words, collecting a large set of diverse data might not be necessary or helpful (see Section 8.4). This goes against the common wisdom in the RL community that data should be as exploratory as possible [Munos, 2003, Chen and Jiang, 2019, Yarats et al., 2022].

We emphasize that survival instinct's implications should be taken into account when interpreting results on existing offline RL benchmarks as well as when designing future ones. We should treat the evaluation of offline RL algorithms differently from online RL algorithms. We suggest evaluating the performance of an offline RL algorithm by training it with wrong rewards in addition to the true reward so that we can isolate the performance due to return maximization from the compounded effects of survival instinct and implicit data bias. We propose to use this performance as a score to *quantify* the data bias (see Eq. (8.3) in Section 8.4.1) in practice.

We believe that our findings shed new light on RL applicability and research. To prac-

titioners, we demonstrate that offline RL does not always require the correct reward to succeed. This opens up the possibility of using offline RL in domains where obtain high-quality rewards is challenging. Research-wise, the existence of the survival instinct raises the question of how to design data collection or data filtering procedures that would help offline RL to leverage this instinct in order to improve RL's performance with incorrect or missing reward labels. While in this chapter we focus on positive data biases, we caution that in practice the data bias might be negatively correlated with a user's intention. In that case, running offline RL even with the right reward would not lead to the right behavior.

8.2 Background

The goal of offline RL is to solve an unknown Markov decision process (MDP) from offline data. Typically, an offline dataset is a collection of tuples, $\mathcal{D} \coloneqq \{(s, a, r, s')|(s, a) \sim \mu(\cdot, \cdot), r = r(s, a), s' \sim P(\cdot|s, a)\}$, where r is the reward, P captures the MDP's transition dynamics, and μ denotes the state-action data distribution induced by some data collection process. Modern offline RL algorithms adopt pessimism to address the issue of policy learning when μ does not have the full state-action space as its support. The basic idea is to optimize for a performance lower bound that penalizes actions leading to out-of-support future states. Such a penalty can take the form of behavior regularization Kostrikov et al. [2021], Fujimoto and Gu [2021], Wu et al. [2019], negative bonuses to discourage visiting less frequent state-action pairs Jin et al. [2021], Rashidinejad et al. [2021], Yu et al. [2020b], pruning less frequent actions [Liu et al., 2020, Kidambi et al., 2020], adversarial training [Cheng et al., 2022, Xie et al., 2021, Uehara and Sun, 2022, Xie et al., 2022, Rigter et al., 2022] or value penalties in modified dynamic programming [Kumar et al., 2020a, Yu et al., 2021].

We are interested in settings where the offline RL agent learns not from $\mathcal{D}$ itself but from its corrupted version $\tilde{\mathcal{D}}$ with a wrong reward $\tilde{R}$, i.e., $\tilde{\mathcal{D}} \coloneqq \{(s, a, \tilde{R}, s')|(s, a, s') \in \mathcal{D}, \tilde{R} = \tilde{R}(s, a) \in [-1, 1]\}$. We assume $\tilde{R}$ is from a reward class $\tilde{\mathcal{R}}$ — which may not necessarily contain the true reward R — and we wish to explain why offline RL can learn good behaviors from the corrupted dataset $\tilde{\mathcal{D}}$ (e.g., as in Fig. 8.1). Given any function $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a policy π, we define the state value function as $V_f^\pi(s) \coloneqq \mathbb{E}_{\pi,P}[\sum_{t=0}^{\infty} \gamma^t f(s_t, a_t)|s_0 = s]$ and

the state-action value function as $Q_f^\pi(s,a) \coloneqq f(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V_f^\pi(s')]$.

Assumption We make the typical assumption in offline RL that the data distribution μ assigns positive probabilities to these state-actions visited by running the optimal policy $\pi^\star$ starting from d_0.

Assumption 8.2.1 (Single-policy Concentrability). We assume $\sup_{s\in\mathcal{S},a\in\mathcal{A}} \frac{d^{\pi^\star}(s,a)}{\mu(s,a)} < \infty$.

This is a standard assumption in the offline RL literature, which in the worst case is a necessary condition to have no regret w.r.t. $\pi^\star$ [Zhan et al., 2022]. There are generalized notions [Xie et al., 2021, 2022] of this kind, which are weaker but requires other smoothness assumptions. We note that Assumption 8.2.1 does not assume that μ is the average state-action visitation frequency of a single behavior policy, nor does it assume μ has a full coverage of all states and actions or all policy distributions [Chen and Jiang, 2019, Foster et al., 2022].

8.3 Why Offline RL can Learn Right Behaviors from Wrong Rewards

In this section, we provide conditions under which offline RL's aforementioned robustness w.r.t. misspecified rewards emerges. Our main finding is summarized in the theorem below.

Theorem 8.3.1. *(Informal) Under Assumption 8.2.1 and certain regularity assumptions, if an offline RL algorithm Algo is set to be sufficiently pessimistic* and *the data distribution μ has a positive bias, for any data reward $\tilde{R} \in \tilde{\mathcal{R}}$, the policy $\hat{\pi}$ learned by Algo from the dataset $\tilde{\mathcal{D}}$ has performance guarantee $V_{\tilde{R}}^{\pi^\star}(d_0) - V_{\tilde{R}}^{\hat{\pi}}(d_0) \le O(\iota)$ as well as safety guarantee $(1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathrm{Prob}\left(\exists \tau \in [0,t], s_\tau \notin supp(\mu) | \hat{\pi}\right) \le O(\iota)$ for a small ι that decreases to zero as the degree of pessimism and dataset size increase.*

In other words, the robustness originates from an *interplay* between pessimism and an implicit positive bias in data. Here is the insight on why Theorem 8.3.1 is true. Because of pessimism, offline RL endows agents with a "survival instinct" — it implicitly solves a constrained MDP (CMDP) problem [Altman, 1999] that enforces the policy to stay within the data support. When combined with a training data distribution that has a positive

bias (e.g., all policies staying within data support are near-optimal) such a survival instinct results in robustness to reward misspecification.

Overall, Theorem 8.3.1 has two important implications:

1. Offline RL is *doubly robust*: it can learn near optimal policies so long as either the reward label is correct or the data distribution is positively biased;
2. Offline RL is *intrinsically safe* when data are safe, regardless of reward labeling, without the need of explicitly modeling safety constraints.

In the remaining of this section, we provide details and discussion of the statements above. We refer readers to [Li et al., 2023b] for complete theoretical statements and proofs.

8.3.1 Intuitions for Survival Instinct and Positive Data Bias

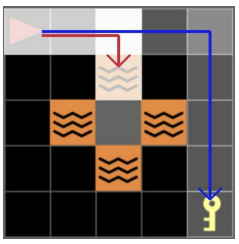


Figure 8.2: A grid world. BC (red); offline RL with wrong rewards (blue). The opacity indicates the frequency of a state in the data (more opaque means more frequent). Offline RL with the three wrong rewards produces the same policy.

We first use a grid world example to build some intuitions. Fig. 8.2 shows a goal-directed problem, where the true reward is +1 and -1 upon touching the key and the lava, respectively. The offline data is suboptimal and does not have full support. All data trajectories that touched the lava were stopped early, while others were allowed to continue until the end of the episode. The goal state (key) is an absorbing state, where the agent can stay *forever* beyond the episode length. We use the wrong rewards in Fig. 8.1 to train PEVI Jin et al. [2021], a finite-horizon, tabular offline RL method. PEVI performs dynamic programming similar to value iteration, but with a pessimistic value initialization and an instantaneous

pessimism-inducing penalty of $O(^{-1}/\sqrt{n(s,a)})$, where $n(s,a)$ is the empirical count in data. The penalties ensure that the learned value lower bounds the true one.

We see the PEVI agent learned with any wrong reward in Fig. 8.1 is able to solve the problem despite data imperfection, while the BC agent that mimics the data directly fails. The main reasons are: *1)* There is a *data bias* whereby longer trajectories end closer to the goal (especially, the longest trajectories are the ones that reach the goal, since the goal is an absorbing state). We call this a *length bias* (see Section 8.3.3). This positive data bias is due to bad trajectories (touching the lava or not reaching the goal) being cut short or timing out. *2)* Pessimism in PEVI gives the agent an *algorithmic bias* (i.e., survival instinct) that favors longer data trajectories. Because of the pessimistic value initialization, PEVI treats trajectories shorter than the episode length as having the lowest return. As a result, by maximizing the pessimistically estimated values, PEVI learns good behaviors despite wrong rewards by leveraging the survival instinct and positive data bias *together*. We now make this claim more general.

8.3.2 Survival Instinct

Survival instinct is a pessimism induced behavior that offline RL algorithms tend to favor policies leading to *in-support trajectories*. We formally characterize this risk aversion behavior by the concept of constrained MDP (CMDP) [Altman, 1999], which we define below.

Definition 8.3.1. Let $f, g : \mathcal{S} \times \mathcal{A} \to [-1, 1]$. A *CMDP* $\mathcal{C}(\mathcal{S}, \mathcal{A}, f, g, P, \gamma)$ is a constrained optimization problem: $\max_\pi V_f^\pi(d_0)$ s.t. $V_g^\pi(d_0) \leq 0$. Let $\pi^\dagger$ denote its optimal policy. For $\delta \geq 0$, we define the set of *δ-approximately optimal policies* $\Pi_{f,g}^\dagger(\delta) \coloneqq \{\pi : V_f^{\pi^\dagger}(d_0) - V_f^\pi(d_0) \leq \delta, V_g^\pi(d_0) \leq \delta\}$.

We prove that when trained on $\tilde{D}$, offline RL, because of its pessimism, implicitly solves the CMDP below even when the algorithm does not explicitly model any constraints:

$$\mathcal{C}_\mu(\tilde{R}) \coloneqq \mathcal{C}(\mathcal{S}, \mathcal{A}, \tilde{R}, c_\mu, P, \gamma), \tag{8.1}$$

where $c_\mu(s,a) \coloneqq \mathbb{1}[\mu(s,a)=0]$ indicates whether (s,a) is out of the support of μ[1] i.e., the constraint in Eq. (8.1) enforces the agent's trajectories to stay within the data support. Note the constraint in this CMDP is feasible because of Assumption 8.2.1.

Proposition 8.3.1 (Survival Instinct)**.** *(Informal) Under certain regularity conditions on $\mathcal{C}_\mu(\tilde{R})$, the policy learned by an offline RL algorithm Algo with the offline dataset $\tilde{D}$ is ι-approximately optimal with respect to $\mathcal{C}_\mu(\tilde{R})$, for some small ι which decreases as the algorithm becomes more pessimistic.*

Proposition 8.3.1 says if an offline RL algorithm is sufficiently pessimistic, then the learned policy is approximately optimal to $\mathcal{C}_\mu(\tilde{R})$. The learned policy has not only small regret with respect to the data reward $\tilde{R}$, but also small chances of escaping the support of the data distribution μ.

We highlight that the survival instinct in Proposition 8.3.1 is a *long-term* behavior. Such a survival behavior cannot be achieved by myopically taking actions in the data support in general (such as BC).[2] For instance, when some trajectories generating the data are truncated (e.g., due to early-stopping or intervention for safety reasons), taking in-support actions may still lead to out-of-support states in the future, as the BC agent in Section 8.3.1.

Proof Sketch The key to prove Proposition 8.3.1 is to show that an offline RL algorithm by pessimism has small regret for not just $\tilde{R}$ but a set of reward functions consistent with $\tilde{R}$ on data but different outside of data, including the Lagrange reward of (8.1) (i.e., $\tilde{R}-\lambda c_\mu$, with a large enough $\lambda \geq 0$). We call this property admissibility. Many existing offline RL algorithms are admissible, including model-free algorithms ATAC Cheng et al. [2022], VI-LCB Rashidinejad et al. [2021] PPI/PQI Liu et al. [2020], PSPI Xie et al. [2021], as well as model-algorithms, ARMOR Xie et al. [2022], Bhardwaj et al. [2023], MOPO Yu et al. [2020b], MOReL Kidambi et al. [2020], and CPPO Uehara and Sun [2022]. See [Li et al., 2023b] for formal proofs. By Proposition 8.3.1 these algorithms have survival instinct.

[1]The use of an indicator function is not crucial here. We can extend the current analysis here to other costs that are zero in the support and strictly positive out of the support.

[2]BC requires a stronger condition, e.g., the data are all complete trajectories without intervention or timeouts.

8.3.3 Positive Data Bias

In addition to survival instinct, another key factor is an implicit positive bias in common offline datasets. Typically these data manifest meaningful behaviors. For example, in collecting data for goal-oriented problems, data recording is normally stopped if the agent fails to reach the goal within certain time limit. Another example is problems (like robotics or healthcare) where wrong decisions can have detrimental consequences, i.e., problems that safe RL studies. In these domains, the data are collected by qualified policies only or under an intervention mechanism to prevent catastrophic failures. Such a one-sided bias can creates an effect that staying within data support would lead to meaningful behaviors. Below we formally define the positive data bias; Later in Section 8.4 (Fig. 8.6), we provide empirical estimates of the degree of positive data bias.

Definition 8.3.2 (Positive Data Bias)**.** A distribution μ is $\frac{1}{\epsilon}$-*positively biased* w.r.t. a reward class $\tilde{\mathcal{R}}$ if

$$\max_{\tilde{R}\in\tilde{\mathcal{R}}} \max_{\pi\in\Pi^{\dagger}_{\tilde{R},c_\mu}(\delta)} V_R^{\pi^\star}(d_0) - V_R^{\pi}(d_0) \leq \epsilon + O(\delta) \tag{8.2}$$

for all $\delta \geq 0$, where $\Pi^{\dagger}_{\tilde{R},c_\mu}(\delta)$ denotes the set of δ-approximately optimal policy of $\mathcal{C}_\mu(\tilde{R})$.

We measure the degree of positive data bias based on how bad a policy can perform in terms of the true reward r when approximately solving the CMDP $\mathcal{C}_\mu(\tilde{R})$ in (8.1) defined with the wrong reward $\tilde{R} \in \tilde{\mathcal{R}}$. If the data distribution μ is ∞-positively biased, then any approximately optimal policy to $\mathcal{C}_\mu(\tilde{R})$ (which includes the policies learned by offline RL, as shown earlier) can achieve high return in the true reward r. We also can view the degree of positiveness as reflecting whether $\tilde{R}$ provides a similar ranking as r, *among policies within the support of* μ. When there is a positive bias, offline RL can learn with $\tilde{R}$ to perform well under r, even when $\tilde{R}$ is not aligned with r globally. This is in contrast to online RL, which requires global alignment due to the exploratory nature of online RL.

Examples We provide a few examples of positive data bias.

- The distribution d^π induced by any policy π is ∞-positively biased for any rewards resulting from potential-based reward shaping [Ng et al., 1999] since it provides the same

ranking for all policies.

- For an IL setup, the data distribution μ is ∞-positively biased for any rewards $\tilde{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ if the data is generated by the optimal policy (i.e., $\mu = d^{\pi^\star}$). This is because following the optimal policy is the only way to stay within the data support. In Section 8.4, we empirically show that offline RL algorithms can learn good policies with wrong rewards on D4RL `expert` datasets, which are collected by near-optimal policies.

- A positive bias happens when longer trajectories in the data have smaller optimality gap. This is a generalized formal definition of the *length bias* mentioned in Section 8.3.1. This condition is typically satisfied when intervention is taken in the data collection process (despite the data collection policy being suboptimal), as in the motivating example in Fig. 8.1. Later in Section 8.4, we will investigate deeper into this kind of length bias empirically.

Remark We highlight that the positive data bias assumption in Definition 8.3.2 is *different* from assuming that the data is collected by expert policies, which is typical in the IL literature. Positive data bias assumption can hold in cases when data is generated by highly suboptimal policies, which we observe in the `hopper` example from Section 8.1. On the other hand, there are also cases where IL can learn well, while positive data bias does not exist (e.g., when learning from data collected by a stochastic expert policy that covers highly suboptimal actions with small probabilities).

8.3.4 Summary: Offline RL is Doubly Robust and Intrinsically Safe

We have discussed the survival instinct from pessimism and the potential positive data bias in common datasets. In short, survival instinct enables offline RL algorithms to learn policies that benefit from a favorable inductive bias of staying within the support of a positive data distribution. As a result, offline RL becomes robust to reward mis-specification (namely, Theorem 8.3.1).

Below we discuss some direct implications of Theorem 8.3.1. First, we can view this phenomenon as a doubly robust property of offline RL. We borrow the name "doubly robust"

from the offline policy *evaluation* literature [Dudík et al., 2011] to highlight the robustness of offline RL to reward mis-specification as an offline policy *optimization* approach.

Corollary 8.3.1. *Under Assumption 8.2.1, offline RL can learn a near optimal policy, as long as the reward is correct,* or *the data has a positive implicit bias.*

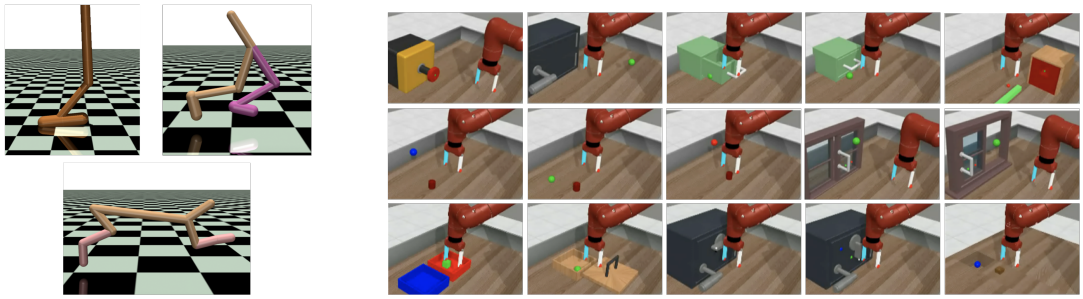
Theorem 8.3.1 also implies that offline RL is an intrinsically safe learning algorithm, unlike its counterpart online RL where additional penalties or constraints need to be explicitly modelled [Achiam et al., 2017, Wachi and Sui, 2020, Wagener et al., 2021, Paternain et al., 2022, Nguyen and Cheng, 2023].

Corollary 8.3.2. *If μ only covers safe states and there exists a safe policy staying within the support of μ, then the policy of offline RL only visits safe states with high probability (see Theorem 8.3.1).*

We should note that covering safe states is a very mild assumption in the safe RL literature Wagener et al. [2021], Hans et al. [2008], e.g., compared with all (data) states have a safe action. It does not require the data collection policies that generate μ are safe. This condition can be easily satisfied by filtering out unsafe states in post processing. The existence of a safe policy is also mild and common assumption. In Section 8.4.2, we validate this inherent safety property on offline SafetyGymnasium Liu et al. [2023a], an offline safe RL benchmark.

8.4 Experiments

We conduct two group of experiments. In Section 8.4.1, we conduct large scale experiment, showing that multiple offline RL algorithms can be robust to reward mis-specification on a variety of datasets. In Section 8.4.2, we experimentally validate the inherent safety of offline RL algorithms stated in Corollary 8.3.2. We show that admissible offline RL algorithms, without modifications, can achieve state-of-the-art performance on an offline safe RL benchmark Liu et al. [2023a].



(a) D4RL [Fu et al., 2020]

(b) Meta-World [Yu et al., 2020a]

Figure 8.3: We study offline RL with wrong rewards for a variety of tasks: (a) three locomotion tasks from D4RL [Fu et al., 2020]; hopper (top left), walker2d (top right) and halfcheetah (bottom). the figures are from Yu et al. [2019] (b) 15 goal-oriented tasks from Meta-World [Yu et al., 2020a]. The 15 tasks are composed of the 10 tasks from MT-10 (top 2 rows) and the 5 testing tasks from ML-45 (bottom row).

8.4.1 Robustness to Mis-specified Rewards

We empirically study the performance of offline RL algorithms under wrong rewards in a variety of tasks. We use the same set of wrong rewards as in Fig. 8.1. *1)* zero: the zero reward, *2)* random: labeling each transition with a reward value randomly sampled from Unif$[0, 1]$, and *3)* negative: the negation of true reward. We consider five offline RL algorithms, ATAC [Cheng et al., 2022], PSPI [Xie et al., 2021], IQL [Kostrikov et al., 2021], CQL [Kumar et al., 2020a] and decision transformer (DT)[3] [Chen et al., 2021]. We deliberately choose offline RL algorithms to cover those that are provably pessimistic [Cheng et al., 2022, Xie et al., 2021] and those that are popular among practitioners [Kostrikov et al., 2021, Kumar et al., 2020a], as well as an unconventional offline RL algorithm [Chen et al., 2021]. We consider a variety of tasks from D4RL [Fu et al., 2020] and Meta-World [Yu et al., 2020a] ranging from safety-critical tasks (i.e., the agent dies when reaching bad states), goal-oriented tasks, and tasks that belong to neither. We train a total of around 16k offline RL

[3]We condition the decision transformer on the return of a trajectory sampled randomly from trajectories that achieve top 10% of returns in terms of data reward.

agents.

Messages We would like to convey three main messages through our experiments. First, implicit data bias *can* exist naturally in a wide range of datasets, and offline RL algorithms that are sufficiently pessimistic can leverage such a bias to succeed when given wrong rewards. Second, offline RL algorithms, regardless of how pessimistic they are, become sensitive to reward when the data does not possess a positive bias. Third, offline RL algorithms without explicit pessimism, e.g., IQL [Kostrikov et al., 2021], can sometimes still be pessimistic enough to achieve good performance under wrong rewards.

Remark on negative results We consider "negative" results, i.e., when offline RL *fails* under wrong rewards, as important as the positive ones. Since they tell us *how* positive data bias can be broken or avoided. We hope our study can provide insights to researchers who are interested in actively incorporating positive bias in data collection, as well as who hope to design offline RL benchmarks specifically with or without positive data bias.

Locomotion Tasks from D4RL

We evaluate offline RL algorithms on three locomotion tasks, `hopper`, `walker2d`[4] and `halfcheetah`, from D4RL [Fu et al., 2020], a widely used offline RL benchmark. For each task, we consider five datasets with different qualites: `random`, `medium`, `medium-replay`, `medium-expert`, and `expert`. We refer readers to Fu et al. [2020] for the construction of these datasets. We measure policy performance in D4RL normalized scores [Fu et al., 2020]. We provide three baselines 1) behavior: normalized score directly computed from the dataset, 2) BC: the behavior cloning policy, and 3) original: the policy produced by the offline RL algorithm using the original dataset (with true reward). The normalized scores for baselines and offline RL algorithms with wrong rewards are presented in Fig. 8.4.

Positive bias exists in some D4RL datasets. We visualize the length bias of D4RL datasets in Fig. 8.5. Here are our main observations. *1)* Most datasets for `hopper` and

[4]We remove terminal transitions from `hopper` and `walker2d` datasets when learning with wrong rewards. See [Li et al., 2023a] for a ablation study on the effect of removing terminals when using true reward.

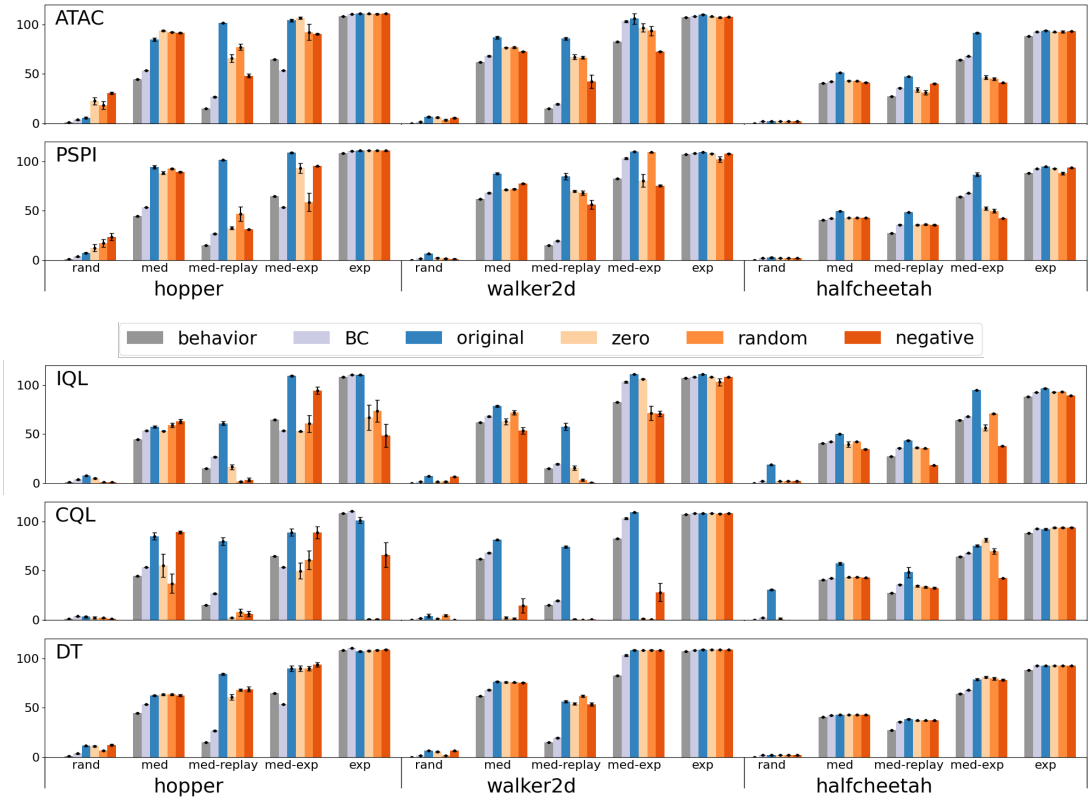


Figure 8.4: Normalized scores for locomotion tasks from D4RL [Fu et al., 2020]. The mean and standard error for normalized scores are computed across 10 random seeds. For each random seed, we evaluate the final policy of each algorithm over 50 episodes.

`walker2d`, with the exception of the medium-replay datasets, have a strong length bias, where longer trajectories have higher returns. This length bias is due to the safety-critical nature of `hopper` and `walker2d` tasks, as the trajectories get terminated when reaching bad states. The length bias is especially salient in `hopper-medium`, where the normalized score is almost proportional to episode length. *2)* The medium-replay datasets for `hopper` and `walker2d` have more diverse behavior, so the bias is smaller. *3)* All `halfcheetah` datasets do not have length bias, as they all have the same length of 1000. *4)* `hopper-expert` dataset has a length bias, while `walker2d` and `halfcheetah-expert` datasets do not have an obvious length bias. However, it is worth noting that all `expert` datasets have an IL-type

positive bias as discussed in Section 8.3.3.

Offline RL can learn good policies with wrong rewards on datasets with strong length bias. On datasets with strong length bias (`hopper-random`, `hopper-medium` and `walker2d-medium`), we observe that ATAC and PSPI with wrong rewards generally produce well-performing policies, in a few cases even out-performing the policies learned from the true reward (original in Fig. 8.5). DT is mostly insensitive to reward quality. IQL and CQL with wrong rewards can sometimes achieve good performance; among the two, we find IQL to be more robust to wrong rewards and CQL with wrong rewards almost fails completely on all `walker2d` datasets.

Offline RL can learn good policies with wrong rewards on `expert` datasets In Section 8.3, we point out that the data has a positive bias when it is generated by the optimal policy. In Fig. 8.4, we observe that all offline RL algorithms, when trained with wrong rewards, can achieve expert-level performance on `walker2d` and `halfcheetah-expert` datasets. ATAC, PSPI and DT perform well on the `hopper-expert` dataset while IQL and CQL receive lower scores when using wrong rewards.

Offline RL needs stronger reward signals on datasets with diverse behavior policy. The medium-replay datasets of `hopper` and `walker2d` are generated from multiple behavior policies. Due to their diverse nature, they are a multiple ways to stay within data support in a long term. As a result, the survival instinct of offline RL by itself is not sufficient to guarantee good performance. Here algorithms with wrong rewards generally under-perform the policies trained with true reward. As datasets have a more diverse coverage, they can be less positively biased and offline RL requires a stronger reward signal to differentiate good survival policies and the bad ones. Practically speaking, when high-quality reward is not available, a diverse dataset can even hurt offline RL performance. This is contrary to the common belief that larger data support is more preferable [Munos, 2003, Chen and Jiang, 2019, Yarats et al., 2022].

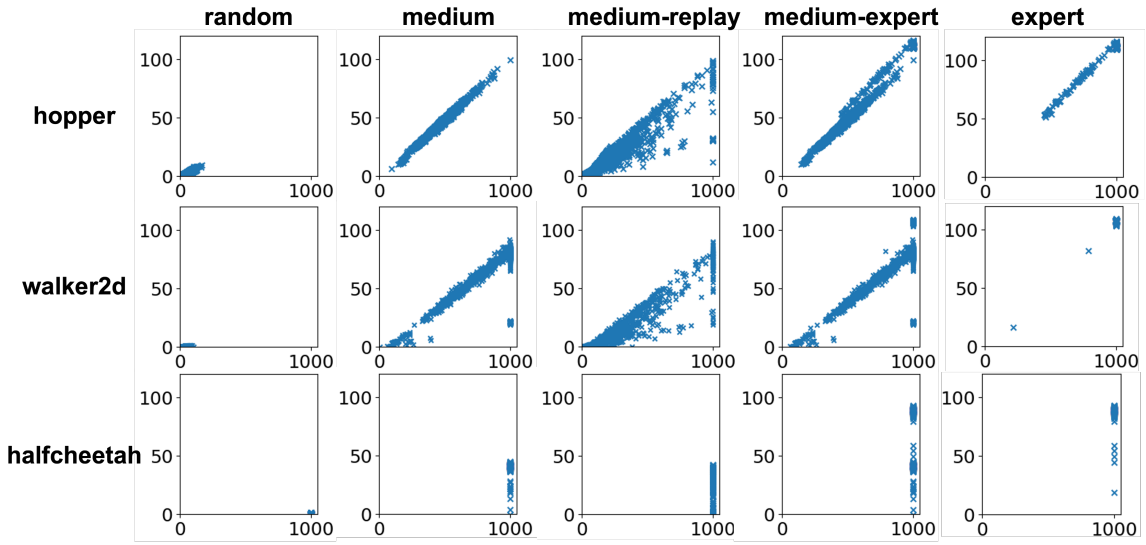


Figure 8.5: A visualization of length bias in datasets from D4RL [Fu et al., 2020]. Each plot corresponds to a dataset for a task (row) with a dataset (column). Each trajectory in a dataset is represented as a data point with the x-coordinate being its episode length and y-coordinate being its normalized score.

Offline RL requires good reward to perform well on datasets without length bias. In all four `halfcheetah` datasets, the trajectories all have the same length, as is demonstrated in Fig. 8.5. This means that there is no data bias. We observe that all algorithms with wrong rewards at best perform similarly as the behavior and BC policies in most cases; this is an imitation-like effect due to the survival instinct. In the `halfcheetah-medium-expert` dataset, since there are a variety of surviving trajectories, with score from 0 to near 100, we observe that the performance of the resulting policy degrades as data reward becomes more different from the true reward.

Offline RL algorithms with provable pessimism produce safer policies. For `hopper` and `walker2d`, we observe that policies learned by ATAC and PSPI, regardless of the reward, can keep the agent from falling for longer. The episode lengths of IQL and CQL policies are often comparable to that of the behavior policies. In Section 8.4.2, we provide

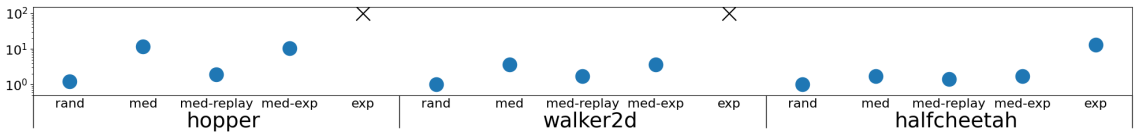


Figure 8.6: Estimated positive data bias of all 15 D4RL datasets given by ATAC. Datasets marked by "×", i.e., `hopper-expert` and `walker2d-expert`, have infinite positive bias.

further results validating the safety properties of ATAC and PSPI.

On empirically estimating positive data bias Inspired by Definition 8.3.2, we propose an empirical estimate of positive data bias. Let J_{zero}, J_{rand} and J_{neg} be the return (with respect to the true reward) of an offline RL algorithm learned with zero, random, and negative rewards, respectively. Then, given an estimated optimal return $\hat{J}^\star$, we propose to empirically estimate the positive data bias by

$$\text{estimated positive bias} = \frac{\hat{J}^\star}{\max\{\hat{J}^\star - \min\{J_{\text{zero}}, J_{\text{rand}}, J_{\text{neg}}\}, 0\}}. \tag{8.3}$$

In Fig. 8.6, we visualize the estimated positive data bias of all 15 D4RL datasets given by ATAC.

Remark on benchmarking offline RL Our observations on the popular D4RL datasets raise an interesting question for benchmarking offline RL algorithms. An implicit positive data bias can give certain algorithms a hidden advantage, as they can already achieve good performance without using the reward. Without controlling data bias, it is hard to differentiate whether the performance of those algorithms are due to their ability to maximize returns or simply due to their survival instinct.

Goal-oriented Manipulation Tasks from Meta-World

We study goal-oriented manipulation tasks from the Meta-World benchmark [Yu et al., 2020a]. We consider 15 goal-oriented tasks from Meta-World: the 10 tasks from the MT-10 set and the 5 testing tasks from the ML-45 set. For each task, we generate a dataset of 110

trajectories using the scripted policies provided by Yu et al. [2020a]; 10 are produced by the scripted policy, and 100 are generated by perturbing the scripted policy with a zero-mean Gaussian noise of standard deviation 0.5. In the datasets, unsuccessful trajectories receive a time-out signal after 500 steps (maximum episode length for Meta-World).

For each task, we measure the success rate of policies for 50 *new* goals unseen during training. Similar to D4RL experiments, we consider BC policies and policies trained with the true reward as baselines. Since the training dataset is generated for a different set of goals, we do not compare the success rate of learned policies with the success rate in data.

Goal-oriented problems have data bias. Goal-oriented problems are fundamentally different from safety-critical tasks (such as `hopper` and `walker2d`). A good goal-oriented policy should reach the goal as fast as possible rather than wandering around until the end of an episode. But another form of length bias still exists here, as successful trajectories are labeled with a termination signal that indicates an unbounded length and failed trajectories have a bounded length (see Section 8.3.1).

Offline RL can learn with wrong rewards on goal-oriented tasks. The success rate of learned policies with different rewards are shown in Fig. 8.7. We observe that ATAC and PSPI with wrong rewards generally achieve comparable or better success rate than BC policies. The exceptions are often due to that ATAC or PSPI, even with the true reward, does not work as well as BC in those tasks, e.g. drawer-open, bin-picking and box-close. This could be caused by overfitting or optimization errors (due to challenges of dynamics programming over long horizon problems). In a number of tasks, such as peg-insert-side, push and reach, ATAC and PSPI with wrong rewards can out-perform BC by a margin. This shows that data collection for goal-oriented problems have a positive bias that offline RL algorithms can succeed without true reward. This is remarkable as when learning with random and negative rewards, unsuccessful trajectories are long and generally have significantly higher return, as reward for each step is non-negative. The offline RL algorithms need to be sufficiently pessimistic and be able to plan over a long horizon to propagate pessimism to the unsuccessful data trajectories. For IQL, there is a

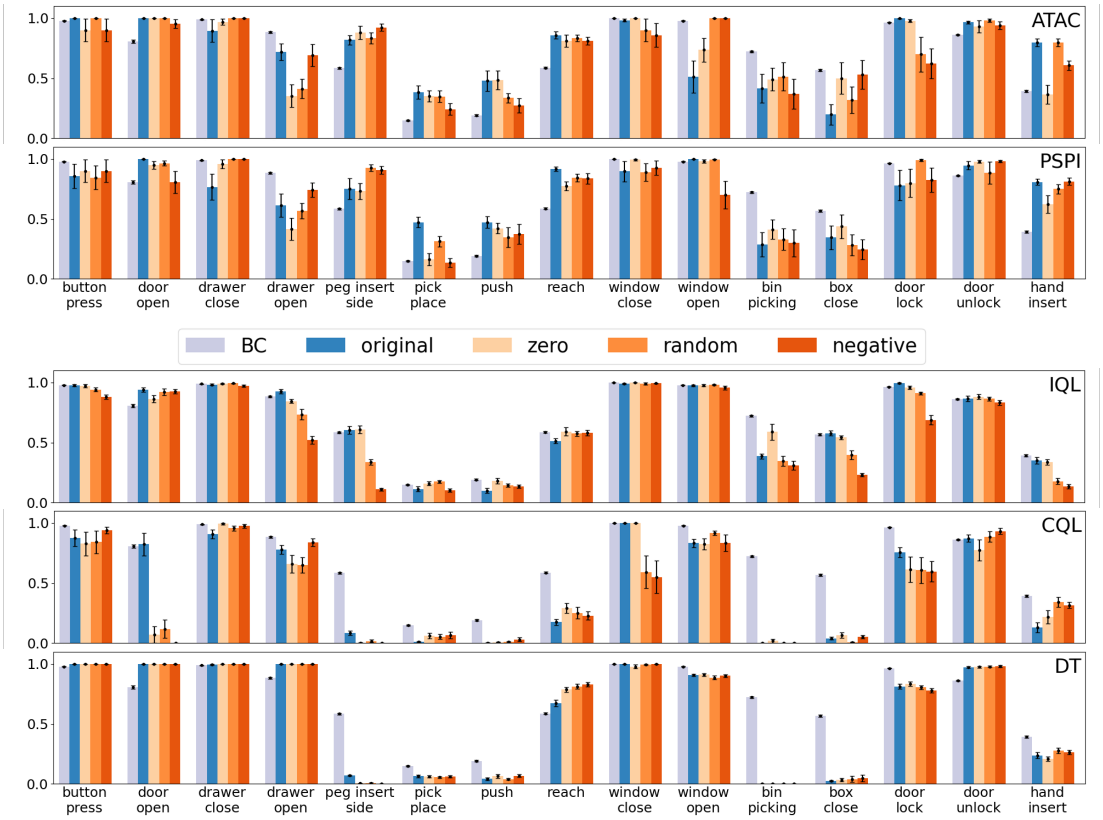


Figure 8.7: Success rate for goal-oriented tasks from Meta-World [Yu et al., 2020a]. The average success rate and confidence interval are computed across 10 random seeds. For each seed, we evaluate final policies for 50 episodes, each with a new goal unseen in the dataset.

gentle performance decrease as the reward becomes more different than the true reward, even though policies learned with wrong rewards are not much worse than BC in many cases. CQL shows robustness to reward in a few tasks such as button-press, drawer-close and door-unlock. Interestingly, DT performs almost uniformly across all rewards, potentially because DT does not explicitly maximize returns.

Remark on BC performance It is noticeable that BC policies in general achieve high performance, in many tasks often out-performing offline RL policies learned with true reward. This effect has also been observed in existing work on similar tasks Zhou et al. [2022].

We would like to clarify that the goal of our experiments is to study the behavior of offline RL algorithms when trained with wrong rewards, rather than showing that offline RL performs better than BC. We refer interested readers to existing studies Zhou et al. [2022], Kumar et al. [2022] on when using offline RL algorithms is or is not preferable over BC.

8.4.2 Inherent Safety Properties of Offline RL Algorithms

We show offline RL algorithms (without any modifications) can behave as a safe RL algorithm. We conduct experiments on offline SafetyGymnasium Liu et al. [2023a] using ATAC Cheng et al. [2022] and PSPI Xie et al. [2021]. We first remove all transitions with non-zero cost and then run offline RL algorithms on the filtered datasets.

Task	ATAC		PSPI		Best None-BC Algorithm from Liu et al. [2023a]		
	reward ↑	cost ↓	reward ↑	cost ↓	reward ↑	cost ↓	algorithm
PointButton1	0.08	0.82	0.10	1.04	0.13	1.35	COpiDICE Lee et al. [2022a]
PointButton2	0.14	0.89	0.19	1.29	0.15	1.51	COpiDICE Lee et al. [2022a]
PointCircle1	0.40	2.79	0.34	2.39	0.59	0.69	CDT Liu et al. [2023b]
PointCircle2	0.61	4.45	0.55	2.84	0.64	1.05	CDT Liu et al. [2023b]
PointGoal1	0.58	0.94	0.59	0.90	0.71	0.98	BCQ-Lag Xu et al. [2022b]
PointGoal2	0.55	2.30	0.45	2.55	0.40	1.31	CPQ Xu et al. [2022b]
PointPush1	0.18	0.51	0.20	0.85	0.33	0.86	BCQ-Lag Xu et al. [2022b]
PointPush2	0.10	0.68	0.10	0.81	0.23	0.99	BCQ-Lag Xu et al. [2022b]
CarButton1	-0.06	1.25	-0.08	1.11	0.21	1.60	CDT Liu et al. [2023b]
CarButton2	-0.11	1.22	-0.08	1.04	0.13	1.58	CDT Liu et al. [2023b]
CarCircle1	0.66	5.51	0.64	5.92	0.60	1.73	CDT Liu et al. [2023b]
CarCircle2	0.64	5.96	0.64	5.99	0.66	2.53	CDT Liu et al. [2023b]
CarGoal1	0.50	0.99	0.41	0.78	0.47	0.78	BCQ-Lag Xu et al. [2022b]
CarGoal2	0.24	1.00	0.24	1.05	0.25	0.91	COptiDICE Lee et al. [2022a]
CarPush1	0.33	0.96	0.32	0.75	0.31	0.40	CDT Liu et al. [2023b]
CarPush2	0.10	0.95	0.11	0.81	0.09	1.07	COptiDICE Lee et al. [2022a]

Table 8.1: Results on `point` and `car` datasets from offline SafetyGymnasium Liu et al. [2023a] over 3 random seeds. Cumulative reward and cost are normalized as described in Liu et al. [2023a]. An agent is considered safe if the cumulative cost is no larger than 1. Unsafe agent with cumulative cost more than 1 (total cost 34.29) is shown in gray.

The results are summarized in Table 8.1. We observe that *offline RL with this naïve data filtering strategy (using less data) can achieve comparable performance as the per-task best performing state-of-the-art offline safe RL algorithm*, which uses the full dataset and has knowledge of the cost target.[5]. Our agents incur low cost except for the circle tasks. We hypothesize that learning a safe policy from the circle datasets is hard, as other baselines also struggle to produce safe policies. Note that these are preliminary results, and better performance might be achievable through using a more sophisticated data filtering strategy.

8.5 Concluding Remarks

We present unexpected results on the robustness of offline RL to reward mis-specification. We show that this property originates from the interaction between the *survival instinct* of offline RL and hidden positive biases in common data collection processes. Our findings suggest extra considerations should be taken when interpreting offline RL results and designing future benchmarks. In addition, our findings open a new space for offline RL research: the possibility of designing algorithms that proactively leverage the survival instinct to learn policies for domains where rewards are nontrivial to specify or unavailable.

8.6 Related Work

Offline RL Offline RL studies the problem of return maximization given an offline dataset. Offline RL algorithms can be broadly classified into model-based approaches, e.g., [Yu et al., 2020b, Kidambi et al., 2020, Uehara and Sun, 2022, Rigter et al., 2022, Xie et al., 2022], and model-free approaches, e.g., [Jin et al., 2021, Fujimoto and Gu, 2021, Xie et al., 2021, Cheng et al., 2022, Kumar et al., 2020a, Kostrikov et al., 2021]. Since the agent needs to learn without online data collection, offline RL becomes more challenging when the offline dataset does not provide good coverage over the state-action distribution of all feasible policies. There have been two common strategies to handle insufficient coverage,

[5]Many algorithms benchmarked in Liu et al. [2023a] use a cost target, which sets the maximum total cost allowed by safety. Liu et al. [2023a] uses three different cost targets, $\{20, 40, 80\}$, and reports the normalized total reward and cost for each algorithm averaged over the three targets. By contrast, standard offline RL algorithms do not use such a cost target. We report results of offline RL algorithms according to an "effective cost target" of 34.29, which is similar to how BC_All results are presented in [Liu et al., 2023a].

behavior regularization approaches, e.g., [Fujimoto and Gu, 2021, Wu et al., 2019, Kostrikov et al., 2021], which restricts the learned policy to be close to the behavior policy, and value regularization approaches, e.g., [Jin et al., 2021, Liu et al., 2020, Xie et al., 2021, Cheng et al., 2022, Kumar et al., 2020a, Xie et al., 2022] which provide a pessimistic value estimates for the learned policy. Both approaches can be viewed as optimizing for a performance lower bound of the agent's return.

Offline RL commonly assumes compatibility between offline dataset and MDP, i.e., the offline dataset should be generated from the task MDP of interest. Our work is distinctive in the offline RL literature as we study the behavior of existing offline RL algorithms when data reward *differs* from the true MDP reward. Existing tools for analyzing offline RL are not directly applicable to our setting: policies with good performance under data reward, as guaranteed by such tools, do not necessarily attain high return under the true reward. Our novel analysis is made possible by studying the properties of offline data distribution, an aspect mostly neglected by existing work in the literature.

Offline RL with imperfect reward In the offline RL literature, there is a line of work studying scenarios where the reward is missing in a subset of the dataset [Singh et al., 2020, Hu et al., 2023, Li et al., 2023a, Yu et al., 2022] or when the data reward is misspecified [Li et al., 2023c]. These approaches propose to construct a new reward function and apply offline RL algorithms on the relabeled dataset. For example, Singh et al. [2020], Yu et al. [2022] label the missing reward with a constant minimum reward. Li et al. [2023a], Hu et al. [2023] learn a pessimistic reward function. Li et al. [2023c] optimize for a reward function which minimizes the reward difference among expert data.

Different from this line of work, we are interested in running offline RL algorithms using the misspecified reward *as it is*. We show both theoretically and empirically that offline RL algorithms can produce well-performing policies with rewards different from the true reward as long as the underlying data distribution satisfies certain conditions. It can be an interesting future work to combine these techniques with our analysis to explore the robustness of offline RL algorithms under the class of learned reward functions.

Recently, Shin et al. [2023] observed a similar robustness phenomena of offline RL.

However, the authors did not provide a complete explanation. They found that offline RL algorithms such as AWR [Peng et al., 2019] empirically can learn well with random or zero rewards on D4RL locomotion datasets. For the expert datasets, they conjectured that this robustness may be due to the fact that offline RL algorithms behave like imitation learning on expert data, but they did not provide details of why offline RL algorithms have this robustness behavior on other datasets. Here we provide a formal theoretical proof to show that this robustness is due to the interaction between the survival instinct of offline RL and an implicit positive data bias. Our results support Shin et al. [2023]'s conjecture on the expert dataset; we show that expert datasets generated by an optimal policy can theoretically be proved to be infinitely positively biased, and we also empirically estimate the positive bias on the D4RL datasets in Fig. 8.6, which agrees with the theoretical prediction. Our results further provide conditions for positive data bias beyond the expert data scenario, such as the length bias. These new insights show why offline RL algorithms can learn good policies with wrong rewards, on datasets collected by non-expert suboptimal policies. Finally, we present strong empirical evidence to show this robustness phenomenon happens in a large number of existing benchmarks with multiple offline RL algorithms.

Offline IL Offline imitation learning (IL) can be considered as a special case of offline RL when reward is missing from the entire dataset, or when the agent has the largest uncertainty about the reward [Cheng et al., 2022]. In offline IL, the learner is instead given the information on whether a transition is generated by an expert. The goal of offline IL is to produce a policy that has comparable performance with the expert. Zolna et al. [2020] learn a discriminator reward to classify expert and non-expert data, and apply offline RL algorithms with the learned reward. Xu et al. [2022a] use the discriminator as the weight for weighted behavior cloning. Kim et al. [2021], Ma et al. [2022b], Zhu et al. [2020] propose DICE [Dai et al., 2020]-style algorithms to minimize the divergence between the state-action or state-(next state) occupancy measure of the learner and the expert. Chang et al. [2021], Kidambi et al. [2021] minimize state-action or state-(next state) distribution divergence under a pessimistic model. Yue et al. [2023] extend maximum entropy inverse RL [Ziebart et al., 2008b] to an offline setting. Smith et al. [2023] propose to run offline RL under

an indicator reward function on whether the data is expert data. Li et al. [2023a] learn a pessimistic reward function where the expert data labeled as the maximum reward.

Our experiments with zero reward function is similar to an offline IL setting with the behavior policy being the expert. However, we show that offline RL algorithms can sometimes achieve significant performance gain over behavior and behavior cloning policies, which is a phenomenon that cannot be explained by existing analysis in offline IL. We provide an explanation to this puzzle: we show that the data distribution is ∞-positively biased (according our definition) for offline RL, when the data is generated by the optimal policy of the true MDP. This means that an admissible offline RL algorithm can achieve near-optimal performance with *any* reward function from its corresponding admissible reward class in this case. We empirically show that in Section 8.4.

Robust offline RL Robust offline RL [Zhang et al., 2022, Chen et al., 2023, Wu et al., 2023, Panaganti et al., 2022, Yang et al., 2022, Zhou et al., 2021, Ma et al., 2022a, Shi and Chi, 2022] aims to develop new offline RL algorithms for cases when the compatibility assumption does not hold, i.e., the offline dataset may not be generated from the task MDP. Among this line of work, Panaganti et al. [2022], Zhou et al. [2021], Ma et al. [2022a], Shi and Chi [2022] study the robust MDP setting, where the dataset can be generated from any MDP within a δ-ball of a nominal MDP. An adversary can choose any MDP from the δ-ball when evaluating the learned policy. Zhang et al. [2022] considers when the adversary can modify ϵ fraction of transition tuples in the offline dataset, while Wu et al. [2023] consider when up to K trajectories can be added in or removed from the original dataset. Chen et al. [2023] use the formulation of Byzantine-robust: when multiple offline datasets are presented to the learner, among them an α fraction are corrupted. Yang et al. [2022] assume the dataset is generated from the true MDP, but the state presented to the policy during test time can be a corrupted up to ϵ distance.

Our work is fundamentally different from this line of work in that we study the *inherent* and somewhat *unintended* robustness of offline RL algorithms. We show that, despite originally designed under the compatibility assumption, offline RL algorithms show strong robustness against perturbation of reward. Another salient difference is that robust offline

RL typically requires an upper bound on the size of perturbation to the true or nominal MDP, and most of proposed algorithms assume knowledge of this upper bound. However, our work does not make assumptions on the size of perturbation and, moreover, the algorithms we study are not even aware of the existence of such perturbation. Indeed, we empirically show that offline RL algorithms, without any modifications, can succeed even when data reward is a constant or the negation of the true reward. Additionally, in this work, we consider the reward perturbation while most work in robust offline RL focuses more on dynamics perturbation.

Constrained and safe offline RL Our work is related to the literature of constrained and safe offline RL [Le et al., 2019, Xu et al., 2022b, Liu et al., 2023b, Lee et al., 2022a, Polosky et al., 2022]. In constrained offline RL, the agent solves for a CMDP given an offline dataset $\{(s, a, r, c, s')\}$. The goal of constrained offline RL is to produce a policy which maximizes the expected return while ensuring that constraint violation is below a given upper bound. Le et al. [2019] uses an offline policy evaluation oracle to solve for the Lagrangian relaxation of the CMDP. Xu et al. [2022b] construct a pessimistic estimation of value and constraint violation. Lee et al. [2022a], Polosky et al. [2022] propose DICE [Dai et al., 2020]-style algorithms to solve for the optimal state-action occupancy measure. Liu et al. [2023b] propose a variant of decision transformer [Chen et al., 2021] capable of producing policies for different constraint violation upper bounds during test time.

Our work, instead, considers a different scenario where the constraint is only given *implicitly* — through the support of the offline dataset. We assume that any transition in the dataset satisfies the constraint, which can be achieved through intervention during data collection. We show that offline RL can inherently produce approximately safe policies due to its survival instinct. Although our assumption seems more restrictive as in that we require a constraint violation-free dataset, we argue that our assumption is more practical in many safety critical scenarios: compared to executing unsafe actions, it is perhaps more reasonable to intervene before constraint violation happens. We believe that our findings open up new possibilities for applying offline RL as a safe RL oracle, which can be especially promising in an offline-online RL setup [Xu et al., 2023, Lee et al., 2022b].

EPILOGUE

In this thesis, we see that by exploiting various types of structure, from the more explicit ones to the more hidden ones, we can build robots that can learn safely and efficiently. When we have known problem structure about the underlying task, we can bake it into a policy class, which allows the policy to have formal safety guarantees, learn efficiently, and generalize well. By reasoning about explicit structure in data, in terms of what information they provide, we propose a unifying formulation, and a new learning algorithm which both has theoretical guarantees and works well empirically. Finally, data can have implicit structure that comes from its collection process. We can leverage such implicit structure to achieve things that sound almost too good to be true. For example, we can learn good policies using offline RL without a reward, and we can learn safe policies without seeing any unsafe data. I would like to end this thesis with two open questions.

How much should we rely on problem structure? As robotics tasks become less predefined and more complex, and as robot becomes more dynamic, such problem structure may not be available or it may not accurately reflect the world. On the other hand, although we see a trend of relying more on data (and less on problem structure) in neighboring fields such as NLP and computer vision, there are some unique challenges that robot learning faces. One, it is possible that we are not going to get as much data as we need, at least in a short term. For NLP and computer vision, humans are basically generating data every single moment, and we can build web-scale datasets. However, this is not exactly the same in robotics, as robots are still not part of our everyday life. Therefore, we need to make specific effort to collect robotics data. Another challenge is that we need to build large, capable models for capturing information from data, but it also means that it takes more time when we query those models. This is a big problem in robot learning, as robotics systems typically need a reasonably high control rate when operating at high speed or in dynamic environment.

How to do data collection or curation for robot learning? A lot of observations in this thesis show that learning algorithms can benefit from structure. But there is a flip side to this observation, which is that learning algorithms can also be *misled* by structure in data. Sometimes, the structure is not even obvious. To make it even worse, when data has "bad" structure which misleads the learning algorithm, just providing it with more information does not necessarily help[6]. So there is a delicate interplay between the data collection or curation process and the learning algorithm. There have been more active research on learning algorithm alone. On the other, how to do data collection or data curation right is less well understood. This process is either treated as a black box, or designed based on "folklore knowledge", such as we should always collect expert data, or more data is always better. It will be interesting future work to explore in the space between data collection and learning algorithms for robot learning. For example, what is a scalable protocol to create "good" structure during data collection, how to effectively make use of existing data from different sources, and how to quantify how much "good" or "bad" structure is in data, etc.

[6]Remember the hopper example in Chapter 8. Even when we give the learning algorithm the negative reward, which penalizes the behavior of jumping forward and staying alive, offline RL algorithms refuse to optimize for this reward due to implicit bias in data.

BIBLIOGRAPHY

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *the twenty-first international conference on Machine learning*, 2004a.

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004b.

J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

E. Altman. *Constrained Markov decision processes*, volume 7. CRC press, 1999.

A. D. Ames, J. W. Grizzle, and P. Tabuada. Control barrier function based quadratic programs with application to adaptive cruise control. In *IEEE 53rd Annual Conference on Decision and Control*, pages 6271–6278. IEEE, 2014.

A. Antos, C. Szepesvári, and R. Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

P. Aumjaud, D. McAuliffe, F. J. Rodríguez-Lera, and P. Cardiff. Reinforcement learning experiments and benchmark for solving robotic reaching tasks. In *Workshop of Physical Agents*, pages 318–331. Springer, 2020.

L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

M. Bhardwaj, S. Choudhury, and B. Boots. Blending mpc & value function approximation for efficient reinforcement learning. In *International Conference on Learning Representations*, 2020.

M. Bhardwaj, T. Xie, B. Boots, N. Jiang, and C.-A. Cheng. Adversarial model for offline reinforcement learning. *arXiv preprint arXiv:2302.11048*, 2023.

S. P. Bhat and D. S. Bernstein. Finite-time stability of continuous autonomous systems. *SIAM Journal on Control and Optimization*, 38(3):751–766, 2000.

F. Bullo and A. D. Lewis. *Geometric control of mechanical systems: modeling, analysis, and design for simple mechanical control systems*, volume 49. Springer Science & Business Media, 2004.

Z. Cao and D. Sadigh. Learning from imperfect demonstrations from agents with varying dynamics. *IEEE Robotics and Automation Letters*, 6(3):5231–5238, 2021.

Z. Cao, Y. Hao, M. Li, and D. Sadigh. Learning feasibility to imitate demonstrators with different dynamics. In *Conference on Robot Learning*, pages 363–372. PMLR, 2021.

H. chaandar Ravichandar and A. Dani. Learning position and orientation dynamics from demonstrations via contraction analysis. *Autonomous Robots*, 43(4):897–912, 2019.

J. Chang, M. Uehara, D. Sreenivas, R. Kidambi, and W. Sun. Mitigating covariate shift in imitation learning via offline data with partial coverage. *Advances in Neural Information Processing Systems*, 34:965–979, 2021.

J. Chen and N. Jiang. Information-theoretic considerations in batch reinforcement learning. In *International Conference on Machine Learning*, pages 1042–1051. PMLR, 2019.

L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Y. Chen, X. Zhang, K. Zhang, M. Wang, and X. Zhu. Byzantine-robust online and offline distributed reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3230–3269. PMLR, 2023.

C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff. RMPflow: A computational graph for automatic motion policy generation. *Proceedings of the 13th Annual Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2018.

C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff. RMPflow: A geometric framework for generation of multi-task motion policies. *arXiv preprint arXiv:2007.14256*, 2020.

C.-A. Cheng, T. Xie, N. Jiang, and A. Agarwal. Adversarially trained actor critic for offline reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2022.

J. Choi, F. Castañeda, C. J. Tomlin, and K. Sreenath. Reinforcement learning for safety-critical control under model uncertainty, using control Lyapunov functions and control barrier functions. *arXiv preprint arXiv:2004.07584*, 2020.

H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019.

B. Christianson. Automatic hessians by reverse accumulation. *IMA Journal of Numerical Analysis*, 12(2):135–150, 1992.

E. Coumans and Y. Bai. PyBullet, a python module for physics simulation in robotics, games and machine learning, 2017.

B. Dai, O. Nachum, Y. Chow, L. Li, C. Szepesvári, and D. Schuurmans. Coindice: Off-policy confidence interval estimation. *Advances in neural information processing systems*, 33:9398–9411, 2020.

G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

S. Desai, I. Durugkar, H. Karnan, G. Warnell, J. Hanna, and P. Stone. An imitation from observation approach to transfer learning with dynamics mismatch. *Advances in Neural Information Processing Systems*, 33:3917–3929, 2020.

A. Dietrich, A. Albu-Schäffer, and G. Hirzinger. On continuous null space projections for torque-based, hierarchical, multi-objective manipulation. In *IEEE International Conference on Robotics and Automation*, pages 2978–2985. IEEE, 2012.

A. Dietrich, C. Ott, and J. Park. The hierarchical operational space formulation: stability analysis for the regulation case. *IEEE Robotics and Automation Letters*, 3(2):1120–1127, 2018.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

M. Dudík, J. Langford, and L. Li. Doubly robust policy evaluation and learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 1097–1104, 2011.

A. Edwards, H. Sahni, R. Liu, J. Hung, A. Jain, R. Wang, A. Ecoffet, T. Miconi, C. Isbell, and J. Yosinski. Estimating Q(s, s') with deep deterministic dynamics gradients. In *International Conference on Machine Learning*, pages 2825–2835. PMLR, 2020.

P. Englert, N. A. Vien, and M. Toussaint. Inverse kkt: Learning cost functions of manip-

ulation tasks from demonstrations. *The International Journal of Robotics Research*, 36 (13-14):1474–1488, 2017. doi: 10.1177/0278364917745980.

A. Escande, N. Mansard, and P.-B. Wieber. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028, 2014.

B. Eysenbach, S. Levine, and R. R. Salakhutdinov. Replacing rewards with examples: Example-based policy search via recursive classification. *Advances in Neural Information Processing Systems*, 34:11541–11552, 2021.

T. Flash and N. Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *Journal of neuroscience*, 5(7):1688–1703, 1985.

D. J. Foster, A. Krishnamurthy, D. Simchi-Levi, and Y. Xu. Offline reinforcement learning: Fundamental barriers for value function approximation. In *Conference on Learning Theory*, pages 3489–3489. PMLR, 2022.

G. F. Franklin, J. D. Powell, M. L. Workman, et al. *Digital control of dynamic systems*, volume 3. Addison-wesley Menlo Park, CA, 1998.

J. Fu, K. Luo, and S. Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

J. Fu, K. Luo, and S. Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018a.

J. Fu, A. Singh, D. Ghosh, L. Yang, and S. Levine. Variational inverse control with events: A general framework for data-driven reward definition. *Advances in neural information processing systems*, 31, 2018b.

J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

S. Fujimoto and S. S. Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019.

O. Gottesman, F. Johansson, J. Meier, J. Dent, D. Lee, S. Srinivasan, L. Zhang, Y. Ding, D. Wihl, X. Peng, et al. Evaluating reinforcement learning algorithms in observational health settings. *arXiv preprint arXiv:1805.12298*, 2018.

M. Grant, S. Boyd, and Y. Ye. CVX: Matlab software for disciplined convex programming, 2009.

A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan. Inverse reward design. *Advances in neural information processing systems*, 30, 2017.

A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft. Safe exploration for reinforcement learning. In *ESANN*, pages 143–148. Citeseer, 2008.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

J. P. Hespanha. *Linear systems theory*. Princeton university press, 2018.

M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

J. Ho and S. Ermon. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*, 2016a.

J. Ho and S. Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016b.

H. Hu, Y. Yang, Q. Zhao, and C. Zhang. The provable benefits of unsupervised data sharing for offline reinforcement learning. In *International Conference on Learning Representations*, 2023.

B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems*, 31, 2018.

J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

Y. Jin, Z. Yang, and Z. Wang. Is pessimism provably efficient for offline rl? In *International Conference on Machine Learning*, pages 5084–5096. PMLR, 2021.

T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

S. M. Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

H. K. Khalil and J. W. Grizzle. *Nonlinear systems*, volume 3. Prentice hall Upper Saddle River, NJ, 2002.

S. M. Khansari-Zadeh. Lasa handwriting dataset. `https://bitbucket.org/khansari/lasahandwritingdataset`, 2019.

S. M. Khansari-Zadeh and A. Billard. Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

S. M. Khansari-Zadeh and A. Billard. Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics and Autonomous Systems*, 62(6):752–765, 2014.

S. M. Khansari-Zadeh and O. Khatib. Learning potential functions from human demonstrations with encapsulated dynamic and compliant behaviors. *Autonomous Robots*, 41 (1):45–69, 2017.

O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020.

R. Kidambi, J. Chang, and W. Sun. Mobile: Model-based imitation learning from observation alone. *Advances in Neural Information Processing Systems*, 34:28598–28611, 2021.

G.-H. Kim, S. Seo, J. Lee, W. Jeon, H. Hwang, H. Yang, and K.-E. Kim. Demodice: Offline imitation learning with supplementary imperfect demonstrations. In *International Conference on Learning Representations*, 2021.

K. Kim, Y. Gu, J. Song, S. Zhao, and S. Ermon. Domain adaptive imitation learning. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2020.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

K. Konyushkova, K. Zolna, Y. Aytar, A. Novikov, S. Reed, S. Cabi, and N. de Fre-

itas. Semi-supervised reward learning for offline reinforcement learning. *arXiv preprint arXiv:2012.06899*, 2020.

I. Kostrikov, A. Nair, and S. Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2021.

A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020a.

A. Kumar, J. Hong, A. Singh, and S. Levine. Should i run offline reinforcement learning or behavioral cloning? In *International Conference on Learning Representations*, 2022.

V. Kumar, D. Hoeller, B. Sundaralingam, J. Tremblay, and S. Birchfield. Joint space control via deep reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3619–3626. IEEE, 2020b.

R. Laroche, P. Trichelair, and R. T. Des Combes. Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, pages 3652–3661. PMLR, 2019.

H. Le, C. Voloshin, and Y. Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712. PMLR, 2019.

J. Lee, N. Mansard, and J. Park. Intermediate desired value approach for task transition of robots in kinematic control. *IEEE Transactions on Robotics*, 28(6):1260–1277, 2012.

J. Lee, C. Paduraru, D. J. Mankowitz, N. Heess, D. Precup, K.-E. Kim, and A. Guez. Coptidice: Offline constrained reinforcement learning via stationary distribution correction estimation. In *International Conference on Learning Representations*, 2022a.

M. A. Lee, C. Florensa, J. Tremblay, N. Ratliff, A. Garg, F. Ramos, and D. Fox. Guided uncertainty-aware policy optimization: Combining learning and model-based strategies for sample-efficient policy learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7505–7512. IEEE, 2020.

S. Lee, Y. Seo, K. Lee, P. Abbeel, and J. Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Conference on Robot Learning*, pages 1702–1712. PMLR, 2022b.

J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg. Ai safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

S. Levine and V. Koltun. Continuous inverse optimal control with locally optimal examples. In *International Conference on Machine Learning 2012*, 2012.

S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

A. Li, C.-A. Cheng, B. Boots, and M. Egerstedt. Stable, concurrent controller composition for multi-objective robotic tasks. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 1144–1151. IEEE, 2019a.

A. Li, M. Mukadam, M. Egerstedt, and B. Boots. Multi-objective policy generation for multi-robot systems using Riemannian motion policies. In *the 19th International Symposium on Robotics Research*, 2019b.

A. Li, C.-A. Cheng, M. A. Rana, M. Xie, K. Van Wyk, N. Ratliff, and B. Boots. RMP2: A structured composable policy class for robot learning. *Robotics: Science and Systems*, 2021.

A. Li, B. Boots, and C.-A. Cheng. Mahalo: Unifying offline reinforcement learning and imitation learning from observations. *arXiv preprint arXiv:2303.17156*, 2023a.

A. Li, D. Misra, A. Kolobov, and C.-A. Cheng. Survival instinct in offline reinforcement learning. In *Advances in neural information processing systems*, 2023b.

J. Li, X. Hu, H. Xu, J. Liu, X. Zhan, Q.-S. Jia, and Y.-Q. Zhang. Mind the gap: Offline policy optimization for imperfect rewards. In *International Conference on Learning Representations*, 2023c.

E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR, 2018.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

F. Liu, Z. Ling, T. Mu, and H. Su. State alignment-based imitation learning. In *International Conference on Learning Representations*, 2019.

Y. Liu, A. Swaminathan, A. Agarwal, and E. Brunskill. Provably good batch off-policy reinforcement learning without great exploration. *Advances in neural information processing systems*, 33:1264–1274, 2020.

Z. Liu, Z. Guo, H. Lin, Y. Yao, J. Zhu, Z. Cen, H. Hu, W. Yu, T. Zhang, J. Tan, et al. Datasets and benchmarks for offline safe reinforcement learning. *arXiv preprint arXiv:2306.09303*, 2023a.

Z. Liu, Z. Guo, Y. Yao, Z. Cen, W. Yu, T. Zhang, and D. Zhao. Constrained decision transformer for offline safe reinforcement learning. *arXiv preprint arXiv:2302.07351*, 2023b.

M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations*, 2018.

K. M. Lynch and F. C. Park. *Modern robotics*. Cambridge University Press, 2017.

X. Ma, Z. Liang, L. Xia, J. Zhang, J. Blanchet, M. Liu, Q. Zhao, and Z. Zhou. Distributionally robust offline reinforcement learning with linear function approximation. *arXiv preprint arXiv:2209.06620*, 2022a.

Y. Ma, A. Shen, D. Jayaraman, and O. Bastani. Versatile offline imitation from observations and examples via regularized state-occupancy matching. In *International Conference on Machine Learning*, pages 14639–14663. PMLR, 2022b.

H. Mania, A. Guy, and B. Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

X. Meng, N. Ratliff, Y. Xiang, and D. Fox. Neural autonomous navigation with Riemannian motion policy. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8860–8866. IEEE, 2019.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

B. Morris, M. J. Powell, and A. D. Ames. Sufficient conditions for the Lipschitz continuity of QP-based multi-objective control of humanoid robots. In *IEEE Conference on Decision and Control*, pages 2920–2926. IEEE, 2013.

M. Mukadam, C.-A. Cheng, D. Fox, B. Boots, and N. Ratliff. Riemannian motion policy fusion through learnable Lyapunov function reshaping. In *Conference on Robot Learning*, pages 204–219, 2019.

R. Munos. Error bounds for approximate policy iteration. In *International Conference on Machine Learning*, volume 3, pages 560–567. Citeseer, 2003.

R. Munos and C. Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(5), 2008.

J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal. Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757, 2008.

K. S. Narendra and J. Balakrishnan. A common Lyapunov function for stable lti systems with commuting a-matrices. *IEEE Transactions on Automatic Control*, 39(12):2469–2471, 1994.

K. Neumann and J. J. Steil. Learning robot motions with stable dynamical systems under diffeomorphic transformations. *Robotics and Autonomous Systems*, 70:1–15, 2015.

A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *the 17th International Conf. on Machine Learning*, 2000.

A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, pages 278–287. Citeseer, 1999.

A. Y. Ng, S. Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, volume 1, page 2, 2000.

H.-A. Nguyen and C.-A. Cheng. Provable reset-free reinforcement learning by no-regret reduction. *arXiv preprint arXiv:2301.02389*, 2023.

K. Ogata. *Discrete-time control systems*. Prentice-Hall, Inc., 1995.

T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

A. Pan, K. Bhatia, and J. Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. In *International Conference on Learning Representations*, 2022.

K. Panaganti, Z. Xu, D. Kalathil, and M. Ghavamzadeh. Robust reinforcement learning using offline data. In *Advances in Neural Information Processing Systems*, 2022.

A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann. Probabilistic movement primitives. *Advances in neural information processing systems*, 26, 2013.

A. Paraschos, R. Lioutikov, J. Peters, and G. Neumann. Probabilistic prioritization of movement primitives. *IEEE Robotics and Automation Letters*, 2(4):2294–2301, 2017.

P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *2009 IEEE International Conference on Robotics and Automation(ICRA)*, pages 763–768. IEEE, 2009.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

S. Paternain, M. Calvo-Fullana, L. F. Chamon, and A. Ribeiro. Safe policies for reinforcement learning via primal-dual methods. *IEEE Transactions on Automatic Control*, 2022.

C. Paxton, N. Ratliff, C. Eppner, and D. Fox. Representing robot task plans as robust logical-dynamical systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

N. Perrin and P. Schlehuber-Caissier. Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems. *Systems & Control Letters*, 96:51–59, 2016.

J. Peters, M. Mistry, F. Udwadia, J. Nakanishi, and S. Schaal. A unifying framework for robot control with redundant dofs. *Autonomous Robots*, 24(1):1–12, 2008.

D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt. The robotarium: A remotely accessible swarm robotics research testbed. In *IEEE International Conference on Robotics and Automation*, pages 1699–1706. IEEE, 2017.

N. Polosky, B. C. Da Silva, M. Fiterau, and J. Jagannath. Constrained offline policy optimization. In *International Conference on Machine Learning*, pages 17801–17810. PMLR, 2022.

D. A. Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, pages 305–313, 1988.

I. Radosavovic, X. Wang, L. Pinto, and J. Malik. State-only imitation learning for dexterous

manipulation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7865–7871. IEEE, 2021.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

M. A. Rana, A. Li, H. Ravichandar, M. Mukadam, S. Chernova, D. Fox, B. Boots, and N. Ratliff. Learning reactive motion policies in multiple task spaces from human demonstrations. In *Conference on Robot Learning*, 2019.

M. A. Rana, A. Li, D. Fox, B. Boots, F. Ramos, and N. Ratliff. Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems. In *Learning for Dynamics and Control*, pages 630–639. PMLR, 2020a.

M. A. Rana, A. Li, D. Fox, S. Chernova, B. Boots, and N. Ratliff. Towards coordinated robot motions: End-to-end learning of motion policies on transform trees. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7792–7799. IEEE, 2020b.

P. Rashidinejad, B. Zhu, C. Ma, J. Jiao, and S. Russell. Bridging offline reinforcement learning and imitation learning: A tale of pessimism. *Advances in Neural Information Processing Systems*, 34:11702–11716, 2021.

N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018.

N. D. Ratliff, K. Van Wyk, M. Xie, A. Li, and M. A. Rana. Generalized nonlinear and finsler geometry for robotics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10206–10212. IEEE, 2021.

H. C. Ravichandar and A. Dani. Learning position and orientation dynamics from demonstrations via contraction analysis. *Autonomous Robots*, 43(4):897–912, 2019.

M. Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005: 16th European Con-*

ference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16, pages 317–328. Springer, 2005.

M. Rigter, B. Lacerda, and N. Hawes. Rambo-RL: Robust adversarial model-based offline reinforcement learning. *Advances in neural information processing systems*, 2022.

S. Ross and D. Bagnell. Efficient reductions for imitation learning. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings.

S. Ross and J. A. Bagnell. Reinforcement and imitation learning via interactive no-regret learning, 2014.

S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

R. Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

S. Sastry. *Nonlinear systems: analysis, stability, and control*, volume 10. Springer Science & Business Media, 2013.

K. Schmeckpeper, O. Rybkin, K. Daniilidis, S. Levine, and C. Finn. Reinforcement learning with videos: Combining offline observations with interaction. In *Conference on Robot Learning*, pages 339–354. PMLR, 2021.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

S. Shaw, B. Abbatematteo, and G. Konidaris. Rmps for safe impedance control in contact-rich manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2707–2713. IEEE, 2022.

L. Shi and Y. Chi. Distributionally robust model-based offline reinforcement learning with near-optimal sample complexity. *arXiv preprint arXiv:2208.05767*, 2022.

D. Shin, A. D. Dragan, and D. S. Brown. Benchmarks and algorithms for offline preference-based reward learning. *Transactions on Machine Learning Research*, 2023.

B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

A. Singh, L. Yang, C. Finn, and S. Levine. End-to-end robotic reinforcement learning without reward engineering. In *Robotics: Science and Systems*, 2019.

A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar, and S. Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. In *Conference on Robot Learning*. PMLR, 2020.

J.-J. E. Slotine, W. Li, et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991.

M. Smith, L. Maystre, Z. Dai, and K. Ciosek. A strong baseline for batch imitation learning. *arXiv preprint arXiv:2302.02788*, 2023.

A. Stooke, J. Achiam, and P. Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. *arXiv preprint arXiv:2007.03964*, 2020.

W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell. Deeply AggreVaTeD: Differentiable imitation learning for sequential prediction. In D. Precup and Y. W. Teh,

editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3309–3318, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4950–4957, 2018.

F. Torabi, G. Warnell, and P. Stone. Adversarial imitation learning from state-only demonstrations. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2229–2231, 2019a.

F. Torabi, G. Warnell, and P. Stone. Recent advances in imitation learning from observation. In *International Joint Conference on Artificial Intelligence*, 2019b.

M. Turchetta, A. Kolobov, S. Shah, A. Krause, and A. Agarwal. Safe reinforcement learning via curriculum induction. *Advances in Neural Information Processing Systems*, 33:12151–12162, 2020.

M. Uehara and W. Sun. Pessimistic model-based offline reinforcement learning under partial coverage. In *International Conference on Learning Representations*, 2022.

J. Urain, M. Ginesi, D. Tateo, and J. Peters. Imitationflow: Learning deep stable stochastic dynamic systems by normalizing flows. *arXiv preprint arXiv:2010.13129*, 2020.

J. Urain, A. Li, P. Liu, C. D'Eramo, and J. Peters. Composable energy policies for reactive motion generation and reinforcement learning. In *Robotics: Science and Systems*, 2021.

K. Van Wyk, M. Xie, A. Li, M. A. Rana, B. Babich, B. Peele, Q. Wan, I. Akinola, B. Sundaralingam, D. Fox, et al. Geometric fabrics: Generalizing classical mechanics to capture the physics of behavior. *IEEE Robotics and Automation Letters*, 2022.

A. Venkatraman, B. Boots, M. Hebert, and J. A. Bagnell. Data as demonstrator with applications to system identification. In *ALR Workshop, NIPS*, 2014.

L. Vu and D. Liberzon. Common Lyapunov functions for families of commuting nonlinear systems. *Systems & control letters*, 54(5):405–416, 2005.

A. Wachi and Y. Sui. Safe reinforcement learning in constrained markov decision processes. In *International Conference on Machine Learning*, pages 9797–9806. PMLR, 2020.

N. C. Wagener, B. Boots, and C.-A. Cheng. Safe reinforcement learning using advantage-based intervention. In *International Conference on Machine Learning*, pages 10630–10640. PMLR, 2021.

R. Wang, C. Ciliberto, P. V. Amadori, and Y. Demiris. Random expert distillation: Imitation learning via expert policy support estimation. In *International Conference on Machine Learning*, pages 6536–6544. PMLR, 2019.

G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

B. Wingo, C.-A. Cheng, M. Murtaza, M. Zafar, and S. Hutchinson. Extending Riemmanian motion policies to a class of underactuated wheeled-inverted-pendulum robots. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3967–3973. IEEE, 2020.

F. Wu, L. Li, H. Zhang, B. Kailkhura, K. Kenthapadi, D. Zhao, and B. Li. Copa: Certifying robust policies for offline reinforcement learning against poisoning attacks. In *International Conference on Learning Representations*, 2023.

Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

M. Xie, A. Handa, S. Tyree, D. Fox, H. Ravichandar, N. D. Ratliff, and K. Van Wyk. Neural geometric fabrics: Efficiently learning high-dimensional policies from demonstration. In *Conference on Robot Learning*, pages 1355–1367. PMLR, 2023.

T. Xie, C.-A. Cheng, N. Jiang, P. Mineiro, and A. Agarwal. Bellman-consistent pessimism for offline reinforcement learning. *Advances in neural information processing systems*, 34: 6683–6694, 2021.

T. Xie, M. Bhardwaj, N. Jiang, and C.-A. Cheng. Armor: A model-based framework for improving arbitrary baseline policies with offline data. *arXiv preprint arXiv:2211.04538*, 2022.

H. Xu, X. Zhan, H. Yin, and H. Qin. Discriminator-weighted offline imitation learning from suboptimal demonstrations. In *International Conference on Machine Learning*, pages 24725–24742. PMLR, 2022a.

H. Xu, X. Zhan, and X. Zhu. Constraints penalized q-learning for safe offline reinforcement learning. In *AAAI*, 2022b.

W. Xu, Y. Ma, K. Xu, H. Bastani, and O. Bastani. Uniformly conservative exploration in reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 10856–10870. PMLR, 2023.

C. Yang, X. Ma, W. Huang, F. Sun, H. Liu, J. Huang, and C. Gan. Imitation learning from observations by minimizing inverse dynamics disagreement. *Advances in neural information processing systems*, 32, 2019.

R. Yang, C. Bai, X. Ma, Z. Wang, C. Zhang, and L. Han. Rorl: Robust offline reinforcement learning via conservative smoothing. In *Advances in Neural Information Processing Systems*, 2022.

D. Yarats, D. Brandfonbrener, H. Liu, M. Laskin, P. Abbeel, A. Lazaric, and L. Pinto. Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. *arXiv preprint arXiv:2201.13425*, 2022.

T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020a.

T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020b.

T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems*, 34:28954–28967, 2021.

T. Yu, A. Kumar, Y. Chebotar, K. Hausman, C. Finn, and S. Levine. How to leverage unlabeled data in offline reinforcement learning. In *International Conference on Machine Learning*, pages 25611–25635. PMLR, 2022.

W. Yu, C. K. Liu, and G. Turk. Policy transfer with strategy optimization. In *International Conference on Learning Representations*, 2019.

S. Yue, G. Wang, W. Shao, Z. Zhang, S. Lin, J. Ren, and J. Zhang. CLARE: Conservative model-based reward learning for offline inverse reinforcement learning. In *International Conference on Learning Representations*, 2023.

W. Zhan, B. Huang, A. Huang, N. Jiang, and J. Lee. Offline reinforcement learning with realizability and single-policy concentrability. In *Conference on Learning Theory*, pages 2730–2775. PMLR, 2022.

X. Zhang, Y. Chen, X. Zhu, and W. Sun. Corruption-robust offline reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 5757–5773. PMLR, 2022.

W. Zhi, I. Akinola, K. Van Wyk, N. D. Ratliff, and F. Ramos. Global and reactive motion generation with geometric fabric command sequences. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 939–945. IEEE, 2023.

G. Zhou, L. Ke, S. Srinivasa, A. Gupta, A. Rajeswaran, and V. Kumar. Real world offline reinforcement learning with realistic data source. *arXiv preprint arXiv:2210.06479*, 2022.

Z. Zhou, Z. Zhou, Q. Bai, L. Qiu, J. Blanchet, and P. Glynn. Finite-sample regret bound for distributionally robust offline tabular reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3331–3339. PMLR, 2021.

Z. Zhu, K. Lin, B. Dai, and J. Zhou. Off-policy imitation learning from observations. *Advances in Neural Information Processing Systems*, 33:12402–12413, 2020.

B. D. Ziebart, A. Maas, J. A. D. Bagnell, and A. Dey. Maximum entropy inverse reinforcement learning. In *Proceeding of AAAI 2008*, July 2008a.

B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. Maximum entropy inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008b.

K. Zolna, A. Novikov, K. Konyushkova, C. Gulcehre, Z. Wang, Y. Aytar, M. Denil, N. de Freitas, and S. Reed. Offline learning from demonstrations and unlabeled experience. *arXiv preprint arXiv:2011.13885*, 2020.